Inmanta Documentation

Release 8.0.0.dev20240505015437

Inmanta NV

May 05, 2024

CONTENTS

1 Quickstart 3

2 Installation 13

3 Architecture 33

4 Language Reference 37

5 Model developer documentation 51

6 Platform developer documentation 105

7 Inmanta Lifecycle Service Manager 119

8 Administrator documentation 187

9 Frequently asked questions 233

10 Glossary 235

11 Inmanta Reference 239

12 Troubleshooting 419

13 Changelog 431

14 PDF version 433

Python Module Index 435

Index 437

Welcome to the Inmanta Service Orchestrator documentation!

Inmanta empowers telecom operators and service providers to speed up service delivery and reduce the total cost of ownership through efficient, end-to-end automation. No longer is automation limited to silos and vendor-specific solutions – you can now integrate with various domains and best-in-class components from any vendor.

Inmanta Service Orchestrator is an automation and orchestration tool to efficiently deploy and manage your end-to-end services across physical and virtual domains and multi-vendor environments. Inmanta's open and extensible micro-services architecture combined with powerful, intent-based service modelling provides the flexibility and efficiency to rapidly create, customize and roll-out new services, while eliminating costly operational errors.

The key characteristics of Inmanta Service Orchestrator are:

- **End-to-end**: Inmanta Service Orchestrator ensures end-to-end consistency, higher flexibility and a shorter time to cash by enabling end-to-end automation of all service delivery aspects:
  - Multi-domain: designed to interact across physical and virtual domains, such as WAN, edge, access network, NFV, cloud, containers, and datacenter.
  - Holistic: A single, unifying automation solution, providing service orchestration, network orchestration, NFV orchestration (NFVO), as well as generic VNF management (gVNFM), cloud orchestration and configuration management. No other automation tools required.
  - Full lifecyle: Manage advanced service lifecycle, covering creation, on-boarding, provisioning, modification, scaling, upgrading and decommissioning.
- **Intent-based programmability**: Inmanta optimizes service development and maintenance for telecom operators and service providers through its unifying, model-driven methodology for intent-based orchestration.
  - Inmanta's powerful domain-specific language (DSL) simplifies service creation and management, and is based on infrastructure as code (IaC) principles to provide a unified way to automate multi-domain and multi-vendor services. The embedded DSL enables the development of modular building blocks that make abstraction of low-level details, enabling re-usability across use cases.
  - Inmanta's intent-based programmability provides out-of-the-box self-healing, safe roll-back, detailed dry run and seamless service upgrades for enhanced stability and resilience.
- **Vendor agnostic**: Inmanta Service Orchestrator is truly open and vendor agnostic for all network layers, domains and OSS/BSS. Service providers can integrate with 3rd party solutions as well as a wide range of open-source technologies to build a best-in-class, all-encompassing solution.
  - Interoperability through pluggable adapters and open APIs
  - API-ification of orchestrated services to easily plug services into the OSS/BSS environment
  - Support for brownfield environments by fine-grained roll-out

The Inmanta Service Orchestrator product is based on mature technology backed by 15+ years of research and interaction with companies offering telecom and cloud services.

CHAPTER

ONE

QUICKSTART

Inmanta is intended to manage complex infrastructures, often in the cloud or other virtualized environments. In this guide we start simple and manage a 3-node CLOS network with a spine and two leaf switches. First we install containerlab and then configure SR Linux containers using **Inmanta open source orchestrator** and `gNMI`.

1. First, we use *Containerlab* to spin-up Inmanta server and its PostgreSQL database, then three *SR Linux* containers, connected in a CLOS like topology
2. After that, we configure IP addresses and OSPF on them using **Inmanta**.

Note: This guide is meant to quickly set up an Inmanta LAB environment to experiment with. It is not recommended to run this setup in production, as it might lead to instabilities in the long term.

1.1 Prerequisites

Python version 3.11, `Docker`, `Containerlab` and `Inmanta` need to be installed on your machine and our `SR Linux` repository has to be cloned in order to proceed. Please make sure to follow the links below to that end.

1. Install Docker.
2. Install Containerlab.
3. Prepare a development environment by creating a *python virtual environment* and installing Inmanta:

```
mkdir -p ~/.virtualenvs
python3 -m venv ~/.virtualenvs/srlinux
source ~/.virtualenvs/srlinux/bin/activate
pip install inmanta
```

4. Clone the SR Linux examples repository:

```
git clone https://github.com/inmanta/examples.git
```

5. Change directory to *SR Linux* examples:

```
cd examples/Networking/SR\ Linux/
```

This folder contains a **project.yml**, which looks like this:

```
name: SR Linux Examples
description: Provides examples for the SR Linux module
author: Inmanta
author_email: code@inmanta.com
license: ASL 2.0
copyright: 2022 Inmanta
modulepath: libs
```

(continues on next page)

(continued from previous page)

```
downloadpath: libs
pip:
  index_url: url: https://packages.inmanta.com/public/quickstart/python/simple/
```

- The `modulepath` setting defines that modules will be stored in `libs` directory.
- The `repo` setting points to one or more Git repositories containing Inmanta modules.
- The `requires` setting is used to pin versions of modules, otherwise the latest version is used.

1. Install the required modules inside the *SR Linux* folder:

   ```
   inmanta project install
   ```

   Note: should you face any errors at this stage, please contact us.

In the next sections we will showcase how to set up and configure `SR Linux` devices.

1.2 Setting up the LAB

Go to the *SR Linux* folder and then *containerlab* to spin-up the containers:

```
cd examples/Networking/SR\ Linux/containerlab
sudo docker pull ghcr.io/nokia/srlinux:latest
sudo clab deploy -t topology.yml
```

Containerlab will spin-up:

1. an *Inmanta* server
2. a *PostgreSQL* Database server
3. Three *SR Linux* network operating systems.

Depending on your system's horsepower, give them a few seconds/minutes to fully boot-up.

1.3 Connecting to the containers

At this stage, you should be able to view the **Web Console** by navigating to:

http://172.30.0.3:8888/console

To get an interactive shell to the Inmanta server:

```
docker exec -it clab-srlinux-inmanta-server /bin/bash
```

In order to connect to *SR Linux* containers, there are two options:

1. Using Docker:

```
docker exec -it clab-srlinux-spine sr_cli
# or
docker exec -it clab-srlinux-leaf1 sr_cli
# or
docker exec -it clab-srlinux-leaf2 sr_cli
```

2. Using SSH (username *admin* and password *NokiaSrl1!*):

```
ssh admin@clab-srlinux-spine
ssh admin@clab-srlinux-leaf1
ssh admin@clab-srlinux-leaf2
```

The output should look something like this:

```
Welcome to the srlinux CLI.
Type 'help' (and press <ENTER>) if you need any help using this.


--{ running }--[  ]--
A:spine#
```

Optionally, you can enter the *configuration mode* by typing:

```
enter candidate
```

Exit the session by typing:

```
quit
```

Now that we have the needed containers, we will need to go up a directory where the project files exist:

```
cd ..
```

Note: The rest of the this guide assumes commands are executed from the root path of the *SR Linux* folder, unless noted otherwise.

1.4 Create an Inmanta project and an environment

A project is a collection of related environments. (e.g. development, testing, production, qa,...). We need to have an environment to manage our infrastructure. An environment is a collection of resources, such as servers, switches, routers, etc.

There are **two ways** to create a project and an environment:

1. Using Inmanta CLI (**recommended**):

```
# Create a project called test
inmanta-cli --host 172.30.0.3 project create -n test
# Create an environment called SR_Linux
inmanta-cli --host 172.30.0.3 environment create -p test -n SR_Linux --save
```

The first option, `inmanta-cli`, will automatically create a `.inmanta` file that contains the required information about the server and environment ID. The compiler uses this file to find the server and to export to the right environment.

2. Using the Web Console: Connect to the Inmanta container http://172.30.0.3:8888/console, click on the *Create new environment* button, provide a name for the project and the environment then click *submit*.

If you have chosen the second option, the Web Console, you need to copy the environment ID for later use, either:

- from the URL, e.g. ec05d6d9-25a4-4141-a92f-38e24a12b721 from the http://172.30.0.3:8888/console/desiredstate?env=ec05d6d9-25a4-4141-a92f-38e24a12b721.
- or by clicking on the *Settings* pane, then in the *Environment* tab, scroll down all the way to the bottom of the page and copy the environment ID.

1.5 Configuring SR Linux

There are a bunch of examples present inside the *SR Linux* folder of the *examples* repository that you have cloned in the previous step, setting up the *lab*.

In this guide, we will showcase two examples on a small **CLOS** topology to get you started:

1. interface configuration.
2. OSPF configuration.

It could be useful to know that Inmanta uses the `gNMI` protocol to interface with `SR Linux` devices.

Note: In order to make sure that everything is working correctly, run `inmanta compile`. This will ensure that the modules are in place and the configuration is valid. If you face any errors at this stage, please contact us.

1.6 SR Linux interface configuration

The interfaces.cf file contains the required configuration model to set IP addresses on point-to-point interfaces between the `spine`, `leaf1` and `leaf2` devices according to the aforementioned topology.

Let's have a look at the partial configuration model:

```
import srlinux
import srlinux::interface as srinterface
import srlinux::interface::subinterface as srsubinterface
import srlinux::interface::subinterface::ipv4 as sripv4
import yang



######## Leaf 1 ########

leaf1 = srlinux::GnmiDevice(
    auto_agent = true,
    name = "leaf1",
    mgmt_ip = "172.30.0.210",
    yang_credentials = yang::Credentials(
        username = "admin",
        password = "NokiaSrl1!"
    )
)

leaf1_eth1 = srlinux::Interface(
    device = leaf1,
    name = "ethernet-1/1",
    mtu = 9000,
    subinterface = [leaf1_eth1_subint]
)

leaf1_eth1_subint = srinterface::Subinterface(
    parent_interface = leaf1_eth1,
    x_index = 0,
    ipv4 = leaf1_eth1_subint_address
)

```

(continues on next page)

(continued from previous page)

```
34 leaf1_eth1_subint_address = srsubinterface::Ipv4(
35     parent_subinterface = leaf1_eth1_subint,
36     address = sripv4::Address(
37         parent_ipv4 = leaf1_eth1_subint_address,
38         ip_prefix = "10.10.11.2/30"
39     )
40 )
```

- Lines 1-5 import the required modules/packages.
- Lines 11-19 instantiate the device; `GnmiDevice` object and set the required parameters.
- Lines 21-26 instantiate the `Interface` object by selecting the parent interface, `ethernet-1/1` and setting the MTU to 9000.
- Lines 28-32 instantiate the `Subinterface` object, link to the parent interface object, set an *index* and link to the child `Ipv4` object.
- Lines 34-40 instantiate the `Ipv4` object, link to the parent `Subinterface` object, set the IP address and prefix.

The rest of the configuration model follows the same method for `leaf2` and `spine` devices, with the only difference being the `spine` having two interfaces, subinterfaces and IP addresses.

Now, we can deploy the model by referring to *Deploy the configuration model* section.

1.7 SR Linux OSPF configuration

The ospf.cf file contains the required configuration model to first set IP addresses on point-to-point interfaces between the `spine`, `leaf1` and `leaf2` devices according to the aforementioned topology and then configure `OSPF` between them.

This model build on top of the `interfaces` model that was discussed in *SR Linux interface configuration*. It first *imports* the required packages, then configures `interfaces` on all the devices and after that, adds the required configuration model for `OSPF`.

Let's have a look at the partial configuration model:

```
import srlinux
import srlinux::interface as srinterface
import srlinux::interface::subinterface as srsubinterface
import srlinux::interface::subinterface::ipv4 as sripv4
import srlinux::network_instance as srnetinstance
import srlinux::network_instance::protocols as srprotocols
import srlinux::network_instance::protocols::ospf as srospf
import srlinux::network_instance::protocols::ospf::instance as srospfinstance
import srlinux::network_instance::protocols::ospf::instance::area as srospfarea
import yang



######## Leaf 1 ########

leaf1 = srlinux::GnmiDevice(
    auto_agent = true,
    name = "leaf1",
    mgmt_ip = "172.30.0.210",
    yang_credentials = yang::Credentials(
        username = "admin",
```

(continues on next page)

(continued from previous page)

```
        password = "admin"
    )
)

# |interface configuration| #

leaf1_eth1 = srlinux::Interface(
    device = leaf1,
    name = "ethernet-1/1",
    mtu = 9000,
    subinterface = [leaf1_eth1_subint]
)

leaf1_eth1_subint = srinterface::Subinterface(
    parent_interface = leaf1_eth1,
    x_index = 0,
    ipv4 = leaf1_eth1_subint_address
)

leaf1_eth1_subint_address = srsubinterface::Ipv4(
    parent_subinterface = leaf1_eth1_subint,
    address = sripv4::Address(
        parent_ipv4 = leaf1_eth1_subint_address,
        ip_prefix = "10.10.11.2/30"
    )
)

# |network instance| #

leaf1_net_instance = srlinux::NetworkInstance(
    device = leaf1,
    name = "default",
)

leaf1_net_instance_int1 = srnetinstance::Interface(
    parent_network_instance = leaf1_net_instance,
    name = "ethernet-1/1.0"
)

# |OSPF| #

leaf1_protocols = srnetinstance::Protocols(
    parent_network_instance = leaf1_net_instance,
    ospf = leaf1_ospf
)

leaf1_ospf_instance = srospf::Instance(
        parent_ospf = leaf1_ospf,
        name = "1",
        router_id = "10.20.30.210",
        admin_state = "enable",
        version = "ospf-v2"
)

leaf1_ospf = srprotocols::Ospf(
    parent_protocols = leaf1_protocols,
```

(continues on next page)

(continued from previous page)

```
78      instance = leaf1_ospf_instance
79  )
80
81  leaf1_ospf_area = srospfinstance::Area(
82      parent_instance = leaf1_ospf_instance,
83      area_id = "0.0.0.0",
84  )
85
86  leaf1_ospf_int1 = srospfarea::Interface(
87      parent_area = leaf1_ospf_area,
88      interface_name = "ethernet-1/1.0",
89  )
```

- Lines 1-10 import the required modules/packages.
- Lines 16-24 instantiate the device; `GnmiDevice` object and set the required parameters.
- Lines 28-33 instantiate the `Interface` object by selecting the parent interface, `ethernet-1/1` and setting the MTU to 9000.
- Lines 35-39 instantiate the `Subinterface` object, link to the parent interface object, set an *index* and link to the child `Ipv4` object.
- Lines 41-47 instantiate the `Ipv4` object, link to the parent `Subinterface` object, set the IP address and prefix.
- Lines 51-54 instantiate `NetworkInstance` object, set the name to `default`.
- Lines 56-59 instantiate a network instance `Interface` object, link to the `default` network instance object and use `ethernet-1/1.0` as the interface.
- Lines 63-66 instantiate the `Protocols` object, link to the `default` network instance object and link to the `OSPF` object which we will create shortly.
- Lines 68-74 instantiate an OSPF instance and OSPF `Instance`, link to the `OSPF instance`, provide a name, router ID, admin state and version.
- Lines 76-79 instantiate an `OSPF` object, link to the `Protocols` object and link to the `OSPF instance`.
- Lines 81-84 instantiate an `Area` object, link to the `OSPF instance` and provide the area ID.
- Lines 86-89 instantiate an area `Interface` object, link to the `OSPF area` object and activates the OSPF on `ethernet-1/1.0` interface.

The rest of the configuration model follows the same method for `leaf2` and `spine` devices, with the only difference being the `spine` having two interfaces, subinterfaces and IP addresses and OSPF interface configuration.

Now, we can deploy the model by referring to *Deploy the configuration model* section.

1.8 Deploy the configuration model

To deploy the project, we must first register it with the management server by creating a project and an environment. We have covered this earlier at *Create an Inmanta project and an environment* section.

Export the `interfaces` configuration model to the Inmanta server:

```
inmanta -vvv export -f interfaces.cf
# or
inmanta -vvv export -f interfaces.cf -d
```

Export the `OSPF` configuration model to the Inmanta server:

```
inmanta -vvv export -f ospf.cf
# or
inmanta -vvv export -f ospf.cf -d
```

Note: The `-vvv` option sets the output of the compiler to very verbose. The `-d` option instructs the server to immediately start the deploy.

When the model is sent to the server, it will start deploying the configuration. To track progress, you can go to the web-console, select the *test* project and then the *SR_Linux* environment and click on `Resources` tab on the left pane to view the progress.

When the deployment is complete, you can verify the configuration using the commands provided in *Verifying the configuration* section.

If the deployment fails for some reason, consult the *troubleshooting page* to investigate the root cause of the issue.

1.9 Verifying the configuration

After a successful deployment, you can connect to `SR Linux` devices and verify the configuration.

Pick all or any of the devices you like, connect to them as discussed in *Connecting to the containers* section and check the configuration:

```
show interface ethernet-1/1.0
show network-instance default protocols ospf neighbor
show network-instance default route-table ipv4-unicast summary
info flat network-instance default
```

1.10 Resetting the LAB environment

To fully clean up or reset the LAB, go to the **containerlab** folder and run the following commands:

```
cd containerlab
sudo clab destroy -t topology.yml
```

This will give you a clean LAB the next time you run:

```
sudo clab deploy -t topology.yml --reconfigure
```

1.11 Reusing existing modules

We host modules to set up and manage many systems on our Github. These are available under https://github.com/inmanta/.

When you use an import statement in your model, Inmanta downloads these modules and their dependencies when you run `inmanta project install`. V2 modules (See *V2 module format*) need to be declared as Python dependencies in addition to using them in an import statement. Some of our public modules are hosted in the v2 format on https://pypi.org/.

1.12 Update the configuration model

The provided configuration models can be easily modified to reflect your desired configuration. Be it a change in IP addresses or adding new devices to the model. All you need to do is to create a new or modify the existing configuration model, say `interfaces.cf` to introduce your desired changes.

For instance, let's change the IP address of interface `ethernet-1/1.0` to *100.0.0.1/24* in the *interfaces.cf* configuration file:

```
import srlinux
import srlinux::interface as srinterface
import srlinux::interface::subinterface as srsubinterface
import srlinux::interface::subinterface::ipv4 as sripv4
import yang



######## Leaf 1 ########

leaf1 = srlinux::GnmiDevice(
    auto_agent = true,
    name = "leaf1",
    mgmt_ip = "172.30.0.210",
    yang_credentials = yang::Credentials(
        username = "admin",
        password = "admin"
    )
)

leaf1_eth1 = srlinux::Interface(
    device = leaf1,
    name = "ethernet-1/1",
    mtu = 9000,
    subinterface = [leaf1_eth1_subint]
)

leaf1_eth1_subint = srinterface::Subinterface(
    parent_interface = leaf1_eth1,
    x_index = 0,
    ipv4 = leaf1_eth1_subint_address
)

leaf1_eth1_subint_address = srsubinterface::Ipv4(
    parent_subinterface = leaf1_eth1_subint,
    address = sripv4::Address(
        parent_ipv4 = leaf1_eth1_subint_address,
        ip_prefix = "100.0.0.1/24"
    )
)
```

Additionally, you can add more SR Linux devices to the *topology.yml* file and explore the possible combinations.

1.13 Modify or Create your own modules

Inmanta enables developers of a configuration model to make it modular and reusable. We have made some videos that can walk you through the entire process in a short time.

Please check our YouTube playlist to get started.

1.13.1 Module layout

A configuration module requires a specific layout:

- The name of the module is determined by the top-level directory. Within this module directory, a `module.yml` file has to be specified.
- The only mandatory subdirectory is the `model` directory containing a file called `_init.cf`. What is defined in the `_init.cf` file is available in the namespace linked with the name of the module. Other files in the model directory create subnamespaces.
- The `files` directory contains files that are deployed verbatim to managed machines.
- The `templates` directory contains templates that use parameters from the configuration model to generate configuration files.
- The `plugins` directory contains Python files that are loaded by the platform and can extend it using the Inmanta API.

```
module
|
|__ module.yml
|
|__ files
|    |__ file1.txt
|
|__ model
|    |__ _init.cf
|    |__ services.cf
|
|__ plugins
|    |__ functions.py
|
|__ templates
     |__ conf_file.conf.tmpl
```

Custom modules should be placed in the `libs` directory of the project.

1.14 Next steps

Model developer documentation

CHAPTER

TWO

INSTALLATION

2.1 Install Inmanta

This page explains how to install the Inmanta orchestrator software and setup an orchestration server. Regardless what platform you installed it on, Inmanta requires at least Python and Git to be installed.

2.1.1 Install the software

Step 1: Install the software

Create a repositories file to point yum to the inmanta service orchestrator release repository. Create a file /etc/yum.repos.d/inmanta.repo with the following content:

```
[inmanta-service-orchestrator-8-stable]
name=inmanta-service-orchestrator-8-stable
baseurl=https://packages.inmanta.com/<token>/inmanta-service-orchestrator-8-stable/
↪rpm/el/8/$basearch
repo_gpgcheck=1
enabled=1
gpgkey=https://packages.inmanta.com/<token>/inmanta-service-orchestrator-8-stable/cfg/
↪gpg/gpg.F4B97D6483D7D2BE.key
gpgcheck=1
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300
pkg_gpgcheck=1
autorefresh=1
type=rpm-md
```

Replace <token> with the token provided with your license.

Use dnf to install the software:

```
sudo dnf install -y inmanta-service-orchestrator-server
```

This command installs the software and all of its dependencies.

Install the license

For the orchestration server to start a license and entitlement file should be loaded into the server. This section describes how to configure the license. The license consists of two files:

- The file with the .license extension is the license file
- The file with the .jwe extension is the entitlement file

Copy both files to the server and store them for example in `/etc/inmanta/license`. If this directory does not exist, create it. Then create a configuration file to point the orchestrator to the license files. Create a file `/etc/inmanta/inmanta.d/license.cfg` with the following content:

```
[license]
license-key=/etc/inmanta/license/<license name>.license
entitlement-file=/etc/inmanta/license/<license name>.jwe
```

Replace `<license name>` with the name of the license you received.

Optional step 2: Setup SSL and authentication

Follow the instructions in *Setting up SSL and authentication* to configure both SSL and authentication. While not mandatory, it is highly recommended you do so.

Step 3: Install PostgreSQL 13

Install the PostgreSQL 13 package included in RHEL. More info in the 'Included in Distribution' section of the postgresql documentation.

RHEL 8

```
sudo dnf module install postgresql:13/server
sudo systemctl enable postgresql
```

RHEL 9

```
sudo dnf install postgresql-server
sudo systemctl enable postgresql
```

Warning: Before moving on to the next step, make sure that the locale used by the system is actually installed. By default, RHEL9 uses the `en_US.UTF-8` locale which can be installed via:

```
sudo dnf install langpacks-en -y
```

Note: If your system uses a different locale, please install the corresponding langpack.

Step 4: Setup a PostgreSQL database for the Inmanta server

Initialize the PostgreSQL server:

```
sudo su - postgres -c "postgresql-setup --initdb"
```

Start the PostgreSQL database and make sure it is started at boot.

```
sudo systemctl enable --now postgresql
```

Create a inmanta user and an inmanta database by executing the following command. This command will request you to choose a password for the inmanta database.

```
sudo -u postgres -i bash -c "createuser --pwprompt inmanta"
sudo -u postgres -i bash -c "createdb -O inmanta inmanta"
```

Change the authentication method for local connections to md5 by changing the following lines in the `/var/lib/pgsql/data/pg_hba.conf` file

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            ident
# IPv6 local connections:
host    all             all             ::1/128                 ident
```

to

```
# IPv4 local connections:
host    all             all             127.0.0.1/32            md5
# IPv6 local connections:
host    all             all             ::1/128                 md5
```

Make sure JIT is disabled for the PostgreSQL database as it might result in poor query performance. To disable JIT, set

```
# disable JIT
jit = off
```

in `/var/lib/pgsql/13/data/postgresql.conf`.

Restart the PostgreSQL server to apply the changes made in the `pg_hba.conf` and `postgresql.conf` files:

```
sudo systemctl restart postgresql
```

Step 5: Set the database connection details

Add a `/etc/inmanta/inmanta.d/database.cfg` file as such that it contains the correct database connection details. That file should look as follows:

```
[database]
host=<ip-address-database-server>
name=inmanta
username=inmanta
password=<password>
```

Replace <password> in the above-mentioned snippet with the password of the inmanta database. By default Inmanta tries to connect to the local server and uses the database inmanta. See the `database` section in the configfile for other options.

Step 6: Set the server address

When virtual machines are started by this server that install the inmanta agent, the correct *server.server-address* needs to be configured. This address is used to create the correct boot script for the virtual machine.

Set this value to the hostname or IP address that other systems use to connect to the server in the configuration file stored at `/etc/inmanta/inmanta.d/server.cfg`.

```
[server]
server-address=<server-ip-address-or-hostname>
```

Note: If you deploy configuration models that modify resolver configuration it is recommended to use the IP address instead of the hostname.

Step 7: Configure ssh of the inmanta user

The inmanta user that runs the server needs a working ssh client. This client is required to checkout git repositories over ssh and if the remote agent is used.

1. Provide the inmanta user with one or more private keys:

   a. Generate a new key with ssh-keygen as the inmanta user: `sudo -u inmanta ssh-keygen -N ""`

   b. Install an exiting key in `/var/lib/inmanta/.ssh/id_rsa`

   c. Make sure the permissions and ownership are set correctly.

   ```
   ls -l /var/lib/inmanta/.ssh/id_rsa

   -rw-------. 1 inmanta inmanta 1679 Mar 21 13:55 /var/lib/inmanta/.ssh/id_rsa
   ```

2. Configure ssh to accept all host keys or white list the hosts that are allowed or use signed host keys (depends on your security requirements). This guide configures ssh client for the inmanta user to accept all host keys. Create `/var/lib/inmanta/.ssh/config` and create the following content:

   ```
   Host *
       StrictHostKeyChecking no
       UserKnownHostsFile=/dev/null
   ```

   Ensure the file belongs to the inmanta user:

   ```
   sudo chown inmanta:inmanta /var/lib/inmanta/.ssh/config
   ```

3. Add the public key to any git repositories and save it to include in configuration models that require remote agents.

4. Test if you can login into a machine that has the public key and make sure ssh does not show you any prompts to store the host key.

Step 8: Configure the server bind address

By default the server only listens on localhost, port 8888. This can be changed by altering the *server.bind-address* and *server.bind-port* options in the `/etc/inmanta/inmanta.d/server.cfg` file.

```
[server]
bind-address=<server-bind-address>
bind-port=<server-bind-port>
```

Step 9: Enable the required Inmanta extensions

Make sure that the required Inmanta extensions are enabled. This is done by adding a configuration file with the following content to `/etc/inmanta/inmanta.d/extensions.cfg`.

```
[server]
enabled_extensions=lsm,ui,support,license
```

This file is also installed by the RPM.

Step 10: Start the Inmanta server

Start the Inmanta server and make sure it is started at boot.

```
sudo systemctl enable --now inmanta-server
```

The web-console is now available on the port and host configured in step 8.

Optional Step 11: Setup influxdb for collection of performance metrics

Follow the instructions in *Performance Metering* to send performance metrics to influxdb. This is only recommended for production deployments.

Optional Step 12: Configure logging

Logging can be configured by following the instructions in *Logging*.

2.2 Install Inmanta with Docker

This page explains how to setup an orchestration server using docker. This guide assumes you already have docker and docker-compose installed on your machine.

2.2.1 Pull the image

Step 1: Log in to Cloudsmith registry

Connect to the Cloudsmith registry using your entitlement token.

```
$ docker login containers.inmanta.com
Username: containers
Password: <your-entitlement-token>

Login Succeeded
$
```

Replace <`your-entitlement-token`> with the entitlement token provided with your license.

Step 2: Pull the image

Use docker pull to get the desired image:

```
docker pull containers.inmanta.com/containers/service-orchestrator:8
```

This command will pull the latest version of the Inmanta Service Orchestrator image.

2.2.2 Start the server with docker-compose

Here is a minimalistic docker-compose file content that can be used to deploy the server on your machine.

```
version: '3'
services:
    postgres:
        container_name: inmanta_db
        image: postgres:13
        environment:
            POSTGRES_USER: inmanta
            POSTGRES_PASSWORD: inmanta
            PGDATA: /var/lib/postgresql/data/pgdata
        networks:
            inm_net:
                ipv4_address: 172.30.0.2
        volumes:
            - type: volume
              source: pgdata
              target: /var/lib/postgresql/data
    inmanta-server:
        container_name: inmanta_orchestrator
        image: containers.inmanta.com/containers/service-orchestrator:8
        ports:
            - 8888:8888
        volumes:
            - ./resources/com.inmanta.license:/etc/inmanta/license/com.inmanta.
↪license
            - ./resources/com.inmanta.jwe:/etc/inmanta/license/com.inmanta.jwe
        networks:
            inm_net:
                ipv4_address: 172.30.0.3
        depends_on:
            - "postgres"
        command: "server --wait-for-host inmanta_db --wait-for-port 5432"
networks:
    inm_net:
        ipam:
            driver: default
            config:
                - subnet: 172.30.0.0/16
volumes:
    pgdata:
```

You can paste this script in a file named *docker-compose.yml* and ensure you have you license files available. With the proposed config, they should be located in a `resources/` folder on the side of the docker-compose file you create, and the license files should be named `com.inmanta.license` and `com.inmanta.jwe`. You can of course update the content of the docker-compose file to match your current configuration. Then bring the containers up by running the following command:

```
docker-compose up
```

You should be able to reach the orchestrator to this address: http://172.30.0.3:8888.

The PostgreSQL server started by the above-mentioned docker-compose file has a named volume `pgdata` attached. This means that no data will be lost when the PostgreSQL container restarts. Pass the `-v` option to the `docker-compose down` to remove the volume.

The default server config included in the container images assumes that the orchestrator can reach a database server with hostname `inmanta_db` and that it can authenticate to it using the username `inmanta` and password `inmanta`. When using a different setup than the one mentioned above, you should overwrite the server config with one matching your needs. You can find more instructions for overwriting the config in a following section, *here*.

Warning: We don't recommend using the setup described above as a production environment. Hosting a database in a container as shown here is not ideal in term of performance, reliability and raises some serious data persistence concerns.

2.2.3 Overwrite default server configuration

By default the server will use the file located in the image at `/etc/inmanta/inmanta.cfg`. If you want to change it, you can copy this file, edit it, then mount it in the container, where the original file was located.

If you use docker-compose, you can simply update this section of the example above:

```
inmanta-server:
    container_name: inmanta_orchestrator
    image: containers.inmanta.com/containers/service-orchestrator:8
    ports:
        - 8888:8888
    volumes:
        - ./resources/com.inmanta.license:/etc/inmanta/license/com.inmanta.license
        - ./resources/com.inmanta.jwe:/etc/inmanta/license/com.inmanta.jwe
        - ./resources/my-server-conf.cfg:/etc/inmanta/inmanta.cfg
```

2.2.4 Starting the ssh server

By default, no ssh server is running in the container. You don't need it to have a functional orchestrator. If you want to enable ssh anyway, for easy access to the orchestrator, you can overwrite the startup command of the container with the following:

```
server-with-ssh
```

If you use docker-compose, it should look like:

```
inmanta-server:
    container_name: inmanta_orchestrator
    ...
    command: "server-with-ssh"
```

Warning: By default, the inmanta user doesn't have any password, if you want to ssh into the container, you also need to set the authorized_keys file for the inmanta user. You can do so by mounting your public key to the following path in the container: /var/lib/inmanta/.ssh/authorized_keys. When starting, the container will make sure that the file has the correct ownership and permissions.

2.2.5 Waiting for the database

Depending on you setup, you might want your container to wait for the database to be ready to accept connections before starting the server (as this one would fail, trying to reach the db). You can do this by adding the following arguments to the startup command of the container:

```
server --wait-for-host <your-db-host> --wait-for-port <your-db-port>
```

If you use docker-compose, it should look like:

```
inmanta-server:
    container_name: inmanta_orchestrator
    ...
    command: "server --wait-for-host <your-db-host> --wait-for-port <your-db-port>"
```

2.2.6 Setting environment variables

You might want your inmanta server to be able to reach some environment variables. There are two ways you can achieve this:

1. Set the environment variables with docker, either using the --env argument or in your docker-compose file. Those variables will be accessible to the inmanta server and any agent it starts, but not to any other process running in the container (if you for example login via ssh to the container and try to install a project again).
2. (Recommended) Set the environment variables in a file and mount it to the following path in the container: /etc/inmanta/env. This file will be loaded when starting the server and for every session that the inmanta user starts in the container.

```
inmanta-server:
    container_name: inmanta_orchestrator
    image: containers.inmanta.com/containers/service-orchestrator:8
    ports:
        - 8888:8888
    volumes:
        - ./resources/com.inmanta.license:/etc/inmanta/license/com.inmanta.license
        - ./resources/com.inmanta.jwe:/etc/inmanta/license/com.inmanta.jwe
        - ./resources/my-server-conf.cfg:/etc/inmanta/inmanta.cfg
        - ./resources/my-env-file:/etc/inmanta/env
```

2.2.7 Changing inmanta user/group id

If you mount a folder of your host in the container, and expect the inmanta user to interact with it, you might face the issue that the inmanta user inside the container doesn't have ownership of the files. You could fix this by changing the ownership in the container, but this change would also be reflected on the host, meaning that you would lose the ownership of you files. This is a very uncomfortable situation. While Podman has been offering the possibility to do a mapping of a user id in the container to a user id on the host at runtime, which would solve our problem here, Docker still doesn't offer this functionality. The inmanta container allows you to change the user and group id of the inmanta user inside the container when starting the container to match the user on the host, getting rid that way of any conflict in the files ownership.

This can be done easily by simply setting those environment variables:

- INMANTA_UID: Will change, when starting the container, the id of the inmanta user.

- INMANTA_GID: Will change, when starting the container, the id of the inmanta group.

If you use docker-compose, you can simply update this section of the example above:

```
inmanta-server:
    container_name: inmanta_orchestrator
    ...
    environment:
        INMANTA_UID: 1000
        INMANTA_GID: 1000
```

2.2.8 Log rotation

By default, the container won't do any log rotation, to let you the choice of dealing with the logs according to your own preferences. We recommend that you do so by mounting a folder inside of the container at the following path: /var/log. This path contains all the logs of inmanta (unless you specified a different path in the config of the server) and the logs of the SSH server.

2.3 Install Inmanta with Podman and Systemd

This page explains how to setup an Inmanta orchestration server using Podman and Systemd. This guide assumes you already have Podman installed on your machine and that you are running a Linux distribution using Systemd.

Note: The instructions below will show you how to install the orchestrator, and make the orchestrator run as a non-root user on the host. To achieve this you can either follow the rootless instructions (User setup), running them as a simple user without elevated privileged, or as root (Root setup). If you follow the latter, make sure to create a system user that we will use to run the orchestrator process. We will assume in the next steps that such system user is named inmanta and its HOME folder is /var/lib/inmanta.

2.3.1 Podman configuration

Follow the Podman documentation to make sure that:

1. The user that will run the orchestrator (your unprivileged user, or the inmanta system user) has a range of subuids and subgids available to use. You can check it is the case running those commands:

   #### User setup

   ```
   $ podman unshare cat /proc/self/uid_map
            0       1000          1
            1     524288      65536
   $ podman unshare cat /proc/self/gid_map
            0       1000          1
            1     524288      65536
   ```

Root setup

```
# sudo -i -u inmanta -- podman unshare cat /proc/self/uid_map
        0        976          1
        1    1000000      65536
# sudo -i -u inmanta -- podman unshare cat /proc/self/gid_map
        0        975          1
        1    1000000      65536
```

If it is not the case, you can set these up following the podman documentation referred above.

2. The user that will run the orchestrator has the `runRoot` folder configured as follow.

User setup

```
$ podman info | grep runRoot
  runRoot: /run/user/1000/containers
```

The value `1000` should match the id of your user.

```
$ id -u
1000
```

Root setup

```
# sudo -i -u inmanta -- podman info | grep runRoot
  runRoot: /run/inmanta/containers
```

We overwrite the default value that podman will set for this system user for two reasons:

1. The default values it picks depends on the way you used `podman` for the first time with this user.
2. The default values it picks will contain the id of the `inmanta` user in its path, which we don't want to make any assumption about in the next steps.

You can change this value by updating the file at `/var/lib/inmanta/.config/containers/storage.conf`, making sure this entry is in the configuration:

```
[storage]
runroot = "/run/inmanta/containers"
```

Then create the folder and reset podman.

```
# mkdir -p /run/inmanta
# chown -R inmanta:inmanta /run/inmanta
# sudo -i -u inmanta -- podman system reset -f
A "/var/lib/inmanta/.config/containers/storage.conf" config file exists.
Remove this file if you did not modify the configuration.
```

2.3.2 Pull the image

Step 1: Log in to container registry

Connect to the container registry using your entitlement token.

User setup

```
$ podman login containers.inmanta.com
Username: containers
Password: <your-entitlement-token>

Login Succeeded
```

Root setup

```
# sudo -i -u inmanta -- podman login containers.inmanta.com
Username: containers
Password: <your-entitlement-token>

Login Succeeded
```

Replace `<your-entitlement-token>` with the entitlement token provided with your license.

Step 2: Pull the image

Use `podman pull` to get the desired image:

User setup

```
$ podman pull containers.inmanta.com/containers/service-orchestrator:8
```

Root setup

```
# sudo -i -u inmanta -- podman pull containers.inmanta.com/containers/service-
↪orchestrator:8
```

This command will pull the latest release of the Inmanta Service Orchestrator image within this major version.

2.3.3 Prepare the orchestrator configuration

1. Get the default configuration file:

   As of now, the container cannot be configured with environment variables, we should use a configuration file, mounted inside the container. To do this, you can get the current configuration file from the container, edit it, and mount it where it should be in the container.

User setup

Let's create a file on the host at `~/.config/inmanta/inmanta.cfg`. We can take as template the default file already packaged in our container image.

```
$ mkdir -p ~/.config/inmanta
$ podman run --rm containers.inmanta.com/containers/service-orchestrator:8 cat /
↪etc/inmanta/inmanta.cfg > ~/.config/inmanta/inmanta.cfg
```

Root setup

Let's create a file on the host at `/etc/inmanta/inmanta.cfg`. We can take as template the default file already packaged in our container image.

```
# mkdir -p /etc/inmanta
# chown -R inmanta:inmanta /etc/inmanta
# sudo -i -u inmanta -- podman run --rm containers.inmanta.com/containers/
↪service-orchestrator:8 cat /etc/inmanta/inmanta.cfg | sudo -i -u inmanta --␣
↪tee /etc/inmanta/inmanta.cfg
```

2. Update database settings:

   It is very unlikely that your database setup will match the one described in the default config we just got. Update the configuration in the `[database]` section to reflect the setup you have.

   Note: The setup described here assumes you already have a PostgreSQL instance available that the orchestrator can use for its persistent storage. If it is not the case, please *jump to the end of this document*, where we explain to you how to easily deploy a database using Postman and Systemd.

3. Make sure that there is a folder on your host that can persist all the logs of the server and that it is owned by the user running the orchestrator service.

User setup

In this setup, the log folder on the host will be `~/.local/share/inmanta-orchestrator-server/logs`.

```
$ mkdir -p ~/.local/share/inmanta-orchestrator-server/logs
```

Root setup

In this setup, the log folder on the host will be `/var/log/inmanta`.

```
# mkdir -p /var/log/inmanta
# chown -R inmanta:inmanta /var/log/inmanta
```

Warning: Inside of the container, this folder will be mounted at `/var/log/inmanta` as it is the default location where the orchestrator saves its logs. This location is configurable in the orchestrator configuration file. If you for any reason would change this location in the configuration, make sure to update any usage of the `/var/log/inmanta` folder in the next installation steps.

4. Get the license files:

   Together with the access to the inmanta container repo, you should also have received a license and an entitlement file. The orchestrator will need them in order to run properly. You can also place them in a config directory on your host.

User setup

After this step, we assume that this folder is `~/.config/inmanta/license/` and that both files are named `com.inmanta.license` and `com.inmanta.jwe` respectively.

```
$ tree .config/inmanta
.config/inmanta
├── inmanta.cfg
└── license
    ├── com.inmanta.jwe
    └── com.inmanta.license

2 directories, 3 files
```

Root setup

After this step, we assume that this folder is `/etc/inmanta/license/` and that both files are named `com.inmanta.license` and `com.inmanta.jwe` respectively.

```
# tree /etc/inmanta
/etc/inmanta
├── inmanta.cfg
└── license
    ├── com.inmanta.jwe
    └── com.inmanta.license

2 directories, 3 files
```

2.3.4 Start the server with systemd

Here is a systemd unit file that can be used to deploy the server on your machine.

User setup

```
[Unit]
Description=Podman
Documentation=https://docs.inmanta.com
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman run \
        --cidfile=%t/%n.ctr-id \
        --cgroups=no-conmon \
        --sdnotify=conmon \
        -d \
        --replace \
        --publish=127.0.0.1:8888:8888 \
        --uidmap=993:0:1 \
        --uidmap=0:1:993 \
```

(continues on next page)

(continued from previous page)

```
        --uidmap=994:994:64543 \
        --gidmap=993:0:1 \
        --gidmap=0:1:993 \
        --gidmap=994:994:64543 \
        --name=inmanta-orchestrator-server \
        --volume=%E/inmanta/inmanta.cfg:/etc/inmanta/inmanta.cfg:z \
        --volume=%E/inmanta/license/com.inmanta.license:/etc/inmanta/license/com.
↪inmanta.license:z \
        --volume=%E/inmanta/license/com.inmanta.jwe:/etc/inmanta/license/com.inmanta.
↪jwe:z \
        --volume=%h/.local/share/inmanta-orchestrator-server/logs:/var/log/inmanta:z \
        --entrypoint=/usr/bin/inmanta \
        --user=993:993 \
        containers.inmanta.com/containers/service-orchestrator:8 \
        --log-file /var/log/inmanta/server.log --log-file-level 2 --timed-logs server
ExecStop=/usr/bin/podman stop \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm \
        -f \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

You can paste this configuration in a file named `inmanta-orchestrator-server.service` in the systemd folder for your user. This folder is typically `~/.config/systemd/user/`.

Root setup

```
[Unit]
Description=Podman
Documentation=https://docs.inmanta.com
Wants=network-online.target
After=network-online.target
RequiresMountsFor=/run/inmanta/containers

[Service]
User=inmanta
Group=inmanta
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman run \
        --cidfile=/run/inmanta/%n.ctr-id \
        --cgroups=no-conmon \
        --sdnotify=conmon \
        -d \
        --replace \
        --publish=127.0.0.1:8888:8888 \
        --uidmap=993:0:1 \
        --uidmap=0:1:993 \
```

(continues on next page)

(continued from previous page)

```
        --uidmap=994:994:64543 \
        --gidmap=993:0:1 \
        --gidmap=0:1:993 \
        --gidmap=994:994:64543 \
        --name=inmanta-orchestrator-server \
        --volume=/etc/inmanta/inmanta.cfg:/etc/inmanta/inmanta.cfg:z \
        --volume=/etc/inmanta/license/com.inmanta.license:/etc/inmanta/license/com.
↪inmanta.license:z \
        --volume=/etc/inmanta/license/com.inmanta.jwe:/etc/inmanta/license/com.
↪inmanta.jwe:z \
        --volume=/var/log/inmanta:/var/log/inmanta:z \
        --entrypoint=/usr/bin/inmanta \
        --user=993:993 \
        containers.inmanta.com/containers/service-orchestrator:8 \
        --log-file /var/log/inmanta/server.log --log-file-level 2 --timed-logs server
ExecStop=/usr/bin/podman stop \
        --ignore -t 10 \
        --cidfile=/run/inmanta/%n.ctr-id
ExecStopPost=/usr/bin/podman rm \
        -f \
        --ignore -t 10 \
        --cidfile=/run/inmanta/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

You can paste this configuration in a file named `inmanta-orchestrator-server.service` in the systemd folder `/etc/systemd/system`.

Note:

In the configuration above, you can observe that the usage of the `--uidmap` and `--gidmap` options. We use them three times to do the following:

1. Map the user `993` inside of the container (the container's `inmanta` user) to the user `0` in the podman user namespace. This user `0` in the user namespace is actually itself mapped to the user running the `podman run` command on the host.
2. Map all users from `0` to `65536` (except for `993`) inside of the container to subids of the host user running the container.

This allow us to easily share files between the host user and the `inmanta` user inside the container, avoiding any ownership conflict as they are then the same user (just seen from a different user namespace). Strictly speaking, if the image is already pulled on the host, you might get away with mapping only the `inmanta` (`--uidmap=993:0:1 --gidmap=993:0:1`) and the `root` (`--uidmap=0:1:1 --gidmap=0:1:1`) user and group inside of the container. But you would face issue if the container image was deleted from your host and the `run` command in the unit file tried to automatically pull the image, as the container image does contain a lot more users and groups than `inmanta` and `root` in its filesystem.

Once the systemd unit files are in place, make sure to enable them and reload the systemctl daemon.

User setup

```
$ systemctl --user daemon-reload
$ systemctl --user enable inmanta-orchestrator-server.service
```

Root setup

```
# systemctl daemon-reload
# systemctl enable inmanta-orchestrator-server.service
```

Then start the container by running the following command:

User setup

```
$ systemctl --user start inmanta-orchestrator-server.service
```

Root setup

```
# systemctl start inmanta-orchestrator-server.service
```

You should be able to reach the orchestrator at this address: http://127.0.0.1:8888 on the host.

2.3.5 Setting environment variables

You might want your inmanta server to be able to use some environment variables. You can set the environment variables by updating your Systemd unit file, relying on the `--env`/`--env-file` options of the `podman run` command. Those variables will be accessible to the inmanta server, the compiler and any agent started by the server.

2.3.6 Log rotation

By default, the container won't do any log rotation, we let you the choice of dealing with the logs according to your own preferences. We recommend you to setup some log rotation, for example using a logrotate service running on your host.

2.3.7 Deploy postgresql with podman and systemd

User setup

1. Pull the postgresql image from dockerhub.

   ```
   $ podman pull docker.io/library/postgres:13
   ```

2. Create a podman network for your database and the orchestrator.

   ```
   $ podman network create --subnet 172.42.0.0/24 inmanta-orchestrator-net
   ```

3. Create a systemd unit file for your database, let's name it `~/.config/systemd/user/inmanta-orchestrator-db.service`.

```
[Unit]
Description=Podman
Documentation=https://docs.inmanta.com
Wants=network-online.target
After=network-online.target
RequiresMountsFor=%t/containers

[Service]
Environment=PODMAN_SYSTEMD_UNIT=%n
Restart=on-failure
TimeoutStopSec=70
ExecStart=/usr/bin/podman run \
        --cidfile=%t/%n.ctr-id \
        --cgroups=no-conmon \
        --sdnotify=conmon \
        -d \
        --replace \
        --network=inmanta-orchestrator-net:ip=172.42.0.2 \
        --uidmap=999:0:1 \
        --uidmap=0:1:999 \
        --uidmap=1000:1000:64537 \
        --gidmap=999:0:1 \
        --gidmap=0:1:999 \
        --gidmap=1000:1000:64537 \
        --name=inmanta-orchestrator-db \
        --volume=%h/.local/share/inmanta-orchestrator-db/data:/var/lib/
↪postgresql/data:z \
        --env=POSTGRES_USER=inmanta \
        --env=POSTGRES_PASSWORD=inmanta \
        docker.io/library/postgres:13
ExecStop=/usr/bin/podman stop \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
ExecStopPost=/usr/bin/podman rm \
        -f \
        --ignore -t 10 \
        --cidfile=%t/%n.ctr-id
Type=notify
NotifyAccess=all

[Install]
WantedBy=default.target
```

4. Create the folder that will contain the persistent storage for the database: `~/.local/shared/inmanta-orchestrator-db/data`.

```
$ mkdir -p ~/.local/share/inmanta-orchestrator-db/data
```

5. Reload the systemd daemon, enable the service, and start it.

```
$ systemctl --user daemon-reload
$ systemctl --user enable inmanta-orchestrator-db.service
$ systemctl --user start inmanta-orchestrator-db.service
```

6. In the unit file of the orchestrator (as described *here*), make sure to attach the orchestrator container to the network the database is a part of, using the `--network` option of the `podman run` command.

7. Don't forget to update the ip address of the database in the inmanta server configuration file (`~/.config/inmanta/inmanta.cfg`)!

Root setup

For a proper install of postgres on your host system as root, please refer to the postgres documentation regarding your operating system.

2.4 Configure agents

Inmanta agents can be started automatically (auto-started agents) or manually (manually-started agents). This section describes how both types of agents can be set up and configured. Inmanta agents only run on Linux.

2.4.1 Auto-started agents

Auto-started agents always run on the Inmanta server. The Inmanta server manages the full lifecycle of these agents.

Configuring auto-started agents via environment settings

Auto-started agents can be configured via the settings of the environment where the auto-started agent belongs to. The following options are configurable:

- *autostart_agent_map*
- *autostart_agent_deploy_interval*
- *autostart_agent_deploy_splay_time*
- *autostart_agent_repair_interval*
- *autostart_agent_repair_splay_time*
- *autostart_on_start*

The *autostart_agent_map* requires an entry for each agent that should be autostarted. The key is the name of the agent and the value is either `local:` for agents that map to the Inmanta server or an SSH connection string when the agent maps to a remote machine. The SSH connection string requires the following format: `ssh://<user>@<host>:<port>?<options>`. Options is a ampersand-separated list of `key=value` pairs. The following options can be provided:

Option name	Default value	Description
retries	10	The amount of times the orchestrator will try to establish the SSH connection when the initial attempt failed.
retry_wait	30	The amount of second between two attempts to establish the SSH connection.
python	python	The Python3 interpreter available on the remote side. This executable has to be discoverable through the system PATH.

Auto-started agents start when they are required by a specific deployment or when the Inmanta server starts if the *autostart_on_start* setting is set to true. When the agent doesn't come up when required, consult the *troubleshooting documentation* to investigate the root cause of the issue.

Configuring the autostart_agent_map via the std::AgentConfig entity

The *std::AgentConfig* entity provides functionality to add an entry to the *autostart_agent_map* of a specific environment. As such, the auto-started agents can be managed in the configuration model.

Special Requirements for remote std::File, std::Package, std::Service and exec::Run

When using the agents built-in ssh capability, to perform actions over ssh on remote hosts, the following requirements must be met:

- The Inmanta server should have passphraseless SSH access on the machine it maps to. More information on how to set up SSH connectivity can be found at *Step 7: Configure ssh of the inmanta user*
- The remote machine should have a Python 2 or 3 interpreter installed. The binary executed by default is `python`.
- The user to log into the remote machine should either be `root` or have the ability to do a passwordless sudo. To enable passwordless sudo for the user `username`, add a file to `/etc/sudoers.d/` containing `username ALL=(ALL) NOPASSWD: ALL`. It is advisable to use a safe editor such as `visudo` or `sudoedit` for this. For more details, go here.

2.4.2 Manually-started agents

Manually started agents can be run on any Linux device, but they should be started and configured manually as the name suggests.

Requirements

The following requirements should be met for agents that don't map to the host running the agent process (i.e. The managed device is remote with respect to the Inmanta agent and the agent has to execute I/O operations on the remote machine using `self._io`):

- The Inmanta agent should have passphraseless SSH access on the machine it maps to. More information on how to set up SSH connectivity can be found at *Step 7: Configure ssh of the inmanta user*
- The remote machine should have a Python 2 or 3 interpreter installed. The binary executed by default is `python`.

Step 1: Installing the required Inmanta packages

In order to run a manually started agent, the `inmanta-service-orchestrator-agent` package is required on the machine that will run the agent.

```
sudo tee /etc/yum.repos.d/inmanta.repo <<EOF
[inmanta-service-orchestrator-8-stable]
name=inmanta-service-orchestrator-8-stable
baseurl=https://packages.inmanta.com/<token>/inmanta-service-orchestrator-8-stable/
↪rpm/el/8/$basearch
gpgcheck=1
gpgkey=https://packages.inmanta.com/<token>/inmanta-service-orchestrator-8-stable/
↪cfg/gpg/gpg.F4B97D6483D7D2BE.key
repo_gpgcheck=1
enabled=1
enabled_metadata=1
EOF

sudo dnf install -y inmanta-service-orchestrator-agent
```

Replace <token> with the token provided with your license.

Step 2: Configuring the manually-started agent

The manually-started agent can be configured via a `/etc/inmanta/inmanta.d/*.cfg` config file. The following options configure the behavior of the manually started agent:

- *config.state-dir*
- *config.agent-names*
- *config.environment*
- *config.agent-map*
- *config.agent-deploy-splay-time*
- *config.agent-deploy-interval*
- *config.agent-repair-splay-time*
- *config.agent-repair-interval*
- *config.agent-reconnect-delay*
- *config.server-timeout*
- *agent_rest_transport.port*
- *agent_rest_transport.host*
- *agent_rest_transport.token*
- *agent_rest_transport.ssl*
- *agent_rest_transport.ssl-ca-cert-file*

The agent will follow the pip configuration defined in the *project.yml*. Make sure it can access the pip index configured by the project (See the pip documentation for netrc for more information on how to setup authentication).

The *config.agent-map* option can be configured in the same way as the `autostart_agent_map` for auto-started agents.

Step 3: Starting the manually-started agent

Finally, enable and start the `inmanta-agent` service:

```
sudo systemctl enable inmanta-agent
sudo systemctl start inmanta-agent
```

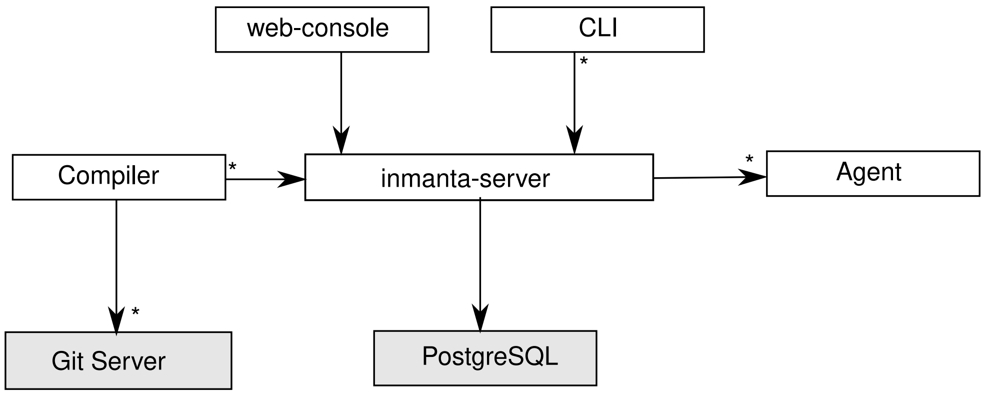
The logs of the agent are written to `/var/log/inmanta/agent.log`. When the agent doesn't come up after starting the `inmanta-agent` service, consult the *troubleshooting documentation* to investigate the root cause of the issue.

CHAPTER

THREE

ARCHITECTURE

The Inmanta orchestrator consists of several components:



- The Inmanta **server**: This server manages the deployment process, it keeps track of all agents and the current state of all projects. The server stores it state in PostgreSQL. All other state can be recovered after a server restart or failover.
- A PostgreSQL database: The Inmanta server stores its state in a PostgreSQL database.
- The git server: The source code of the configuration models is stored in (one or more) git repositories.
- The **compiler**: The compiler converts the source code into deployable resources and exports it to the server.
- CLI and web-console: To control the server, you can use either the web-console or the command line tools. Both communicate through the server rest API.
- The Inmanta **agents**: Agents execute configuration changes on targets. A target can be a server, a network switch or an API or cloud service. An agent can manage local and remote resources. This provides the flexibility to work in an agent based or agent-less architecture, depending on the requirements.

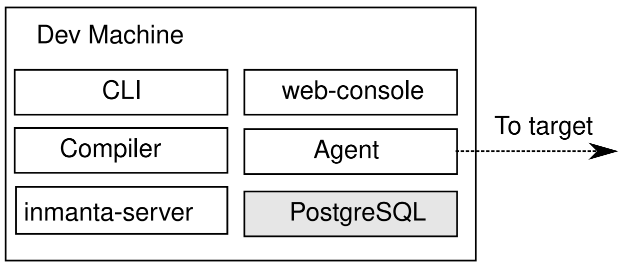
3.1 Usage modes

Inmanta can be used in three modes:

- **embedded**: all components are started with the *deploy* command, the server is terminated after the deploy is finished. Suitable only for development.
- **push to server**: the server runs on a external machine. Models are compiled on the developer machine and pushed to the server directly. Suitable only for small setups or for developement/debug purposes.
- **autonomous server**: the server runs on a external machine. Models are stored in git repos and compiled by the server.

The last two modes support agents on same machine as the server and automatically started, or deployed as an external process.

3.1.1 All in one

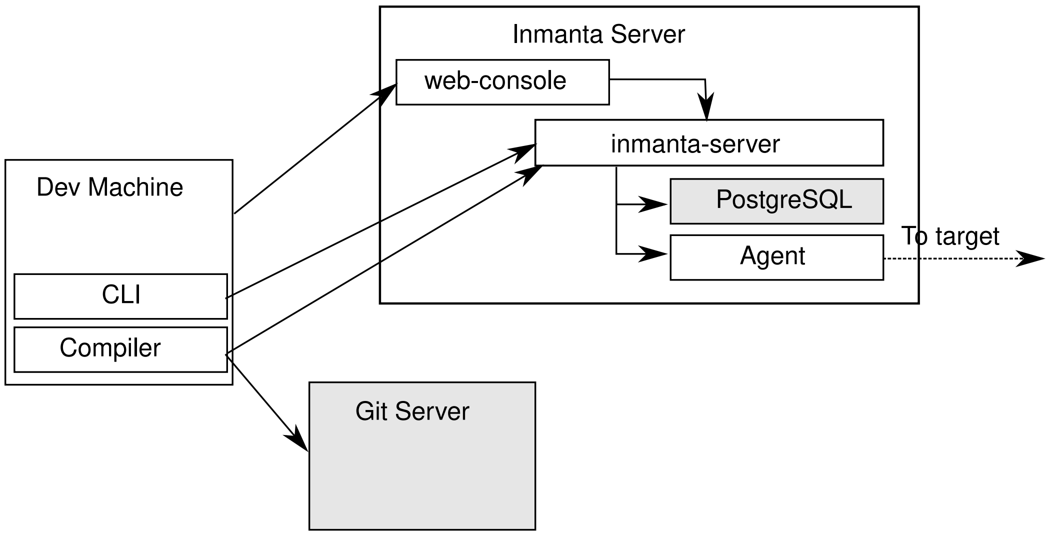


In a all-in-one deployment, all components (server, agent and postgres) are started embedded in the compiler and terminated after the deploy is complete. No specific setup is required. To deploy the current model, use:

```
inmanta deploy
```

The all-in-one deployment is ideal for testing, development and one-off deployments. State related to orchestration is stored locally in data/deploy.

3.1.2 Push to server



In a push to server model, the server is deployed on an external machine, but models are still compiled on the developer machine. This gives faster feedback to developers, but makes the compilation less reproducible. It also complicates collaboration.

Both the developer machine and the server need to have Inmanta installed. To compile and export models to the server from the developer machine a `.inmanta` file is required in the project directory (where you find the main.cf and the project.yaml file) to connect the compiler with the server.

Create a `.inmanta` file in the project directory and include the following configuration:

```
[config]
environment=$ENV_ID

[compiler_rest_transport]
```

(continues on next page)

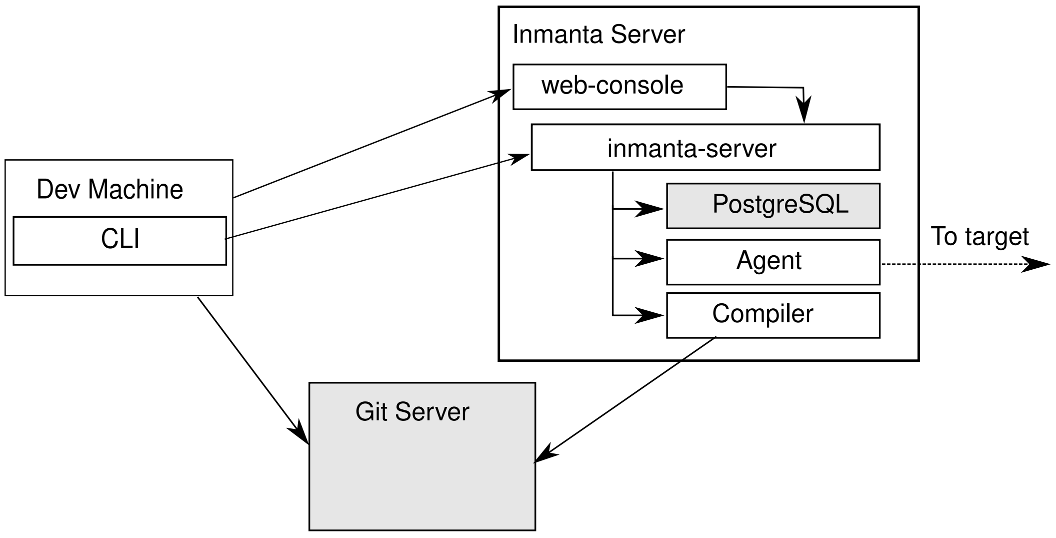
(continued from previous page)

```
host=$SERVER_ADDRESS
port=$SERVER_PORT
```

Replace `$ENV_ID`, `$SERVER_ADDRESS` and `$SERVER_PORT` with the correct values (See `compiler_rest_transport` for more details when using ssl and or auth, `config.environment` explains the environment setting). A best practice is to not add the .inmanta to the git repository. Because different developer may use different orchestration servers.

- `inmanta compile` compiles the current project but does not upload the result to the orchestration server.
- `inmanta export` compiles and uploads the current project to the orchestration server. Depending on the environment settings the server will release and deploy the model or it becomes available in the *new* state.
- `inmanta export -d` compiles, uploads and releases the current project. The result will start deploying immediately.

3.1.3 Autonomous server



With an autonomous server, developers can no longer push models into production directly. Only the server itself compiles the models. This ensures that every compile is repeatable and allows collaboration because all changes *have* to be committed.

3.2 Agent modes

The Inmanta agent performs all changes in the infrastructure. Either the orchestration server starts an agents or an agent is deployed as a separate process.

- **agentless**: Autostarted agents allow for an agentless mode: no explicit agents need to be started. When the agent needs to make changes on machine/vm it can make the changes over remote over ssh. Autostarted agents are controlled by using `std::AgentConfig`. `ip::Host` and subclasses can automatically configure an agent with the *remote_agent* attribute.
- **external agent**: External agent processes need explicit configuration to connect to the orchestration server. The aws and openstack modules use the platform module to generate a user_data bootscript for virtual machines to install an agent and connect to the orchestration server. The *install_agent* boolean controls this option.

3.3 Resource deployment

The agent is responsible for:

- repair the infrastructure at regular intervals
- change the infrastructure at regular intervals
- enforce desired state when the server requests it

3.3.1 Repair

At regular intervals the agent verifies that the current state of all resources it manages matches the desired state provided by the orchestration server. For a repair the agent verifies all resources, even if the last known current state already matches the desired state. In the current release all deploys are done through a repair and run by default every 600 seconds. This is controlled with *`config.agent-repair-interval`*, when this option is set to 0 no repairs are performed.

3.3.2 Deploy changes

For very large infrastructures or infrastructure that is too slow (for example network devices with underpowered control planes or thousands of managed resources) a repair cannot run often. For example, only once a week. When this is the case, the agent can deploy only known changes (based on the previous deployed state cached by the orchestration server). This interval is controlled by *`config.agent-deploy-interval`*. This interval should be a lot shorter than *`config.agent-repair-interval`*

When a repair is running and a deploy run is started, the repair is cancelled, the deploy is performed and then the repair is restarted. This repair starts again from scratch. So when repairs take a very long time, they might never finish completely when there is a high rate of change.

3.3.3 Push changes

For very interactive changes the server pushes changes to the agent. The server can push full and incremental desired state to the agent.

- **incremental** only deploys resource for which the orchestrator knows there are changes, based on the last known deploy status of the resource.
- **full** always deploys all resources even if the last know status of the resource already matches desired state.

CHAPTER

FOUR

LANGUAGE REFERENCE

The Inmanta language is a declarative language to model the configuration of an infrastructure.

The evaluation order of statements is determined by their dependencies on other statements and not based on the lexical order. i.e. The code is not necessarily executed top to bottom.

4.1 Modules

The source is organized in modules. Each module is a git repository with the following structure:

```
module/
+-- files/
+-- model/
|  +-- _init.cf
+-- plugins/
+-- templates/
+-- module.yml
```

Note: The module format described here is the v1 module format. For more details see *Understanding Modules*.

The `module.yml` file, the `model` directory and the `model/_init.cf` are required.

For example:

```
test/
+-- files/
+-- model/
|  +-- _init.cf
|  +-- services.cf
|  +-- policy
|  |  +-- _init.cf
|  |  +-- other.cf
+-- plugins/
+-- templates/
+-- module.yml
```

The model code is in the `.cf` files. Each file forms a namespace. The namespaces for the files are the following.

File	Namespace
test/model/_init.cf	test
test/model/services.cf	test::services
test/model/policy/_init.cf	test::policy
test/model/policy/other.cf	test::policy::other

Modules are only loaded when they are imported by a loaded module or the `main.cf` file of the project.

To access members from another namespace, it must be imported into the current namespace.:

```
import test::services
```

Imports can also define an alias, to shorten long names:

```
import test::services as services
```

4.2 Variables

Variables can be defined in any lexical scope. They are visible in their defining scope and its children. A lexical scope is either a namespaces or a code block (area between `:` and `end`).

Variable names must start with a lower case character and can consist of the characters: `a-zA-Z_0-9-`

A value can be assigned to a variable exactly once. The type of the variable is the type of the value. Assigning a value to the same variable twice will produce a compiler error, unless the values are identical.

Variables from other modules can be referenced by prefixing them with the module name (or alias)

```
import redhat
os = redhat::fedora23
import ubuntu as ubnt
os2 = ubnt::ubuntu1204
```

4.3 Literals values

Literal values can be assigned to variables

```
var1 = 1 # assign an integer, var1 contains now a number
var2 = 3.14 # assign a float, var2 also contains a number
var3 = "This is a string" # var3 contains a string
var4 = r"This is a raw string" # var4 contains a raw string

# var 5 and 6 are both booleans
var5 = true
var6 = false

# var7 is a list of values
var7 = ["fedora", "ubuntu", "rhel"]

# a dictionary with string keys and any type of values is also a primitive
var8 = { "foo":"bar", "baz": 1}

# var9 contains the same value as var2
var9 = var2

# next assignment will not return an error because var1 already contains this value
var1 = 1

# next assignment would return an error because var1 already has a different value
#var1 = "test"

#ref to a variable from another namespace
```

(continues on next page)

(continued from previous page)

```
import ip::services
sshservice = ip::services::ssh
```

4.4 Arithmetic operations

The following arithmetic operations are supported:

- Addition (+)
- Substraction (-)
- Multiplication (*)
- Division (/)
- Exponentiation (**)
- Modulo (%)

Example:

```
var = 3 + 5
var = 10 - 2
var = 4 * 2
var = int(16 / 2)
var = 2 ** 3
var = 18 % 10
```

Note that the result of the division operation is cast to the type `int`. This is done because a division always results in a value of type `float`.

4.5 Primitive types

The basic primitive types are `string`, `float`, `int` or `bool`. These basic types also support type casts:

Note: To initialize or assign a float, the value should either include a decimal point or be explicitly converted to a float type.

```
assert = true
assert = int("1") == 1
assert = float("1.2") == 1.2
assert = int(true) == 1
assert = bool(1.2) == true
assert = bool(0) == false
assert = bool(null) == false
assert = bool("x") == true
# like in Python, only empty strings are considered false
assert = bool("false") == true
assert = bool("") == false
assert = string(true) == "true"
```

Constrained primitive types can be derived from the basic primitive type with a typedef statement. Constrained primitive types add additional constraints to the basic primitive type with either a Python regex or a logical *condition*. The name of the constrained primitive type must not collide with the name of a variable or type in the same lexical scope.

A regex matches a given string when zero or more characters at the beginning of that string match the regular expression. A dollar sign should be used at the end of the regex if a full string match is required.

```
typedef : 'typedef' ID 'as' PRIMITIVE 'matching' condition|regex;
```

For example

```
typedef tcp_port as int matching self > 0 and self < 65535
typedef mac_addr as string matching /([0-9a-fA-F]{2})(:[0-9a-fA-F]{2}){5}$/
```

Lists of primitive types are also primitive types: `string[]`, `float[]`, `bool[]` or `mac_addr[]`

`dict` is the primitive type that represents a dictionary, with string keys. Dict values can be accessed using the `[]` operator. All members of a dict have to be set when the dict is constructed. e.g.

```
#correct
a = {"key":"value", "number":7}
value = a["key"]
# value = "value"
# incorrect, can't assign to dict after construction
# a["otherkey"] = "othervalue"
```

4.5.1 Strings

There are four kinds of strings in the Inmanta language:

- regular strings

```
regular_string_1 = "This is...\n...a basic string."

# Output when displayed:
# This is...
# ...a basic string.


regular_string_2 = 'This one too.'

# Output when displayed:
# This one too.
```

- multi-line strings

It is possible to make a string span multiple lines by triple quoting it e.g.:

```
multi_line_string = """This
string
spans
multiple
lines"""

# Output when displayed:
# This
# string
# spans
# multiple
# lines
```

Note: Unlike python's multi-line strings, only double quotes are supported to define a multi-line string i.e. `"""` is valid, but `'''` is not.

- raw strings

Raw strings are similar to python's raw strings in that they treat backslashes as regular characters. On the other hand, in regular and multi-line strings, escape characters (e.g. `\n`, `\t...`) are interpreted and therefore backslashes need to be escaped in order to be displayed. In addition, no variable expansion is performed in raw strings.

```
raw_string = r"This is...\n...a raw string."

# Output when displayed:
# This is...\n...a raw string.

hostname = "serv1.example.org"
raw_motd = r"Welcome to {hostname}"

# Output when displayed:
# Welcome to {hostname}
```

- f-strings

An alternative syntax similar to python's f-strings can be used for string formatting.

```
hostname = "serv1.example.org"
motd = f"Welcome to {hostname}"

# Output when displayed:
# Welcome to serv1.example.org
```

Python's format specification mini-language can be used for fine-grained formatting:

```
width = 10
precision = 2
arg = 12.34567

std::print(f"result: {arg:{width}.{precision}f}")

# Output:
# result:      12.35
```

Note: The '=' character specifier added in python 3.8 is not supported yet in the Inmanta language.

Note: Unlike in python, raw and format string cannot be used together in the same string e.g. `raw_and_format = rf"Both specifiers"` is not allowed.

String interpolation

An alternative syntax to f-strings is string interpolation. It allows variables to be included as parameters inside a regular or multi-line string. The included variables are resolved in the lexical scope of the string they are included in:

```
hostname = "serv1.example.org"
motd = "Welcome to {{hostname}}"

# Output when displayed:
# Welcome to serv1.example.org
```

String concatenation

Strings can be concatenated with the + operator.

```
hello_world = "hello " + "world"
```

4.6 Conditions

Conditions can be used in typedef, implements and if statements. A condition is an expression that evaluates to a boolean value. It can have the following forms

```
condition : '(' condition ')'
    | condition 'or' condition
    | condition 'and' condition
    | 'not' condition
    | value
    | value ('>' | '>=' | '<' | '<=' | '==' | '!=') value
    | value 'in' value
    | value 'not in' value
    | functioncall
    | value 'is' 'defined'
    ;
```

The `in` and `not in` operators can be used to check if a value is present in a list:

```
myfiles = ["/a/b/c", "/c/d/e", "x/y/z/u/v/w"]

condition1 = "/a/b/c" in myfiles # evaluates to True
condition2 = "/f/g/h" in myfiles # evaluates to False

condition3 = "/a/b/c" not in myfiles # evaluates to False
condition4 = "/f/g/h" not in myfiles # evaluates to True

condition5 = not "/a/b/c" in myfiles # evaluates to False
condition6 = not "/f/g/h" in myfiles # evaluates to True
```

The `is defined` keyword checks if a value was assigned to an attribute or a relation of a certain entity. The following example sets the monitoring configuration on a certain host when it has a monitoring server associated:

```
entity Host:

end

```

(continues on next page)

(continued from previous page)

```
entity MonitoringServer:

end

Host.monitoring_server [0:1] -- MonitoringServer

implement Host using monitoringConfig when monitoring_server is defined

implementation monitoringConfig for Host:
    # Set monitoring config
end
```

Empty lists are considered to be unset.

4.7 Function calls / Plugins

Each module can define plugins. Plugins can contribute functions to the module's namespace. The function call syntax is

```
functioncall : moduleref '.' ID '(' arglist? ')';
arglist : arg
        | arglist ',' arg
        ;
arg : value
    | key '=' value
    | '**' value
    ;
```

For example

```
std::familyof(host.os, "rhel")
a = param::one("region", "demo::forms::AWSForm")

hello_world = "Hello World!"
hi_world = std::replace(hello_world, new = "Hi", old = "Hello")
dct = {
    "new": "Hi",
    "old": "Hello",
}
hi_world = std::replace(hello_world, **dct)
```

4.8 Entities

Entities model configuration concepts. They are like classes in other object oriented languages: they can be instantiated and they define the structure of their instances.

Entity names must start with an upper case character and can consist of the characters: `a-zA-Z_0-9-`

Entities can have a number of attributes and relations to other entities. Entity attributes have primitive types, with an optional default value. An attribute has to have a value unless the nulable variant of the primitive type is used. An attribute that can be null uses a primitive type with a `?` such as `string?`. A value can also be assigned only once to an attribute that can be null. To indicate that no value will be assigned, the literal `null` is available. `null` can also be the default value of an attribute.

Entities can inherit from multiple other entities. Entities inherits attributes and relations from parent entities. All entities inherit from `std::Entity`.

It is not possible to override or rename attributes or relations. However, it is possible to override defaults. Default values for attributes defined in the class take precedence over those in the parent classes. When a class has multiple parents, the left parent takes precedence over the others. A default value can be removed by setting its value to `undef`.

The syntax for defining entities is:

```
entity: 'entity' ID ('extends' classlist)? ':' attribute* 'end';

classlist: class
         | class ',' classlist;

attribute: primitve_type ID ('=' literal)?;
```

Defining entities in a configuration model

```
entity File:
   string path
   string content
   int mode = 640
   string[] list = []
   dict things = {}
end
```

4.9 Relations

A Relation is a unidirectional or bidirectional relation between two entities. The consistency of a bidirectional double binding is maintained by the compiler: assignment to one side of the relation is an implicit assignment of the reverse relation.

Relations are defined by specifying each end of the relation together with the multiplicity of each relation end. Each end of the relation is named and is maintained as a double binding by the compiler.

Defining relations between entities in the domain model

```
relation: class '.' ID multi '--' class '.' ID multi
        | class '.' ID multi annotation_list class '.' ID multi ;
annotation_list: value
        | annotation_list ',' value
```

For example a bidirectional relation:

```
File.service [1] -- Service.file [1:]
```

Or a unidirectional relation

```
uni_relation : class '.' ID multi '--' class
       | class '.' ID multi annotation_list class;
```

For example

```
Service.file [1:] -- File
```

Relation multiplicities are enforced by the compiler. If they are violated a compilation error is issued.

4.10 Instantiation

Instances of an entity are created with a constructor statement

```
File(path="/etc/motd")
```

A constructor can assign values to any of the properties (attributes or relations) of the entity. It can also leave the properties unassigned. For attributes with default values, the constructor is the only place where the defaults can be overridden.

Values can be assigned to the remaining properties as if they are variables. To relations with a higher arity, multiple values can be assigned. Additionally, *null* can be assigned to relations with a lower arity of 0 to indicate explicitly that the model will not assign any values to the relation attribute.

```
Host.files [0:] -- File.host [1]

h1 = Host("test")
f1 = File(host=h1, path="/opt/1")
f2 = File(host=h1, path="/opt/2")
f3 = File(host=h1, path="/opt/3")

# h1.files equals [f1, f2, f3]

FileSet.files [0:] -- File.set [1]

s1 = FileSet()
s1.files = [f1,f2]
s1.files = f3

# s1.files equals [f1, f2, f3]

s1.files = f3
# adding a value twice does not affect the relation,
# s1.files still equals [f1, f2, f3]
```

In addition, attributes can be assigned in a constructor using keyword arguments by using `**dct` where `dct` is a dictionary that contains attribute names as keys and the desired values as values. For example:

```
Host.files [0:] -- File.host [1]
h1 = Host("test")

file1_config = {"path": "/opt/1"}
f1 = File(host=h1, **file1_config)
```

It is also possible to add elements to a relation with the += operator:

```
Host.files [0:] -- File.host [1]

h1 = Host("test")
h1.files += f1
h1.files += f2
h1.files += f3

# h1.files equals [f1, f2, f3]
```

Note: This syntax is only defined for relations. The += operator can not be used on variables, which are immutable.

4.10.1 Referring to instances

When referring to entities in the same module, a parent model or std, short names can be used

Following code blocks are equivalent and both valid

```
std::Host("test")
```

```
Host("test")
```

When constructing entities from other modules, the fully qualified name must be used

```
import srlinux
import srlinux::interface

interface = srlinux::Interface(
     subinterface = srlinux::interface::Subinterface(
     )
 )
```

When nesting constructors, short names can be used for the nested constructors, because their types can be inferred

```
import srlinux
import srlinux::interface

interface = srlinux::Interface( # This type is qualified
     subinterface = Subinterface( # This type is inferred
     )
 )
```

However, when relying on type inference:

1. avoid creating sibling types with the same name, but different fully qualified name, as they may become indistinguishable, breaking the inference on existing models.
   1. if multiple types exist with the same name, and one is in scope, that one is selected (i.e. it is defined in this module, a parent module or `std`)
   2. if multiple types exist that are all out of scope, inference fails
2. make sure the type you want to infer is imported somewhere in the model. Otherwise the compiler will not find it.

4.11 Refinements

Entities define what should be deployed. Entities can either be deployed directly (such as files and packages) or they can be refined. Refinement expands an abstract entity into one or more more concrete entities.

For example, `apache::Server` is refined as follows

```
implementation apacheServerDEB for Server:
    pkg = std::Package(host=host, name="apache2-mpm-worker", state="installed")
    pkg2 = std::Package(host=host, name="apache2", state="installed")
    svc = std::Service(host=host, name="apache2", state="running", onboot=true,␣
↪reload=true, requires=[pkg, pkg2])
    svc.requires = self.requires

    # put an empty index.html in the default documentroot so health checks do not fail
    index_html = std::ConfigFile(host=host, path="/var/www/html/index.html", content="
```

(continues on next page)

(continued from previous page)

```
↪",
                                         requires=pkg)
    self.user = "www-data"
    self.group = "www-data"
end

implement Server using apacheServerDEB when std::familyof(host.os, "ubuntu")
```

For each entity one or more refinements can be defined with the `implementation` statement. Implementation are connected to entities using the `implement` statement.

When an instance of an entity is constructed, the runtime searches for refinements. One or more refinements are selected based on the associated *conditions*. When no implementation is found, an exception is raised. Entities for which no implementation is required are implemented using *std::none*.

In the implementation block, the entity instance itself can be accessed through the variable self.

`implement` statements are not inherited, unless a statement of the form `implement ServerX using parents` is used. When it is used, all implementations of the direct parents will be inherited, including the ones with a where clause.

The syntax for implements and implementation is:

```
implementation: 'implementation' ID 'for' class ':' statement* 'end';
implement: 'implement' class 'using' implement_list
         | 'implement' class 'using' implement_list_cond 'when' condition
         ;
implement_list: implement_list_cond
              | 'parents'
              | implement_list ',' implement_list
              ;
implement_list_cond: ID
                   | ID ',' implement_list_cond
                   ;
```

4.12 Indexes and queries

Index definitions make sure that an entity is unique. An index definition defines a list of properties that uniquely identify an instance of an entity. If a second instance is constructed with the same identifying properties, the first instance is returned instead.

All identifying properties must be set in the constructor.

Indices are inherited. i.e. all identifying properties of all parent types must be set in the constructor.

Defining an index

```
entity Host:
    string  name
end

index Host(name)
```

Explicit index lookup is performed with a query statement

```
testhost = Host[name="test"]
```

For indices on relations (instead of attributes) an alternative syntax can be used

```
entity File:
    string path
end

Host.files [0:] -- File.host [1]

index File(host, path)

a = File[host=vm1, path="/etc/passwd"]  # normal index lookup
b = vm1.files[path="/etc/passwd"]  # selector style index lookup
# a == b
```

Note: The use of `float` (or `number`) as part of index properties is generally discouraged. This is due to the reliance of index matching on precise equality, while floating-point numbers are represented with an inherent imprecision. If floating-point attributes are used in an index, it is crucial to handle arithmetic operations with caution to ensure the accuracy of the attribute values for index operations.

4.13 For loop

To iterate over the items of a list, a for loop can be used

```
for i in std::sequence(size, 1):
    app_vm = Host(name="app{{i}}")
end
```

The syntax is:

```
for: 'for' ID 'in' value ':' statement* 'end';
```

4.14 If statement

An if statement allows to branch on a condition.

```
if nodecount > 1:
    self.cluster_mode = "multi"
elif node == 1:
    self.cluster_mode = "single"
else:
    self.cluster_mode = "off"
end
```

The syntax is:

```
if : 'if' condition ':' statement* ('elif' condition ':' statement*)* ('else' ':'␣
↪statement*)? 'end';
```

The *Conditions* section describes allowed forms for the condition.

4.15 Conditional expressions

A conditional expression is an expression that evaluates to one of two subexpressions depending on its condition.

```
x = n > 0 ? n : 0
```

Which evaluates to n if n > 0 or to 0 otherwise.

The syntax is:

```
conditional_expression : condition '?' expression ':' expression;
```

The *Conditions* section describes allowed forms for the condition.

4.16 List comprehensions

A list comprehension constructs a list (either a primitive list or a relation) by mapping over another list, optionally filtering some values.

```
myfiles = ["/a/b/c", "/c/d/e", "x/y/z/u/v/w"]
# create File instance for each file in myfiles shorter than 10 characters
host.files = [File(path=path) for path in myfiles if std::length(path) < 10]
```

The syntax is the following.

```
list_comprehension : '[' expression ('for' ID 'in' expression)+ ('if' expression)* ']'
```

It shows that the list comprehension allows for multiple `for` expressions and multiple `if` guards. The top `for` is always executed first, as if it were the outer `for` in a conventional for loop. Here's an example:

```
all_short_files = [
    file
    for host in all_hosts
    for file in host.files  # we can refer to the upper loop variable `host`
    if host.name != "exclude_this_host"
    if std::length(file.path) < 10
]
```

While the inmanta language does not make any guarantees about statement execution order, it does provide some guarantees regarding data ordering for list comprehensions. In the context of relations even data order doesn't matter, but in the context of a literal list it might. In such a context the list comprehension promises to keep the order of the list in the `for` expression.

```
my_ordered_numbers = std::sequence(10)
my_ordered_pairs = ["{{i}}-{{i}}" for i in my_ordered_numbers]
# order is kept => ["0-0", "1-1", "2-2", ...]
```

4.17 Transformations

At the lowest level of abstraction the configuration of an infrastructure often consists of configuration files. To construct configuration files, templates and string interpolation can be used.

4.17.1 Templates

Inmanta integrates the Jinja2 template engine. A template is evaluated in the lexical scope where the `std::template` function is called. This function accepts as an argument the path of a template file. The first part of the path is the module that contains the template and the remainder of the path is the path within the template directory of the module.

The integrated Jinja2 engine supports to the entire Jinja feature set, except for subtemplates. During execution Jinja2 has access to all variables and plug-ins that are available in the scope where the template is evaluated. However, the `::` in paths needs to be replaced with a `.`. The result of the template is returned by the template function.

Using a template to transform variables to a configuration file

```
hostname = "wwwserv1.example.com"
admin = "joe@example.com"
motd_content = std::template("motd/message.tmpl")
```

The template used in the previous listing

```
Welcome to {{ hostname }}
This machine is maintainted by {{ admin }}
```

4.18 Plug-ins

For more complex operations, python plugins can be used. Plugins are exposed in the Inmanta language as function calls, such as the template function call. A template accepts parameters and returns a value that it computed out of the variables. Each module that is included can also provide plug-ins. These plug-ins are accessible within the namespace of the module. The *Developing Plugins* section of the module guide provides more details about how to write a plugin.

CHAPTER

FIVE

MODEL DEVELOPER DOCUMENTATION

5.1 Developer Getting Started Guide

This guide explains how to set up the recommended developer setup on a Linux machine. Other development setups are possible, but this one provides a good starting point.

- Install VS Code and Inmanta extension.
- Setting up Python virtual environments.
- Setting up a project.
- Set project sources
- Setting up a module
- Run tests
- Module developers guide
- Required environment variables

The examples below are using `pip` your system might require you to use `pip3`.

5.1.1 Install VS Code and Inmanta extension

The developer setup is based on VSCode with the Inmanta extension.

In order to install VS Code, you can refer to this page.

Inmanta's extension in VS Code marketplace can be found here.

Further information about Inmanta VS Code extension is available on this page.

5.1.2 Setting up Python virtual environments

For every project that you work on, we recommend using a new virtual environment. If you are unfamiliar with venv's, you can check out this page.

To create a virtual environment:

```
python3 -m venv ~/.virtualenvs/my_project
```

Then activate it by running:

```
source ~/.virtualenvs/my_project/bin/activate
```

Upgrading your `pip` will save you a lot of time and troubleshooting.

You can do so by running:

```
pip install --upgrade pip wheel
```

5.1.3 Setting up a project

At the time of this writing, linting and code navigation in IDEs work only if you have a project, so even if you only work on a single module, it is best to have a project.

There are two scenarios:

1. *Working on a New Project*.
2. *Working on an Existing Project*.

Working on a New Project

To create a new project you need to install some essential packages as follows:

```
pip install inmanta-core pytest-inmanta
```

Create a new project using the inmanta-project-template:

```
pip install cookiecutter

cookiecutter https://github.com/inmanta/inmanta-project-template.git
```

Navigate into the project and install the module dependencies using the inmanta CLI tool:

```
cd <project_name>

inmanta project install
```

V1 modules will be downloaded to the `downloadpath` configured in the `project.yml` file. V2 modules are installed in the active Python environment. For more details go *here*. Once you are done with creating a project, you can open VS Code by running:

```
code .
```

Working on an Existing Project

When working on an existing project, you need to `clone` them first:

```
git clone <project_url>
```

They also come with a `requirements.dev.txt` to install the development dependencies:

```
cd <project_name>

pip install -r requirements.dev.txt
```

The module dependencies are installed using the inmanta CLI tool:

```
inmanta project install
```

5.1.4 Set project sources

When starting a new project, the next step is to set the sources of your project so that it knows where to get its required modules from.

V1 module source

If you only use opensource v1 modules as provided by Inmanta, you can skip below step.

1. Find the module you want to work on
2. Copy the SSH URL of the repo
3. In your VS code, open the `project.yml` file and under `repo:`, add the copied line there but keep in mind to replace the name of a specific module with a place holder, like below example:

```
code project.yml
```

```
repo:
    - url: git@code.inmanta.com:example/my_module.git
      type: git
```

Becomes:

```
repo:
    - url: git@code.inmanta.com:example/{}.git
      type: git
```

- Now, in your `main.cf` file, if you import a module like, `import <my_module>` and save the file, you can get code completion. If you are working on an existing project with a populated `main.cf` file, code completion will work as expected.

Please note, code completion and navigation work on modules that are imported in the `main.cf` **file**.

Pip index for V2 modules and V1 modules' dependencies

Add the pip index where your modules and dependencies are hosted to `project.yml` in the `pip.index-url` *section*. For example, for modules hosted on PyPi:

```
pip:
    index-url: https://pypi.org/simple/
```

5.1.5 Setting up a module

Like projects, there are also two scenarios:

1. *Working on a New Module*.
2. *Working on an Existing Module*.

Working on a New Module

Same as *Working on a New Project* part, modules can also be created like:

```
pip install cookiecutter
cookiecutter --checkout v1 https://github.com/inmanta/inmanta-module-template.git
```

for a v1 module. If you want to use the module in a project, make sure to put it in the project's module path.

For a v2 module, use the v2 cookiecutter template, then install the module:

```
pip install cookiecutter
cookiecutter https://github.com/inmanta/inmanta-module-template.git
pip install -e ./<module-name>
```

This will install a Python package with the name `inmanta-module-<module-name>` in the active environment.

If you want to use the v2 module in a project, make sure to set up a v2 module source as outlined in the section above, then add the module as a dependency of the project as described in *Working on an Existing Module*. The location of the module directory is not important for a v2 module.

For more information on how to work on modules, see *Understanding Modules* and the module template documentation.

Working on an Existing Module

Modules that you want to work on, have to be added to your Inmanta project using the following command. This command also installs the module into the project.

```
inmanta module add --v1 <module-name>
```

for a v1 module or

```
inmanta module add --v2 <module-name>
```

for a v2 module. The latter will implicitly trust any Python package named `inmanta-module-<module-name>` in the project's configured module source.

When starting to work on an existing module, it is recommended to check the `readme.md` file that comes with the module to see the instructions on how to install and use them.

5.1.6 Running Test

To run test on modules, it is *recommended* to set the `INMANTA_TEST_ENV` environment variable to speed up your tests and avoid creating virtual environments at each test run.

1. Create a temp directory and export the path:

```
export INMANTA_TEST_ENV=$(mktemp -d)
```

2. Install required dependencies

```
pip install -r requirements.txt -r requirements.dev.txt
```

3. Run the test

```
python -m pytest tests
```

5.2 Project creation guide

This guide explains how to create a project. For detailed documentation see: *project.yml*.

5.2.1 Create a new source project

The Inmanta compiler expects a *project* with basic configuration. This project is a directory that contains the source code of the configuration model. This project also matches with a *project* defined on the server, from which multiple *environments* can be deployed.

```
pip install cookiecutter
cookiecutter gh:inmanta/inmanta-project-template
```

Note: The cookiecutter template also sets up git for the new project. This is a best practice to version control your infrastructure code.

Inside the project the compiler expects a `project.yml` file that defines metadata about the project, the location to store modules, repositories where to find modules and possibly specific versions of modules. *project.yml* provides an overview about the supported metadata attributes.

An example `project.yml` could be:

```yaml
name: test
description: a test project
author: Inmanta
author_email: code@inmanta.com
license: ASL 2.0
copyright: 2020 Inmanta
modulepath: libs
downloadpath: libs
repo:
    - url: https://github.com/inmanta/
      type: git
install_mode: release
requires:
pip:
    index-url: https://pypi.org/simple
    extra-index-url: []
    pre: false
    use-system-config: false
```

Warning: Using more than one Python package index in the project config is discouraged. It is a security risk and using more than one should be done with extreme care. Only proceed if you are aware of dependency confusion attacks. For more information see the pip documentation and the draft PEP 708

5.2.2 The main file

The `main.cf` is the place where the compiler starts executing code first. For example, the `main.cf` below calls the print plugin from the std module.

```
std::print("hello world")
```

Note: The std module is the only module that does not have to be imported explicitly.

Before the project can be executed, the std module has to be installed. This is done by executing the following command in the project directory:

```
inmanta project install
```

The example can be executed with `inmanta compile`. This prints out "hello world" on stdout.

5.3 Module creation guide

This guide explains how to create a module. For detailed documentation see: *module.yml* and *setup.cfg*.

5.3.1 Create a new source module

For a v1 module:

```
pip install cookiecutter
cookiecutter --checkout v1 gh:inmanta/inmanta-module-template
```

For a v2 module:

```
pip install cookiecutter
cookiecutter gh:inmanta/inmanta-module-template
```

Note: The cookiecutter template also sets up git for the new module. This is a best practice to version control your infrastructure code.

Inside the module the compiler expects a `module.yml` file (for v1) or a `setup.cfg` file (for v2) that defines metadata about the module. *module.yml* and *setup.cfg* provide an overview about the supported metadata attributes.

5.4 Understanding Modules

In Inmanta all orchestration model code and related files, templates, plugins and resource handlers are packaged in a module. Modules can be defined in two different formats, the V1 format and the V2 format. The biggest difference between both formats is that all Python tools can run on V2 modules, because V2 modules are essentially Python packages. New modules should use the V2 module format. The following sections describe the directory layout of the V1 and the V2 module formats and their metadata files.

Note: V2 modules can not depend on V1 modules.

5.4.1 V2 module format

A complete V2 module might contain the following files:

```
module
|
|__ MANIFEST.in
|__ setup.cfg
|__ pyproject.toml
|
|__ model
|    |__ _init.cf
|    |__ services.cf
|
|__ inmanta_plugins/<module-name>/
|    |__ __init__.py
|    |__ functions.py
|
|__ files
|    |__ file1.txt
|
|__ templates
     |__ conf_file.conf.tmpl
```

- The root of the module directory contains a `setup.cfg` file. This is the metadata file of the module. It contains information, such as the version of the module. More details about the `setup.cfg` file are defined in the next section.
- The `pyproject.toml` file defines the build system that should be used to package the module and install the module into a virtual environment from source.
- The only mandatory subdirectory is the `model` directory containing a file called `_init.cf`. What is defined in the `_init.cf` file is available in the namespace linked with the name of the module. Other files in the model directory create subnamespaces.
- The `inmanta_plugins/<module-name>/` directory contains Python files that are loaded by the platform and can extend it using the Inmanta API. This python code can provide plugins or resource handlers.

The template, file and source plugins from the std module expect the following directories as well:

- The `files` directory contains files that are deployed verbatim to managed machines.
- The `templates` directory contains templates that use parameters from the orchestration model to generate configuration files.

The setup.cfg metadata file

The `setup.cfg` file defines metadata about the module. The following code snippet provides an example about what this `setup.cfg` file looks like:

```
[metadata]
name = inmanta-module-mod1
version = 1.2.3
license = Apache 2.0

[options]
install_requires =
  inmanta-modules-net ~=0.2.4
  inmanta-modules-std >1.0,<2.5

```

(continues on next page)

(continued from previous page)

```
  cookiecutter~=1.7.0
  cryptography>1.0,<3.5

[options.extras_require]
feature-x =
  inmanta-modules-mod2

zip_safe=False
include_package_data=True
packages=find_namespace:

[options.packages.find]
include = inmanta_plugins*
```

- The `metadata` section defines the following fields:
  - `name`: The name of the resulting Python package when this module is packaged. This name should follow the naming schema: `inmanta-module-<module-name>`.
  - `version`: The version of the module. Modules must use semantic versioning.
  - `license`: The license under which the module is distributed.
  - `deprecated`: Optional field. If set to True, this module will print a warning deprecation message when used.
- The `install_requires` config option in the `options` section of the `setup.cfg` file defines the dependencies of the module on other Inmanta modules and external Python libraries. These version specs use PEP440 syntax. Adding a new module dependency to the module should be done using the `inmanta module add` command instead of altering the `setup.cfg` file by hand. Dependencies with extras can be defined in this section using the `dependency[extra-a,extra-b]` syntax.
- The `options.extras_require` config option can be used to define optional dependencies, only required by a specific feature of the inmanta module.

A full list of all available options can be found in *here*.

The pyproject.toml file

The `pyproject.toml` file defines the build system that has to be used to build a python package and perform editable installs. This file should always have the following content:

```
[build-system]
requires = ["setuptools", "wheel"]
build-backend = "setuptools.build_meta"
```

The MANIFEST.in file

This file enables `setuptools` to correctly build the package. It is documented here. An example that includes the model, files, templates and metadata file in the package looks like this:

```
include inmanta_plugins/mod1/setup.cfg
recursive-include inmanta_plugins/mod1/model *.cf
graft inmanta_plugins/mod1/files
graft inmanta_plugins/mod1/templates
```

You might notice that the model, files and templates directories, nor the metadata file reside in the `inmanta_plugins` directory. The inmanta build tool takes care of this to ensure the included files are included in the package installation directory.

5.4.2 V1 module format

A complete module might contain the following files:

```
module
|
|__ module.yml
|
|__ model
|    |__ _init.cf
|    |__ services.cf
|
|__ plugins
|    |__ functions.py
|
|__ files
|    |__ file1.txt
|
|__ templates
|    |__ conf_file.conf.tmpl
|
|__ requirements.txt
```

The directory layout of the V1 module is similar to that of a V2 module. The following difference exist:

- The metadata file of the module is called `module.yml` instead of `setup.cfg`. The structure of the `module.yml` file also differs from the structure of the `setup.cfg` file. More information about this `module.yml` file is available in the next section.
- The files contained in the `inmanta_plugins/<module-name>/` directory in the V2 format, are present in the `plugins` directory in the V1 format.
- The `requirements.txt` file defines the dependencies of this module to other V2 modules and the dependencies to external libraries used by the code in the `plugins` directory. This file is not present in the V2 module format, since V2 modules defined their dependencies in the `setup.cfg` file. Dependencies with extras are supported in the `requirements.txt` file using the `dependency[extra-a,extra-b]` syntax.
- The `pyproject.toml` file doesn't exist in a V1 module, because V1 modules cannot be packaged into a Python package.

Module metadata

The module.yml file provides metadata about the module. This file is a yaml file with the following three keys mandatory:

- *name*: The name of the module. This name should also match the name of the module directory.
- *license*: The license under which the module is distributed.
- *version*: The version of this module. For a new module a start version could be 0.1dev0 These versions are parsed using the same version parser as python setuptools.
- *deprecated*: Optional field. If set to True, this module will print a warning deprecation message when used.

For example the following module.yml from the *Quickstart*

```yaml
name: lamp
license: Apache 2.0
version: 0.1
```

The *requires* key can be used to define the dependencies of this module on other Inmanta modules. Each entry in the list should contain the name of an Inmanta module, optionally associated with a version constraint. These version specs use PEP440 syntax. Adding a new entry to the requires list should be done using the `inmanta module add <module-name>` command.

An example of a `module.yml` file that defines requires:

```
license: Apache 2.0
name: ip
source: git@github.com:inmanta/ip
version: 0.1.15
requires:
    - net ~= 0.2.4
    - std >1.0 <2.5
```

`source` indicates the authoritative repository where the module is maintained.

A full list of all available options can be found in *here*.

5.4.3 Convert a module from V1 to V2 format

To convert a V1 module to the V2 format, execute the following command in the module folder

```
inmanta module v1tov2
```

5.4.4 Inmanta module template

To quickly initialize a module use the *inmanta module template*.

5.4.5 Extending Inmanta

Inmanta offers module developers an orchestration platform with many extension possibilities. When modelling with existing modules is not sufficient, a module developer can use the Python SDK of Inmanta to extend the platform. Python code that extends Inmanta is stored in the plugins directory of a module. All python modules in the plugins subdirectory will be loaded by the compiler when at least a `__init__.py` file exists, exactly like any other python package.

The Inmanta Python SDK offers several extension mechanism:

- Plugins
- Resources
- Resource handlers
- Dependency managers

Only the compiler and agents load code included in modules (See *Architecture*). A module can include external dependencies. Both the compiler and the agent will install this dependencies with `pip install` in an virtual environment dedicated to the compiler or agent. By default this is in *.env* of the project for the compiler and in */var/lib/inmanta/agent/env* for the agent.

Inmanta uses a special format of requirements that was defined in python PEP440 but never fully implemented in all python tools (setuptools and pip). Inmanta rewrites this to the syntax pip requires. This format allows module developers to specify a python dependency in a repo on a dedicated branch. And it allows inmanta to resolve the requirements of all module to a single set of requirements, because the name of module is unambiguously defined in the requirement. The format for requires in requirements.txt is the following:

- Either, the name of the module and an optional constraint
- Or a repository location such as git+https://github.com/project/python-foo The correct syntax to use is then: eggname@git+https://../repository#branch with branch being optional.

5.5 Installing modules

Since modules often have dependencies on other modules, it is common to develop against multiple modules (or a project and one or more modules) simultaneously. One might for example need to extend a dependent module to add support for some new feature. Because this use case is so common, this section will describe how to work on multiple modules simultaneously so that any changes are visible to the compiler. This procedure is of course applicable for working on a single module as well.

5.5.1 Setting up the dev environment

To set up the development environment for a project, activate your development Python environment and install the project with `inmanta project install`. To set up the environment for a single v2 module, run `pip install -e .` instead.

The following subsections explain any additional steps you need to take if you want to make changes to one of the dependent modules as well.

v1 modules

Any modules you find in the project's `modulepath` after starting from a clean project and setting up the development environment are v1 modules. You can make changes to these modules and they will be reflected in the next compile. No additional steps are required.

v2 modules

All other modules are v2 and have been installed by `inmanta project install` into the active Python environment. If you want to be able to make changes to one of these modules, the easiest way is to check out the module repo separately and run `pip install -e <path>` on it, overwriting the published package that was installed previously. This will install the module in editable form: any changes you make to the checked out files will be picked up by the compiler. You can also do this prior to installing the project, in which case the pre-installed module will remain installed in editable form when you install the project, provided it matches the version constraints. Since these modules are essentially Python packages, you can double check the desired modules are installed in editable mode by checking the output of `pip list --editable`.

5.5.2 Working on the dev environment

After setting up, you should be left with a dev environment where all required v2 modules have been installed (either in editable or in packaged form). If you're working on a project, all required v1 modules should be checked out in the `modulepath` directory.

When you run a compile from the active Python environment context, the compiler will find both the v1 and v2 modules and use them for both their model and their plugins.

Similarly, when you run a module's unit tests, the installed v2 modules will automatically be used by the compiler. As for v1 modules, by default, the `pytest-inmanta` extension makes sure the compile itself runs against an isolated project, downloading any v1 module dependencies. If you want to compile against local versions of v1 modules, have a look at the `--use-module-in-place` option in the `pytest-inmanta` documentation.

5.5.3 Module installation on the server

The orchestrator server generally installs modules from the configured Python package repository, respecting the project's constraints on its modules and all inter-module constraints. The server is then responsible for supplying the agents with the appropriate `inmanta_plugins` packages.

The only exception to this rule is when using the `inmanta export` command. It exports a project and all its modules' `inmanta_plugins` packages to the orchestrator server. When this method is used, the orchestrator does not install any modules from the Python package repository but instead contains all Python code as present in the local Python environment.

Configure the Inmanta server to install modules from a private python package repository

V2 modules can be installed from a Python package repository that requires authentication. This section explains how the Inmanta server should be configured to install v2 modules from such a Python package repository.

Create a file named `/var/lib/inmanta/.netrc` in the orchestrator's file system. Add the following content to the file:

```
machine <hostname of the private repository>
login <username>
password <password>
```

For more information see the doc about pip authentication.

You will also need to specify the url of the repository in the `project.yml` file of your project (See: *Configure pip index*).

By following the previous steps, the Inmanta server will be able to install modules from a private Python package repository.

5.5.4 Inter-module dependencies

The plugins code of a module mod-a can have a dependency on the plugins code of another V2 module mod-b. When doing this, care should be taken that the module(s) you depend on, do not define any resources or providers. Otherwise the python environment of the agent can get corrupt in the following way:

1. The configuration model (in the project or one of the modules) constructs resources from both modules mod-a and mod-b.
2. mod-a-1.0 and mod-b-1.0 are exported: The exporter exports the x.py file to the server and the agent puts the x.py file in its code directory.
3. The configuration model is changed to only construct resources from module mod-a.
4. mod-a-1.0 is exported again. mod-b-1.0 is no longer exported because it doesn't have any resources. The x.py file still exists in the agent code directory.
5. A new version of module mod-b (mod-b-2.0) is released and included in the project. The project is re-exported: mod-a-1.0 is exported again, mod-b-2.0 is not (again because it doesn't have any resources). The old x.py file still exists in the agent code directory. It is loaded by the agent instead of the one from mod-b-2.0.

This issue will be resolved by a restart of the agent process.

5.6 Releasing and distributing modules

5.6.1 V2 modules

Distributing V2 modules

V2 modules are distributed as Python packages. To build a package for a module, run `inmanta module build` in the source directory of the module. The resulting Python wheel can then be found in the dist directory of the module. You can then publish this to the Python package repository of your choice, for example the public PyPi repository. The inmanta build tool will package a module named `my_module` under the name `inmanta-module-my-module`.

5.6.2 V1 modules

Inmanta V1 modules are versioned based on git tags. The current version is reflected in the `module.yml` file. The commit should be tagged with the version in the git repository as well. To release a module, use the release command as outlined below.

Development Versions

To make changes to a module, first create a new git branch:

```
git checkout -b mywork
```

When done, first use git to add files:

```
git add *
```

Create a new dev version:

```
inmanta module release --dev --patch -m "Fixed small bug"
```

This command will set the version to the next dev version, e.g. `+0.0.1dev` for a patch increment.

The module tool supports semantic versioning. Use one of `--major`, `--minor` or `--patch` to update version numbers: `<major>.<minor>.<patch>`

For the dev releases, no tags are created. Once the dev version is ready for release, perform a proper release by following the steps in the *Release Versions* section below.

Release Versions

To perform an actual stable release, checkout the main development branch and use the `inmanta module release` command:

```
inmanta module release
git push
git push origin {tag}
```

This will create a stable version corresponding to the current dev version without the `.dev` and tag it. This will also setup the main development branch for further development by creating a new dev version that is a patch ahead of the latest released version.

Distributing V1 modules

V1 modules are generally simply distributed using a Git repository. They can however also be built as a V2 Python package and distributed the same as other V2 modules.

Git repository distribution format

Distributing a V1 module using a Git repository happens by storing the source code of that module on a Git repository that is accessible by the Inmanta orchestrator. The orchestrator will clone the source code of the module and install it in the Inmanta project. Tagging release versions as outlined above allows specifying constraints on the module version.

V2 package distribution format

A V2 package can be built for a V1 module with `inmanta module build`. This package can be distributed as described in *Distributing V2 modules*. Modules installed from a package will always act as V2 modules and will be considered such by the compiler.

5.6.3 Freezing a project

Prior to releasing a new stable version of an inmanta project, you might wish to freeze its module dependencies. This will ensure that the orchestrator server will always work with the exact versions specified. You can achieve this with `inmanta project freeze --recursive --operator "=="`. This command will freeze all module dependencies to their exact version as they currently exist in the Python environment. The recursive option makes sure all module dependencies are frozen, not just the direct dependencies. In other words, if the project depends on module `a` which in turn depends on module `b`, both modules will be pinned to their current version in `setup.cfg`.

5.7 Developing Plugins

5.7.1 Adding new plugins

Plugins provide *functions* that can be called from the *DSL*. This is the primary mechanism to interface Python code with the orchestration model at compile time. For Example, this mechanism is also used for std::template and std::file. In addition to this, Inmanta also registers all plugins with the template engine (Jinja2) to use as filters.

A plugin is a python function, registered with the platform with the *`plugin()`* decorator. This plugin accepts arguments when called from the DSL and can return a value. Both the arguments and the return value must by annotated with the allowed types from the orchestration model. Type annotations are provided as a string (Python3 style argument annotation). `any` is a special type that effectively disables type validation.

Through the arguments of the function, the Python code in the plugin can navigate the orchestration model. The compiler takes care of scheduling the execution at the correct point in the model evaluation.

Note: A module's Python code lives in the `inmanta_plugins.<module_name>` namespace.

A simple plugin that accepts no arguments, prints out "hello world" and returns no value requires the following code:

```python
from inmanta.plugins import plugin

@plugin
def hello() -> None:
    print("Hello world!")
```

If the code above is placed in the plugins directory of the example module (`examples/plugins/__init__.py`) the plugin can be invoked from the orchestration model as follows:

```
import example

example::hello()
```

The plugin decorator accepts an argument name. This can be used to change the name of the plugin in the DSL. This can be used to create plugins that use python reserved names such as `print` for example:

```
from inmanta.plugins import plugin

@plugin("print")
def printf() -> None:
    """
        Prints inmanta
    """
    print("inmanta")
```

A more complex plugin accepts arguments and returns a value. Compared to what python supports as function arguments, only positional-only arguments are not supported. The following example creates a plugin that converts a string to uppercase:

```
from inmanta.plugins import plugin

@plugin
def upper(value: "string") -> "string":
    return value.upper()
```

This plugin can be tested with:

```
import example

std::print(example::upper("hello world"))
```

Argument type annotations are strings that refer to Inmanta primitive types or to entities. If an entity is passed to a plugin, the python code of the plugin can navigate relations throughout the orchestration model to access attributes of other entities.

A base exception for plugins is provided in `inmanta.plugins.PluginException`. Exceptions raised from a plugin should be of a subtype of this base exception.

```
from inmanta.plugins import plugin, PluginException

@plugin
def raise_exception(message: "string") -> None:
    raise PluginException(message)
```

If your plugin requires external libraries, add them as dependencies of the module. For more details on how to add dependencies see *Understanding Modules*.

5.7.2 Deprecate plugins

To deprecate a plugin the `deprecated()` decorator can be used in combination with the *plugin()* decorator. Using this decorator will log a warning message when the function is called. This decorator also accepts an optional argument `replaced_by` which can be used to potentially improve the warning message by telling which other plugin should be used in the place of the current one.

for example if the plugin below is called:

```python
from inmanta.plugins import plugin, deprecated

@deprecated(replaced_by="my_new_plugin")
@plugin
def printf() -> None:
    """
        Prints inmanta
    """
    print("inmanta")
```

it will give following warning:

```
Plugin 'printf' in module 'inmanta_plugins.<module_name>' is deprecated. It should be␣
↪replaced by 'my_new_plugin'
```

Should the replace_by argument be omitted, the warning would look like this:

```
Plugin 'printf' in module 'inmanta_plugins.<module_name>' is deprecated.
```

If you want your module to stay compatible with older versions of inmanta you will also need to add a little piece of code that changes how `deprecated()` is imported as it does not exist in all versions.

The previous example would then look like this. For older inmanta versions, replace the decorator with a no-op.

```python
from inmanta.plugins import plugin

try:
    from inmanta.plugins import deprecated
except ImportError:
    deprecated = lambda function=None, **kwargs: function if function is not None␣
↪else deprecated


@deprecated(replaced_by="my_new_plugin")
@plugin
def printf() -> None:
    """
        Prints inmanta
    """
    print("inmanta")
```

5.8 Finalizers

When writing models it can be useful to have functions that will be run at the end of the compilation. A typical use case is making sure all resources are properly flushed back and all connections are properly closed. To help with this, finalizers can be used.

5.8.1 Adding new finalizers

A finalizer is a python function that is registered by using the `finalizer()` function as decorator or as callback. This function should be a function that doesn't take arguments and that doesn't return anything. Functions registered this way will be call when the compiler finishes (with no guarantee on the execution order).

an example of a finalizer that will close an open connection using the decorator option requires the following code:

```python
from inmanta import compiler

connection = None

def get_connection():
   global connection
   if connection is None:
        connection = connect()
   return connection

@compiler.finalizer
def finalize_connection():
   if connection:
      connection.close()
```

the same example but using the callback option would look like this:

```python
from inmanta import compiler

connection = None

def get_connection():
  global connection
  if not connection:
       connection = connect()
       compiler.finalizer(finalize_connection)
   return connection

def finalize_connection():
   if connection:
      connection.close()
```

5.9 Developing South Bound Integrations

The inmanta orchestrator comes with a set of integrations with different platforms (see: *Inmanta modules*). But it is also possible to develop your own south bound integrations.

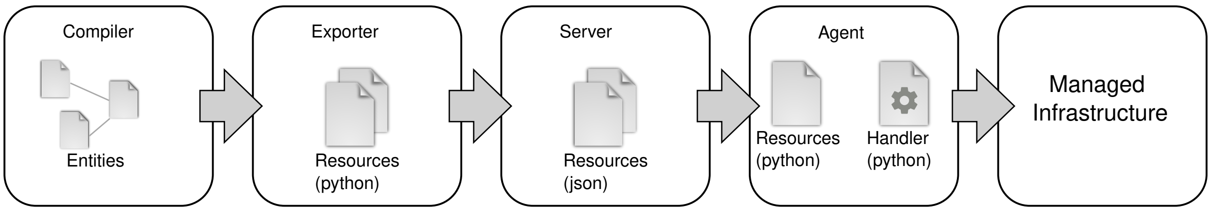
To integrate a new platform into the orchestrator, you must take the following steps:

1. Create a new module to contain the integration (see: *Understanding Modules*).
2. Model the target platform as set of *entities*.
3. Create *resources* and *handler*, as described below.

5.9.1 Overview

A South Bound integration always consists of three parts:

- one or more *entities* in the model
- a *resource* that serializes the entities and captures all information required to enforce the *desired state*.
- a *handler*: the python code required to enforce the desired state.



- In the *compiler*, a model is constructed that consists of entities. The entities can be related to each other.
- The *exporter* will search for all *entities* that can be directly deployed by a *handler*. These are the *resources*. Resources are self-contained and can not refer to any other entity or resource.
- The *resources* will be sent to the server in json serialized form.
- The *agent* will present the *resources* to a *handler* in order to have the *desired state* enforced on the managed infrastructure.

5.9.2 Resource

A resource is represented by a Python class that is registered with Inmanta using the `@resource` decorator. This decorator decorates a class that inherits from the `Resource` class.

The fields of the resource are indicated with a `fields` field in the class. This field is a tuple or list of strings with the name of the desired fields of the resource. The orchestrator uses these fields to determine which attributes of the matching entity need to be included in the resource.

Fields of a resource cannot refer to an instance in the orchestration model or fields of other resources. The resource serializers allows to map field values. Instead of referring directly to an attribute of the entity it serializes (path in std::File and path in the resource map one on one). This mapping is done by adding a static method to the resource class with `get_$(field_name)` as name. This static method has two arguments: a reference to the exporter and the instance of the entity it is serializing.

```python
from inmanta.resources import resource, Resource

@resource("std::File", agent="host.name", id_attribute="path")
class File(Resource):
    fields = ("path", "owner", "hash", "group", "permissions", "purged", "reload")

```

(continues on next page)

(continued from previous page)

```
    @staticmethod
    def get_hash(exporter, obj):
        hash_id = md5sum(obj.content)
        exporter.upload_file(hash_id, obj.content)
        return hash_id

    @staticmethod
    def get_permissions(_, obj):
        return int(x.mode)
```

Classes decorated with *@resource* do not have to inherit directly from *Resource*. The orchestrator already offers two additional base classes with fields and mappings defined: *PurgeableResource* and *ManagedResource*. This mechanism is useful for resources that have fields in common.

A resource can also indicate that it has to be ignored by raising the *IgnoreResourceException* exception.

5.9.3 Handler

Handlers interface the orchestrator with resources in the *infrastructure*. Handlers take care of changing the current state of a resource to the desired state expressed in the orchestration model.

The compiler collects all python modules from Inmanta modules that provide handlers and uploads them to the server. When a new orchestration model version is deployed, the handler code is pushed to all agents and imported there.

Handlers should inherit the class *CRUDHandler*. The *@provider* decorator registers the class with the orchestrator.

Each Handler should override 4 methods of the CRUDHandler:

1. *read_resource()* to read the current state of the system.
2. *create_resource()* to create the resource if it doesn't exist.
3. *update_resource()* to update the resource when required.
4. *delete_resource()* to delete the resource when required.

The context (See *HandlerContext*) passed to most methods is used to report results, changes and logs to the handler and the server.

Using facts

Facts are properties of the environment whose values are not managed by the orchestrator. Facts are either used as input in a model, e.g. a virtual machine provider provides an ip and the model then uses this ip to run a service, or used for reporting purposes.

Retrieving a fact in the model is done with the std::getfact() function.

Example taken from the openstack Inmanta module:

```
implementation fipAddr for FloatingIP:
    self.address = std::getfact(self, "ip_address")
end
```

Facts can be pushed or pulled through the handler.

Pushing a fact is done in the handler with the *set_fact()* method during resource deployment (in `read_resource` and/or `create_resource`). e.g.:

```python
@provider("openstack::FloatingIP", name="openstack")
class FloatingIPHandler(OpenStackHandler):
    def read_resource(self, ctx: handler.HandlerContext, resource: FloatingIP) ->␣
↪None:
        ...

    def create_resource(self, ctx: handler.HandlerContext, resource: FloatingIP) ->␣
↪None:
        ...
        # Setting fact manually
        for key, value in ...:
            ctx.set_fact(fact_id=key, value=value, expires=True)
```

By default, facts expire when they have not been refreshed or updated for a certain time, controlled by the *server.fact-expire* config option. Querying for an expired fact will force the agent to refresh it first.

When reporting a fact, setting the `expires` parameter to `False` will ensure that this fact never expires. This is useful to take some load off the agent when working with facts whose values never change. On the other hand, when working with facts whose values are subject to change, setting the `expires` parameter to `True` will ensure they are periodically refreshed.

Facts are automatically pulled periodically (this time interval is controlled by the *server.fact-renew* config option) when they are about to expire or if a model requested them and they were not present. The server periodically asks the agent to call into the handler's *facts()* method. e.g.:

```python
@provider("openstack::FloatingIP", name="openstack")
class FloatingIPHandler(OpenStackHandler):
    ...

    def facts(self, ctx, resource):
        port_id = self.get_port_id(resource.port)
        fip = self._neutron.list_floatingips(port_id=port_id)["floatingips"]
        if len(fip) > 0:
            ctx.set_fact("ip_address", fip[0]["floating_ip_address"])
```

Warning: If you ever push a fact that does expire, make sure it is also returned by the handler's `facts()` method. If you omit to do so, when the fact eventually expires, the agent will keep on trying to refresh it unsuccessfully.

Note: Facts should not be used for things that change rapidly (e.g. cpu usage), as they are not intended to refresh very quickly.

5.9.4 Built-in Handler utilities

The *Inmanta Agent*, responsible for executing handlers has built-in utilities to help handler development. This section describes the most important ones.

Logging

The agent has a built-in logging facility, similar to the standard python logger. All logs written to this logger will be sent to the server and are available via the web-console and the API. Additionally, the logs go into the agent's logfile and into the resource-action log on the server.

To use this logger, use one of the methods: `ctx.debug`, `ctx.info`, `ctx.warning`, `ctx.error`, `ctx.critical` or `ctx.exception`.

This logger implements the *~inmanta.agent.handler.LoggerABC* logging interface and supports kwargs. The kwargs have to be json serializable. They will be available via the API in their json structured form.

For example:

```
def create_resource(self, ctx: HandlerContext, resource: ELB) -> None:
    # ...
    ctx.debug("Creating loadbalancer with security group %(sg)s", sg=sg_id)
```

An alternative implementation of the *~inmanta.agent.handler.LoggerABC* logging interface that just logs to the Python logger is provided in *~inmanta.agent.handler.PythonLogger*. This logger is not meant to be used in actual handlers but it can be used for the automated testing of helper methods that accept a *~inmanta.agent.handler.LoggerABC* instance. In production, these helpers would receive the actual `HandlerContext` and log appropriately, while for testing the *PythonLogger* can be passed.

Caching

The agent maintains a cache, that is kept over handler invocations. It can, for example, be used to cache a connection, so that multiple resources on the same device can share a connection.

The cache can be invalidated either based on a timeout or on version. A timeout based cache is kept for a specific time. A version based cache is used for all resource in a specific version. The cache will be dropped when the deployment for this version is ready.

The cache can be used through the `@cache` decorator. Any method annotated with this annotation will be cached, similar to the way lru_cache works. The arguments to the method will form the cache key, the return value will be cached. When the method is called a second time with the same arguments, it will not be executed again, but the cached result is returned instead. To exclude specific arguments from the cache key, use the *ignore* parameter.

For example, to cache the connection to a specific device for 120 seconds:

```
@cache(timeout=120, ignore=["ctx"])
def get_client_connection(self, ctx, device_id):
   # ...
   return connection
```

To do the same, but additionally also expire the cache when the next version is deployed, the method must have a parameter called *version*. *for_version* is True by default, so when a version parameter is present, the cache is version bound by default.

```
@cache(timeout=120, ignore=["ctx"], for_version=True)
def get_client_connection(self, ctx, device_id, version):
   # ...
   return connection
```

To also ensure the connection is properly closed, an *on_delete* function can be attached. This function is called when the cache is expired. It gets the cached item as argument.

```
@cache(timeout=120, ignore=["ctx"], for_version=True,
   call_on_delete=lambda connection:connection.close())
def get_client_connection(self, ctx, device_id, version):
```

(continues on next page)

(continued from previous page)

```
    # ...
    return connection
```

5.10 Test plugins

Testing the behavior of an Inmanta plugin can be done by using the `project` fixture, which is part of the `pytest-inmanta` package. This fixture provides functionality to call a plugin directly from a pytest test case.

5.10.1 Install the pytest-inmanta package

The `pytest-inmanta` package can be installed via pip:

```
pip install pytest-inmanta
```

5.10.2 Writing a test case

Take the following plugin as an example:

```
# example_module/plugins/__init__.py

from inmanta.plugins import plugin

@plugin
def hostname(fqdn: "string") -> "string":
    """
        Return the hostname part of the fqdn
    """
    return fqdn.split(".")[0]
```

A test case, to test this plugin looks like this:

```
1  # example_module/tests/test_hostname.py
2
3  def test_hostname(project, inmanta_plugins):
4      host = "test"
5      fqdn = f"{host}.something.com"
6      assert inmanta_plugins.example_module.hostname(fqdn) == host
```

- **Line 3:** Creates a pytest test case, which requires the `project` fixture.
- **Line 6:** Uses the `inmanta_plugins` fixture to access the `hostname` function from the `example_module` module's Python namespace. As such, this line tests whether `host` is returned when the plugin function `hostname` is called with the parameter `fqdn`.

Note: V2 modules do not need to use the `inmanta_plugins` fixture. They can just import from the `inmanta_plugins` namespace directly at the top of the test file.

For more information see: pytest-inmanta

5.11 Understanding Projects

A project is the basic unit of orchestration. It contains:

- `main.cf`: the entry point for the compiler to start executing
- `project.yml`: the project meta data, defines where to find modules and which versions to use. For detailed documentation see: *project.yml*.
- `requirements.txt`: (optional) the python dependencies of the project, defines which python dependencies to install and which versions to use. Dependencies with extras can be defined in this file using the `dependency[extra-a,extra-b]` syntax. It has two main use cases:
  - It contains the listing of all modules that should be installed as a V2 module.
  - It contains version constraints to help pip resolve version conflicts on python packages.

```
project
|
|__ project.yml
|__ requirements.txt
|__ main.cf
```

5.12 Model debugging

Warning: This is a beta feature. It does not support the full language yet and it might not work as expected. Currently known limitations:

- lists and dicts not supported
- string interpolation not supported
- constructor kwargs not supported
- plugins not supported
- conditionals not supported
- for loops not supported
- boolean operations not supported
- explicit index lookups not supported
- only double assignment, exceeding relation arity and incomplete instance errors are supported

Support for the listed language features will be added gradually.

The inmanta DSL is essentially a data flow oriented language. As a model developer you never explicitly manipulate control flow. Instead you declare data flow: the statement `x = y` for example declares that the data in `y` should flow towards `x`. Even dynamic statements such as implementations and for loops do not explicitly manipulate control flow. They too can be interpreted as data flow declarations.

Because of this property conventional debugging methods such as inspecting a stack trace are not directly applicable to the inmanta language. A stack trace is meant to give the developer insight in the part of the control flow that led to the error. Extending this idea to the inmanta DSL leads to the concept of a data trace. Since the language is data flow oriented, a trace of the flow to some erroneous part of the configuration model gives the developer insight in the cause of the error.

Additionally, a root cause analysis will be done on any incomplete instances and only those root causes will be reported.

The first section, *Enabling the data trace* describes how to enable these two tools. The tools themselves are described in the sections *Interpreting the data trace* and *Root cause analysis* respectively. An example use case is shown in *Usage example*, and the final section, *Graphic visualization*, shortly describes a graphic representation of the data flow.

5.12.1 Enabling the data trace

To show a data trace when an error occurs, compile the model with the `--experimental-data-trace` flag. For example:

Listing 1: main.cf

```
x = 1
x = 2
```

Compiling with `inmanta compile --experimental-data-trace` results in

```
inmanta.ast.DoubleSetException: value set twice:
    old value: 1
        set at ./main.cf:1
    new value: 2
        set at ./main.cf:2

data trace:
x
├── 1
│   SET BY `x = 1`
│   AT ./main.cf:1
└── 2
    SET BY `x = 2`
    AT ./main.cf:2
 (reported in x = 2 (./main.cf:2))
```

5.12.2 Interpreting the data trace

Let's have another look at the data trace for the model above:

```
1 x
2 ├── 1
3 │   SET BY `x = 1`
4 │   AT ./main.cf:1
5 └── 2
6     SET BY `x = 2`
7     AT ./main.cf:2
```

Line 1 shows the variable where the error occurred. A tree departs from there with branches going to lines 2 and 5 respectively. These branches indicate the data flow to `x`. In this case line 2 indicates `x` has been assigned the literal 1 by the statement `x = 1` at `main.cf:1` and the literal 2 by the statement `x = 2` at `main.cf:2`.

Now let's go one step further and add an assignment to another variable.

Listing 2: variable-assignment.cf

```
x = 0
x = y
y = 1
```

Listing 3: data trace for variable-assignment.cf

```
x
├── y
│   SET BY `x = y`
│   AT ./variable-assignment.cf:2
│   └── 1
│       SET BY `y = 1`
│       AT ./variable-assignment.cf:3
└── 0
    SET BY `x = 0`
    AT ./variable-assignment.cf:1
```

As before we can see the data flow to x as declared in the model. Following the tree from x to its leaves leads to the conclusion that x has indeed received two inconsistent values, and it gives insight into how those values came to be assigned to x (0 directly and 1 via y).

One more before we move on to entities:

Listing 4: assignment-loop.cf

```
x = y
y = z
z = x

x = 0
z = u
u = 1
```

Listing 5: data trace for assignment-loop.cf

```
1  z
2  EQUIVALENT TO {x, y, z} DUE TO STATEMENTS:
3      `x = y` AT ./assignment-loop.cf:1
4      `y = z` AT ./assignment-loop.cf:2
5      `z = x` AT ./assignment-loop.cf:3
6  ├── u
7  │   SET BY `z = u`
8  │   AT ./assignment-loop.cf:6
9  │   └── 1
10 │       SET BY `u = 1`
11 │       AT ./assignment-loop.cf:7
12 └── 0
13     SET BY `x = 0`
14     AT ./assignment-loop.cf:5
```

This model defines an assignment loop between x, y and z. Assignment to either of these variables will result in a flow of data to all of them. In other words, the variables are equivalent. The data trace shows this information at lines 2–5 along with the statements that caused the equivalence. The rest of the trace is similar to before, except that the tree now shows all assignments to any of the three variables part of the equivalence. The tree now no longer shows just the data flow to x but to the equivalence as a whole, since any data that flows to the equivalence will also flow to x.

Listing 6: entities.cf

```
entity A:
    number n
end
```

(continues on next page)

(continued from previous page)

```

implement A using std::none

x = A(n = 0)

template = x

y = A(n = template.n)
y.n = 1
```

Listing 7: data trace for entities.cf

```
 1  attribute n on __config__::A instance
 2  SUBTREE for __config__::A instance:
 3      CONSTRUCTED BY `A(n=template.n)`
 4      AT ./entities.cf:11
 5  ├── template.n
 6  │   SET BY `A(n=template.n)`
 7  │   AT ./entities.cf:11
 8  │   SUBTREE for template:
 9  │       └── x
10  │           SET BY `template = x`
11  │           AT ./entities.cf:9
12  │           └── __config__::A instance
13  │               SET BY `x = A(n=0)`
14  │               AT ./entities.cf:7
15  │               CONSTRUCTED BY `A(n=0)`
16  │               AT ./entities.cf:7
17  │   └── 0
18  │       SET BY `A(n=0)`
19  │       AT ./entities.cf:7
20  └── 1
21      SET BY `y.n = 1`
22      AT ./entities.cf:12
```

As usual, line 1 states the variable that represents the root of the data flow tree. In this case it's the attribute `n` of an instance of `A`. Which instance? That is shown in the subtree for that instance on lines 2–4. In this case it's a very simple subtree that shows just the construction of the instance and the line number in the configuration model. The tree for the attribute starts at line 5. The first branch shows the assignment to `template.n` in the constructor for `y`. Then another subtree is shown at lines 8–16, this one more useful. It shows a data flow graph like we're used to by now, with `template` as the root. Then at line 17 the trace shows the data flow `template.n <- 0` referring to `entities.cf:7`. This line doesn't assign to `template.n` directly, but it does assign to the instance at the end of the subtree for `template` (the data that flows to `template`).

Let's have a look at an implementation:

Listing 8: implementation.cf

```
entity A:
    number n
end

implement A using i

implementation i for A:
    self.n = 42
end
```

(continues on next page)

(continued from previous page)

```

x = A(n = 0)
```

Listing 9: data trace for implementation.cf

```
 1  attribute n on __config__::A instance
 2  SUBTREE for __config__::A instance:
 3      CONSTRUCTED BY `A(n=0)`
 4      AT ./implementation.cf:11
 5  ├── 0
 6  │   SET BY `A(n=0)`
 7  │   AT ./implementation.cf:11
 8  └── 42
 9      SET BY `self.n = 42`
10      AT ./implementation.cf:8
11      IN IMPLEMENTATION WITH self = __config__::A instance
12          CONSTRUCTED BY `A(n=0)`
13          AT ./implementation.cf:11
```

The only thing new in this trace can be found at lines 11—13. It highlights that a statement was executed within a dynamic context and shows a subtree for the `self` variable.

And finally, an index:

Listing 10: index.cf

```
entity A:
    number n
    number m
end

index A(n)

implement A using std::none

A(n = 42, m = 0)
A(n = 42, m = 1)
```

Listing 11: data trace for index.cf

```
1  attribute m on __config__::A instance
2  SUBTREE for __config__::A instance:
3      CONSTRUCTED BY `A(n=42,m=0)`
4      AT ./index.cf:10
5
6      INDEX MATCH: `__config__::A instance`
7          CONSTRUCTED BY `A(n=42,m=1)`
8          AT ./index.cf:11
9  ├── 1
10 │   SET BY `A(n=42,m=1)`
11 │   AT ./index.cf:11
12 └── 0
13     SET BY `A(n=42,m=0)`
14     AT ./index.cf:10
```

This data trace highlights the index match between the two constructors at lines 6–8.

5.12.3 Root cause analysis

Enabling the data trace also enables a root cause analysis when multiple attributes have not received a value. For example, compiling the model below results in three errors, one for each of the instances.

```
entity A:
    number n
end

implement A using std::none

x = A()
y = A()
z = A()

x.n = y.n
y.n = z.n
```

Listing 12: compile output

```
Reported 3 errors
error 0:
  The object __config__::A (instantiated at ./main.cf:7) is not complete: attribute n (./main.cf:2) is not set
error 1:
  The object __config__::A (instantiated at ./main.cf:9) is not complete: attribute n (./main.cf:2) is not set
error 2:
  The object __config__::A (instantiated at ./main.cf:8) is not complete: attribute n (./main.cf:2) is not set
```

Compiling with data trace enabled will do a root cause analysis on these errors. In this case it will infer that `x.n` and `y.n` are only unset because `z.n` is unset. Compiling then shows:

Listing 13: compile output with –experimental-data-trace

```
Reported 1 errors
error 0:
  The object __config__::A (instantiated at ./main.cf:9) is not complete: attribute n␣
↪(./main.cf:2) is not set
```

In cases where a single error leads to errors for a collection of related attributes, this can greatly simplify the debugging process.

5.12.4 Usage example

Let's have a look at the model below:

Listing 14: service.cf

```
entity Port:
    string host
    number portn
end

index Port(host, portn)

entity Service:
    string name
    string host
    number portn
end

Service.port [0:1] -- Port.service [0:1]


implement Port using std::none
implement Service using bind_port


implementation bind_port for Service:
    self.port = Port(host = self.host, portn = self.portn)
end


sshd = Service(
    name = "opensshd",
    host = "my_host",
    portn = 22,
)


custom_service = Service(
    name = "some_custom_service",
    host = "my_host",
    portn = 22,
)
```

Compiling this with data trace disabled outputs the following error:

Listing 15: compilation output for service.cf with data trace disabled

```
Could not set attribute `port` on instance `__config__::Service (instantiated at ./
↪service.cf:33)` (reported in self.port = Construct(Port) (./service.cf:22))
caused by:
  Could not set attribute `service` on instance `__config__::Port (instantiated at ./
↪service.cf:22,./service.cf:22)` (reported in __config__::Port (instantiated at ./
↪service.cf:22,./service.cf:22) (./service.cf:22))
  caused by:
    value set twice:
    old value: __config__::Service (instantiated at ./service.cf:26)
        set at ./service.cf:22
    new value: __config__::Service (instantiated at ./service.cf:33)
        set at ./service.cf:22
 (reported in self.port = Construct(Port) (./service.cf:22))
```

The error message refers to `service.cf:22` which is part of an implementation. It is not clear which `Service` instance is being refined, which makes finding the cause of the error challenging. Enabling data trace results in the trace below:

Listing 16: data trace for service.cf

```
1  attribute service on __config__::Port instance
2  SUBTREE for __config__::Port instance:
3      CONSTRUCTED BY `Port(host=self.host,portn=self.portn)`
4      AT ./service.cf:22
5      IN IMPLEMENTATION WITH self = __config__::Service instance
6          CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
7          AT ./service.cf:26
8
9      INDEX MATCH: `__config__::Port instance`
10         CONSTRUCTED BY `Port(host=self.host,portn=self.portn)`
11         AT ./service.cf:22
12         IN IMPLEMENTATION WITH self = __config__::Service instance
13             CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',
   ↪portn=22)`
14             AT ./service.cf:33
15 ├── __config__::Service instance
16 │   SET BY `self.port = Port(host=self.host,portn=self.portn)`
17 │   AT ./service.cf:22
18 │   IN IMPLEMENTATION WITH self = __config__::Service instance
19 │       CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
20 │       AT ./service.cf:33
21 │   CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
22 │   AT ./service.cf:33
23 └── __config__::Service instance
24     SET BY `self.port = Port(host=self.host,portn=self.portn)`
25     AT ./service.cf:22
26     IN IMPLEMENTATION WITH self = __config__::Service instance
27         CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
28         AT ./service.cf:26
29     CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
30     AT ./service.cf:26
```

At lines 15 and 23 it shows the two `Service` instances that are also mentioned in the original error message. This time, the dynamic implementation context is mentioned and it's clear that these instances have been assigned in a refinement for the `Service` instances constructed at lines 26 and 33 in the configuration model respectively.

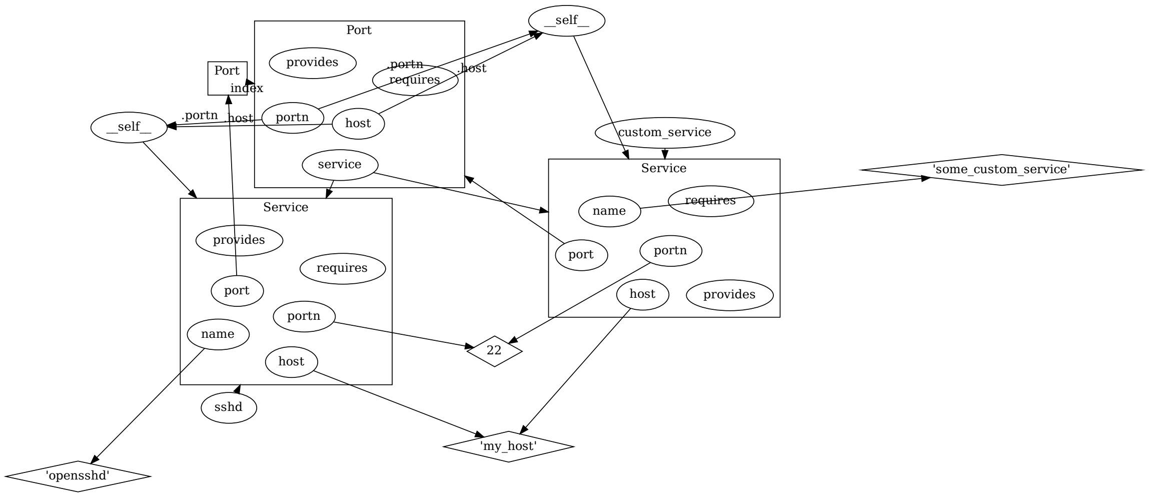
Lines 2–14 in the trace give some additional information about the Port instance. It indicates there is an index match between the Port instances constructed in the implementations for both Service instances. This illustrates the existence of the two branches at lines 15 and 23, and why the assignment in this implementation resulted in the exceeding of the relation arity: the right hand side is the same instance in both cases.

5.12.5 Graphic visualization

Warning: This representation is not as complete as the data trace explained above. It does not show information about statements responsible for each assignment. It was primarily developed as an aid in developing the data flow framework on which the data trace and the root cause analysis tools are built. It's described here because it's closely related to the two tools described above. Its actual use in model debugging might be limited.

Note: Using this feature requires one of inmanta's optional dependencies to be installed: `pip install inmanta[dataflow_graphic]`. It also requires the `fdp` command to be available on your system. This is most likely packaged in your distribution's `graphviz` package.

Let's compile the model in *service.cf* again, this time with `--experimental-dataflow-graphic`. The compile results in an error, as usual, but this time it's accompanied by a graphic visualization of the data flow.



It shows all assignments, as well as the index match between the two Port constructions. An assignment where the right hand side is an attribute `x.y` is shown by an arrow to `x`, labeled with `.y`. Variables are represented by ellipses, values by diamonds and instances by rectangular containers.

5.13 Model Design Guidelines

This section provides design guidelines for experienced developers. It is intended as a way of sharing experience and improving design.

Warning: We provide guidelines here. These are not absolute rules and not all rules are appropriate at all times. Trust your own good judgement before anything else.

5.13.1 Overview

South Bound Integration:

1. Keep close to the API. Keep the structure of the inmanta model as close as possible to the API you model. Refrain from adding abstraction layers when doing pure integration.
2. Prefer modeling relations as relations, avoid reference by string.

5.13.2 Keep close to the API

When doing south bound integrations, it is tempting to *improve* the existing API. Resist this temptation. It leads to the following problems:

1. It costs a lot of effort to integrate the API and redesign it at the same time.
2. Often, you don't understand the API as well as the people who designed it. The improvements you make when starting out often lead to dead ends. Some features that are trivial to represent in the original API become impossible to express in your improved API.
3. APIs evolve. When the API changes in the future, it may become very hard to maintain you improved API.

When you want to offer an improved API, do it in two stages: first model and integrate the existing API, then add an abstraction layer in the model. This neatly separates the integration and abstraction effort.

5.13.3 Prefer modeling relations as relations

Often, APIs have relations. For example, when creating a virtual machine on AWS EC2, it can refer to one or more SecurityGroups. This is modeled in the AWS handler as an explicit relation: `aws::VirtualMachine.security_groups`.

There are different modeling styles possible: 1. Model the relation as a relation between two model entities. (e.g. `aws::VirtualMachine.security_groups`) 2. Model the relation as a (textual) reference. (e.g. `aws::database::RDS.subnet_group`.)

These styles can be mixed within one module.

Explicit relations have the advantage that consistency can be enforced within the model. Type errors and dangling reference are easily prevented. Higher functionality, like correct ordering of the deployment is easy to implement.

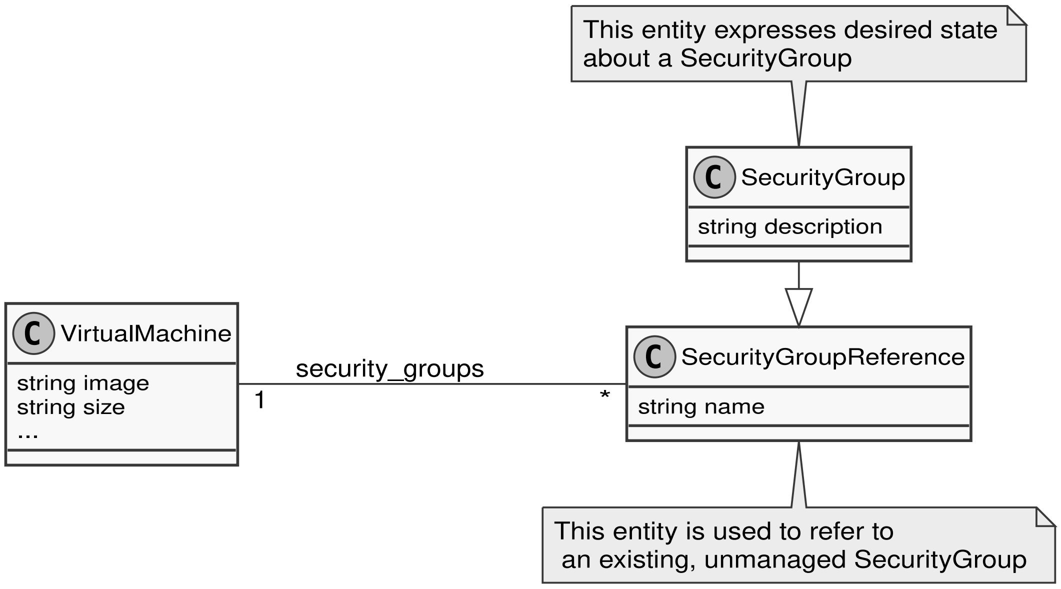
Textual references have the advantage that it is easy to refer to things that are not in the model.

When starting to build up a model, textual reference are attractive, as the modeling effort required is very limited. It is however difficult to migrate away from the textual references later on, because this is a breaking change for any existing model.

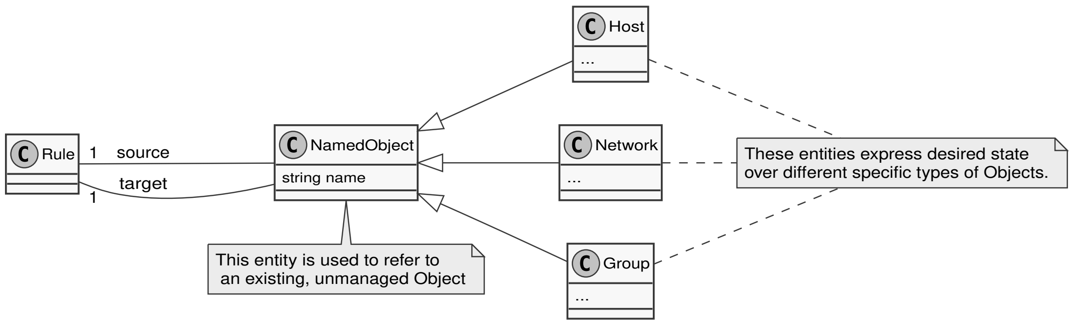
One solution to allow reference to unmanaged entities is to extend `std::ManagedResource`. This allows an entity to exist in the model, but when `managed` is set to `false`, it will never become a resource. However, the entity must still be valid. All attributes and relations still have to be filled in correctly. For entities with many non-optional relations, this is also not the best solution.

Another solution is to introduce a parent entity type that explicitly represents the unmanaged entity. It has only those attributes that are required to correctly refer to it. The concrete, managed entity is a subtype of the unmanaged version. This requires a bit more types, but it is most evolution friendly. No naming convention for the unmanaged parent has been established.

As an example, we could implement `aws::VirtualMachine.security_groups` as follows:


This entity expresses desired state about a SecurityGroup
SecurityGroup
string description
VirtualMachine
string image
string size
...
security_groups
1
*
SecurityGroupReference
string name
This entity is used to refer to an existing, unmanaged SecurityGroup

In cases where there is a single relation that can point to multiple specific subtypes, we can use the existing supertype entity to represent unmanaged entities.


Host
...
Rule
1
source
target
1
NamedObject
string name
Network
...
These entities express desired state over different specific types of Objects.
This entity is used to refer to an existing, unmanaged Object
Group
...

5.14 Partial compiles

Warning: This is an advanced feature, targeted at mature models that have the need to scale beyond their current capabilities. Care should be taken to *implement this safely*, and the user should be aware of *its limitations*.

Note: For partial compiles through LSM, see *its documentation* on how to manage resource sets for an LSM service in addition to the documentation below.

Small updates to large models can be compiled quickly using partial compiles. We merely recompile a tiny, independent portion of the model, as opposed to doing it for the entire model. A `resource set` is made up of the resources in a specific portion of the model.

The model's resources must be separated into resource sets in order to employ partial compilations. The model can then be shrunk to only include the entities for the resource sets that need to be modified. The changes will be pushed to the server when this smaller model is recompiled and exported in partial mode, but all other resource sets won't be impacted.

While the remainder of this document will focus on the straightforward scenario of manually trimming down the model to facilitate quicker compilations, the partial compile feature is actually most useful in conjunction with additional tooling (such as a model generator based on a YAML file) or an Inmanta extension (such as LSM) that offers dynamic entity construction.

5.14.1 Resource sets

Instances of the `std::ResourceSet` entity serve as the model's representation of resource sets. The name of the set and a list of its resources are held by this entity. These `ResourceSet` instances are found by the default exporter to ascertain which resources belong to which set.

In the example below, 1000 networks of 5 hosts each are created. Each host is part of its network's resource set.

Listing 17: main.cf

```
entity Network:
    """
    A network consisting of hosts. Each network is modelled fully independent from␣
↪others.
    """
    int id
end
Network.hosts [0:] -- Host.network [1]

index Network(id)

implementation network_resource_set for Network:
    # The Host resources for a network are all part of the same resource set
    set = std::ResourceSet(name="network-{{ self.id }}")
    for host in self.hosts:
        set.resources += host
    end
end


entity Host extends std::Resource:
    int id
end
index Host(network, id)

implementation host for Host:
    # Resource that doesn't belong to any resource set and is shared
    std::AgentConfig(autostart=true, agentname="host_agent")
end


implement Network using network_resource_set
implement Host using host


# create 1000 networks with 5 hosts each
for i in std::sequence(1000):
    network = Network(id=i)
    for j in std::sequence(5):
        Host(network=network, id=j)
    end
end
```

5.14.2 Partial compiles

When a model is partially compiled, it only includes the entities and resources for the resource sets that need to be changed (as well as their dependencies on additional resources that aren't part of a resource set). It is the server's responsibility to create a new version of the desired state utilizing the resources from the old version and those from the partial compile.

Only the resource sets that are present in the partially compiled model will be replaced when a partial export to the server is performed. Other sets' resources won't be impacted in any way. Shared resources are those that aren't a part of any resource collection and can always be added.

The resources from the prior example would be updated by a partial export for the model below:

Listing 18: main.cf

```
entity Network:
    """
    A network consisting of hosts. Each network is modelled fully independent from␣
↪others.
    """
    int id
end
Network.hosts [0:] -- Host.network [1]

index Network(id)

implementation network_resource_set for Network:
    # The Host resources for a network are all part of the same resource set
    set = std::ResourceSet(name="network-{{ self.id }}")
    for host in self.hosts:
        set.resources += host
    end
end


entity Host extends std::Resource:
    int id
end
index Host(network, id)

implementation host for Host:
    # Resource that doesn't belong to any resource set and is shared
    std::AgentConfig(autostart=true, agentname="host_agent")
end


implement Network using network_resource_set
implement Host using host


# turns out network 0 only needs one host
Host(network=Network(id=0), id=0)
```

As a result, network 0 would be changed to only have one host (the other four resources are removed), but the other networks would continue to function as they had before (because their resource set was not present in the partial export). The comparable complete model would seem as follows:

Listing 19: main.cf

```
entity Network:
    """
    A network consisting of hosts. Each network is modelled fully independent from␣
↪others.
    """
    int id
end
Network.hosts [0:] -- Host.network [1]

index Network(id)

implementation network_resource_set for Network:
    # The Host resources for a network are all part of the same resource set
    set = std::ResourceSet(name="network-{{ self.id }}")
    for host in self.hosts:
        set.resources += host
    end
end


entity Host extends std::Resource:
    int id
end
index Host(network, id)

implementation host for Host:
    # Resource that doesn't belong to any resource set and is shared
    std::AgentConfig(autostart=true, agentname="host_agent")
end


implement Network using network_resource_set
implement Host using host


# create network 0 with only one host
Host(network=Network(id=0), id=0)
# create 999 networks with 5 hosts each
for i in std::sequence(999, start=1):
    network = Network(id=i)
    for j in std::sequence(5):
        Host(network=network, id=j)
    end
end
```

Keep in mind that each resource set contains a collection of independent resources. In this example scenario, since the host instances for other sets do not exist at compilation time, it would be impossible to enforce a host index that was based just on the id and excluded the network.

The model developer is accountable for the following: Each resource set in a partial compilation needs to be separate from and independent of the resource sets that aren't included in the partial model. When performing partial compilations, this is a crucial assumption. If this condition is not satisfied, partial compilations may end up being incompatible with one another (a full compilation with the identical changes would fail), as the index example shows. This can result in undefinable behavior.

Constraints and rules

When using partial compiles, the following rules have to be followed:

- A resource cannot be a part of more than one resource set at once.
- A resource does not have to be part of a resource set.
- Resources cannot be migrated using a partial compile to a different resource set. A full compile is necessary for this process.
- A resource set that is contained in a partial export must be complete, meaning that all of its resources must be present.
- Resources that weren't assigned to a specific resource set can never be updated or removed by a partial build. Although, adding resources is allowed.
- Resources within a resource set cannot depend on resources in another resource set. Dependencies on shared resources are allowed.
- Multiple resource sets may be updated simultaneously via a partial build.

For a guide on how to design a model in order to take these into account, see *Modeling guidelines*.

Exporting a partial model to the server

Three arguments can be passed to the `inmanta export` command in order to export a partial model to the server:

- `--partial` To specify that the model being compiled only contains the resources that need to be updated in relation to the previous version of the model.
- `--delete-resource-set <resource-set-name>` This option, which may be used more than once, instructs the model to remove the resource set with the specified name. Only in conjunction with the preceding choice may this option be utilized. Note that utilizing a `std::ResourceSet` that includes no resources allows resource sets to be implicitly deleted during a partial compilation.
- `--soft-delete` To silently ignore deletion of resource sets specified through the `--delete-resource-set` option if the model is exporting resources that are part of these sets.

Limitations

- **The compiler cannot verify all constraints that would be verified when a full build is run. Some index constraints, for instance, cannot be verified. The model creator is in charge of making sure that these constraints are met.**

  See *Modeling guidelines* on how to design your model.
- If just a partial compile is performed, it is possible for a shared resource to become obsolete. The shared resource will become obsolete when a partial compile deletes the last resource that depended on it, but it is preserved as a server-managed resource because partial compiles cannot delete shared resources. A full compile is required to remove shared resources. Scheduled full compilations that `garbage-collect` these shared resources are one way to fix this. The *`auto_full_compile`* environment setting is used to schedule full compilations. As an example, to plan a daily full compile for 01:00 UTC, use the `auto_full_compile` environment setting: `0 1 * * *`.

5.14.3 Modeling guidelines

This section will introduce some guidelines for developing models for use with the partial compilation feature. Take extreme care when not following these guidelines and keep in mind the *Constraints and rules*. The purpose of these guidelines is to present a modelling approach to safely make use of partial compiles. In essence, this boils down to developing the model so that a partial compile only succeeds if a full one would as well.

In this guide, we only cover models where each set of independent resources is defined by a single top-level entity, which we will refer to as the "service" or "service entity" (as in `LSM`). We will use the term "identity" to refer to any set of attributes that uniquely identify an instance. In the model this usually corresponds to an index.

All potential instances of a service entity must be refined to compatible (low level) configuration when creating an Inmanta model. In the model this config is represented by the resources. Therefore these guidelines will focus on creating valid and compatible resources. With well-designed resources, valid and compatible config will follow.

To safely make use of partial compiles, each service must be the sole owner of its resources and any shared resources must be identical across service instances. The graph below pictures a valid service for partial compiles. Each arrow represents a refinement: one entity creating another in one of its implementations. The valid service results in fully separate resource sets for each instance. Additionally, the one shared resource is created consistently between service instances. For each entity type, the `id` attribute is assumed to be an identifying attribute for the instance (i.e. there is an index on the attribute).

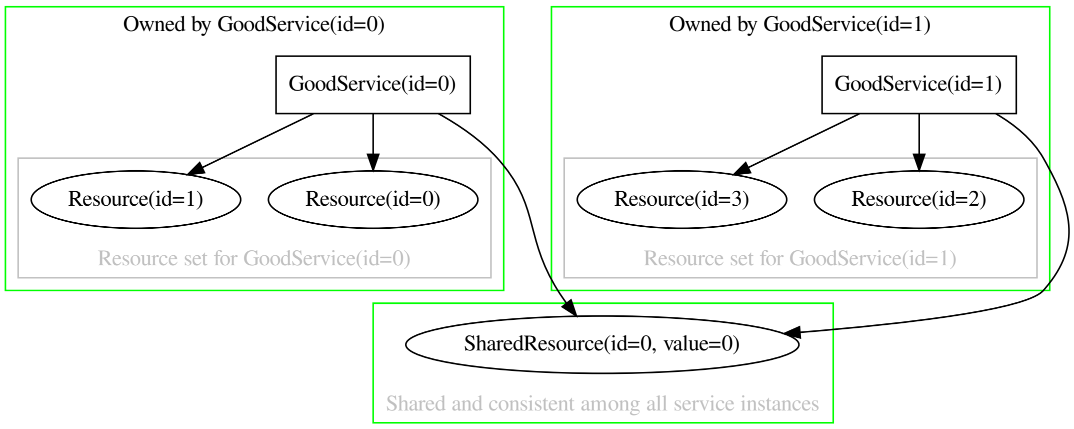

Owned by GoodService(id=0)
GoodService(id=0)
Resource(id=1)
Resource(id=0)
Resource set for GoodService(id=0)
Owned by GoodService(id=1)
GoodService(id=1)
Resource(id=3)
Resource(id=2)
Resource set for GoodService(id=1)
SharedResource(id=0, value=0)
Shared and consistent among all service instances

Fig. 1: A good service for partial compiles.

In contrast, the graph below shows an invalid service definition. Its resources overlap between instances. The invalid service can thus not be allowed for partial compiles because no resource can be considered completely owned by a single service instance.

Finally, the graph below shows another invalid model. Here, the resources are clearly divided into sets, but the shared resource is created inconsistently: one instance sets its value to 0 while the other sets it to 1.

In conclusion, each service's refinements (through implementations) form a tree that may only intersect between service instances on shared nodes. The whole subtree below such a shared node should be considered shared and any resources in it must not be part of a resource set. All shared resources should be consistent between any two service instances that might create the object (see *Constraints and rules*). All other nodes should generally be considered owned by the service and all their resources be part of the service's resource set. For more details on what it means to own a resource (or any child node in the tree) and how to ensure two service instance's trees can not intersect on owned nodes, see the *Ownership* subsection.

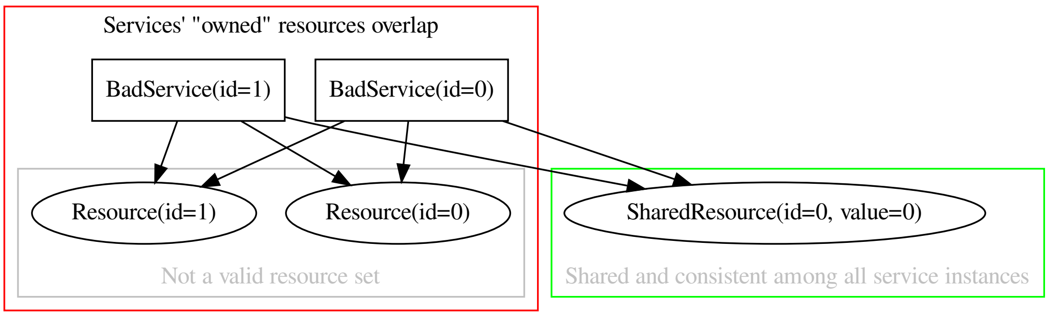

Services' "owned" resources overlap
BadService(id=1)
BadService(id=0)
Resource(id=1)
Resource(id=0)
Not a valid resource set
SharedResource(id=0, value=0)
Shared and consistent among all service instances

Fig. 2: A bad service for partial compiles: no owned resources

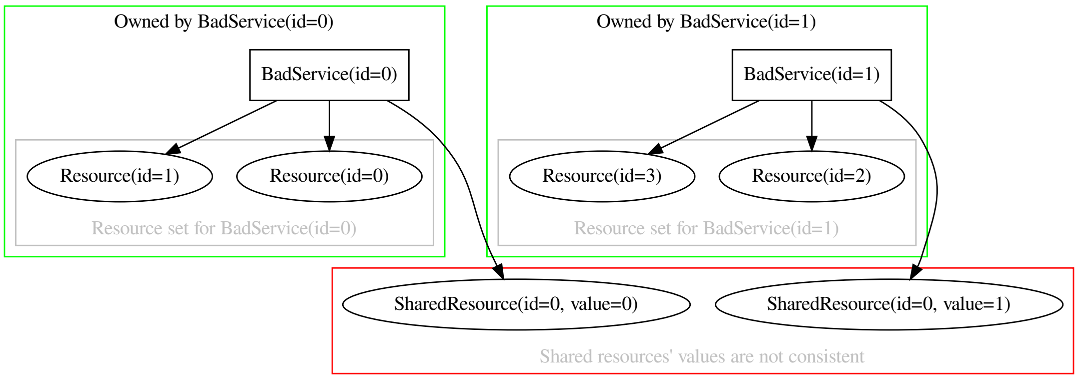

Owned by BadService(id=0)
BadService(id=0)
Resource(id=1)
Resource(id=0)
Resource set for BadService(id=0)
Owned by BadService(id=1)
BadService(id=1)
Resource(id=3)
Resource(id=2)
Resource set for BadService(id=1)
SharedResource(id=0, value=0)
SharedResource(id=0, value=1)
Shared resources' values are not consistent

Fig. 3: A bad service for partial compiles: conflicting shared resources

Service instance uniqueness

With full compiles, indexes serve as the identity of a service instance in the model. The compiler then validates that no conflicting service instances exist. With partial compiles this validation is lost because only one service instance will be present in the model. However, it is still crucial that such conflicts do not exist. Put simply, we need to make sure that a partial compile succeeds only when a full compile would succeed as well. This subsection deals solely with the uniqueness of service instances. The *Ownership* subsection then deals with safe refinements into resources.

To ensure service instance definitions are distinct, the model must make sure to do appropriate validation on the full set of definitions. When doing a partial compile, the model must verify that the service instance it is compiling for has a different identity from any of the previously defined service instances. This can be achieved by externally checking against some sort of inventory that there are no matches for any set of input attributes that identify the instance.

The current implementation of partial compiles does not provide any helpers for this verification. It is the responsibility of the model developer or the tool/extension that does the export to ensure that no two service instances can be created that are considered to have the same identity by the model.

For example, suppose we modify the example model to take input from a simple yaml file:

```
for network_def in mymodule::read_from_yaml():
    network = Network(id=network_def["id"])
    for host in network["hosts"]:
        network.hosts += Host(id=host["id"])
    end
end
```

```
networks:
    - id: 0
      hosts:
        - id: 0
    - id: 1
      hosts:
        - id: 0
        - id: 1
        - id: 2
        - id: 3
        - id: 4
    - id: 0
      hosts:
        - id: 0
        - id: 1
```

The `read_from_yaml()` plugin would have to verify that no two networks with the same id are defined. After this validation, if doing a partial, it may return a list with only the relevant network in it. For the yaml given above validation would fail because two networks with the same id are defined.

Ownership

A resource can safely be considered owned by a service instance if it could never be created by another service instance. There are two main mechanisms that can be used to provide this guarantee, both of which will be described in their own subsection below. One is the use of indexes on appropriate locations, the other is the use of some external allocator of unique values (e.g. a plugin to generate a UUID or to allocate values in an inventory).

In either case, the goal is to make sure that any object that is marked as owned by a service instance, is unique to that instance. In the index case we do so by making sure the object's identity is in fact completely and uniquely derived from the identity of the service instance. In the case where unique values are externally produced/allocated, responsibility for uniqueness falls to the plugin that produces the values.

Ownership through indexes

As stated above, during partial compiles indexes alone can not serve as a uniqueness guarantee because each compile only contains a single service instance. And yet, indexes can still be used as a mechanism to guarantee ownership: e.g. if a value for a resource's index is uniquely derived from the identity of its service instance, this in itself is a guarantee that no other service instance could result in this same resource. In other words, rather than count on the stand-alone identity aspect of the index, we will make sure the identity is fully defined by the service instance's identity (or an external inventory). This, coupled with the *Service instance uniqueness* guarantee ensures that the refinement trees will not intersect. This in turn allows us to conclude that the partial compile behavior will be the same as the full compile behavior.

Generally, for every index on a set of attributes of an owned resource, at least one of the fields must be either derived from the identity of the service instance, or allocated in a safe manner by a plugin as described above. The same goes for every pair of resource id and agent. If the first constraint is not met, a full compile might fail, while if the second is not met, the export will be rejected because two services are trying to configure the same resources.

For example, consider the example model from before. If two networks with two hosts each would be created, they would result in two disjunct resource sets, as pictured below.

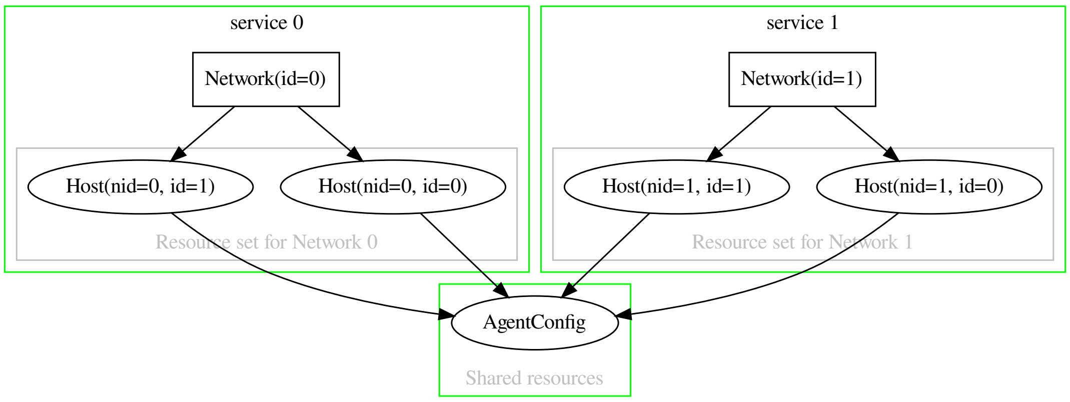


Fig. 4: Two valid service instances with their resource sets

Now suppose the index on `Host` did not include the network instance. In that case the identity of a `Host` instance would no longer be derived from the identity of its `Network` instance. It would then be possible to end up with two networks that refine to the same host objects as shown below. The resource sets are clearly no longer disjunct.

Ownership through allocation

Instead of the index `Host(network, id)` we could also use an allocation plugin to determine the id of a host. Suppose we add such a plugin that allocates a unique value in some external inventory, then the index is no longer required for correct behavior because the allocator guarantees uniqueness for the host id:

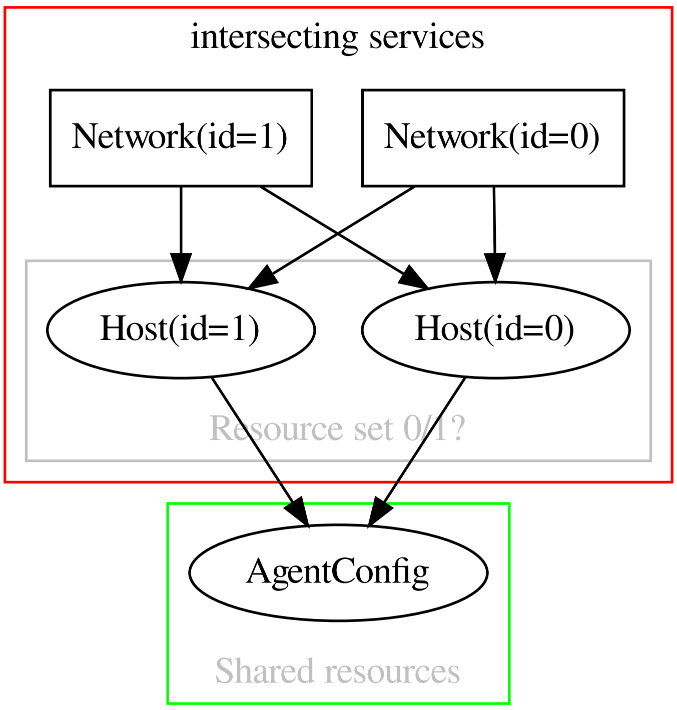


Fig. 5: Two invalid service instances with a resource set conflict

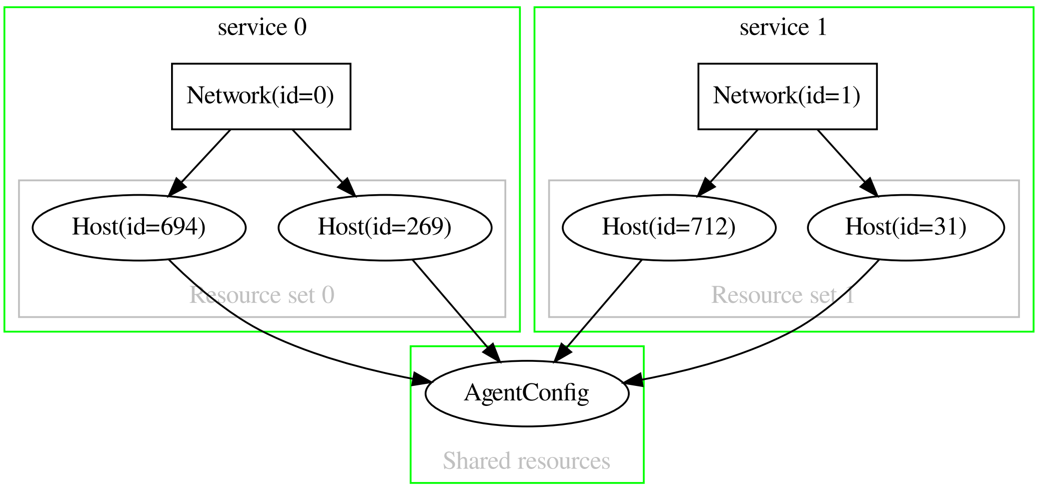


Fig. 6: Two valid services with their resource sets, using allocation

Inter-resource set dependencies

Resources within a resource set can only depend on resources within the same resource set or on shared resources. Shared resources on the other hand can have dependencies on any resource in the model. The diagram below provides an example where the resource dependency graph satisfies these requirements. The arrows in the diagram show the requires relationship between entities/resources.

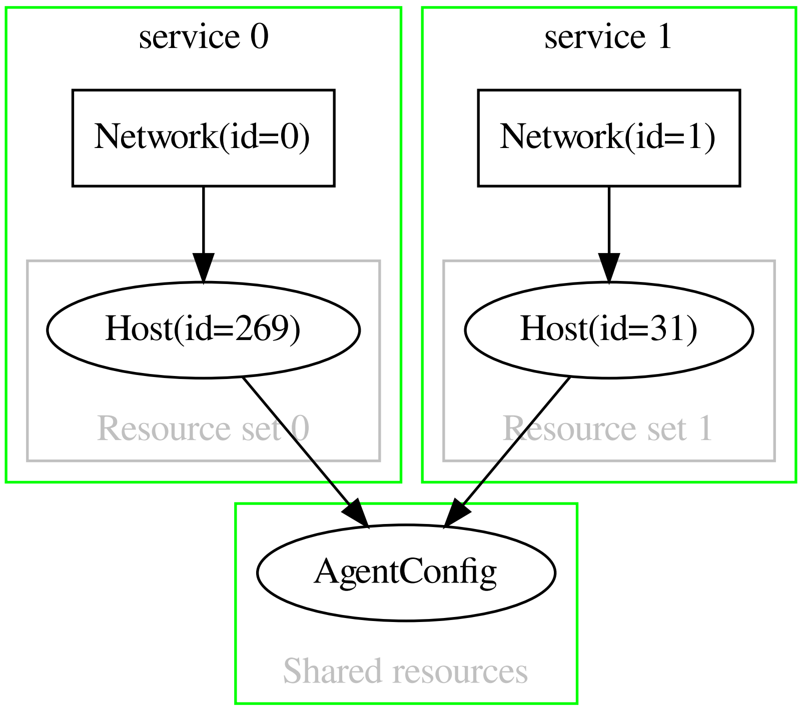


Fig. 7: Two resource sets satisfying the dependency constraints

In the diagram below, resource `Host(id=269)` that belongs to resource set 0 depends on resource `Host(id=31)` that belongs to resource set 1. This inter-resource set dependency is not allowed.

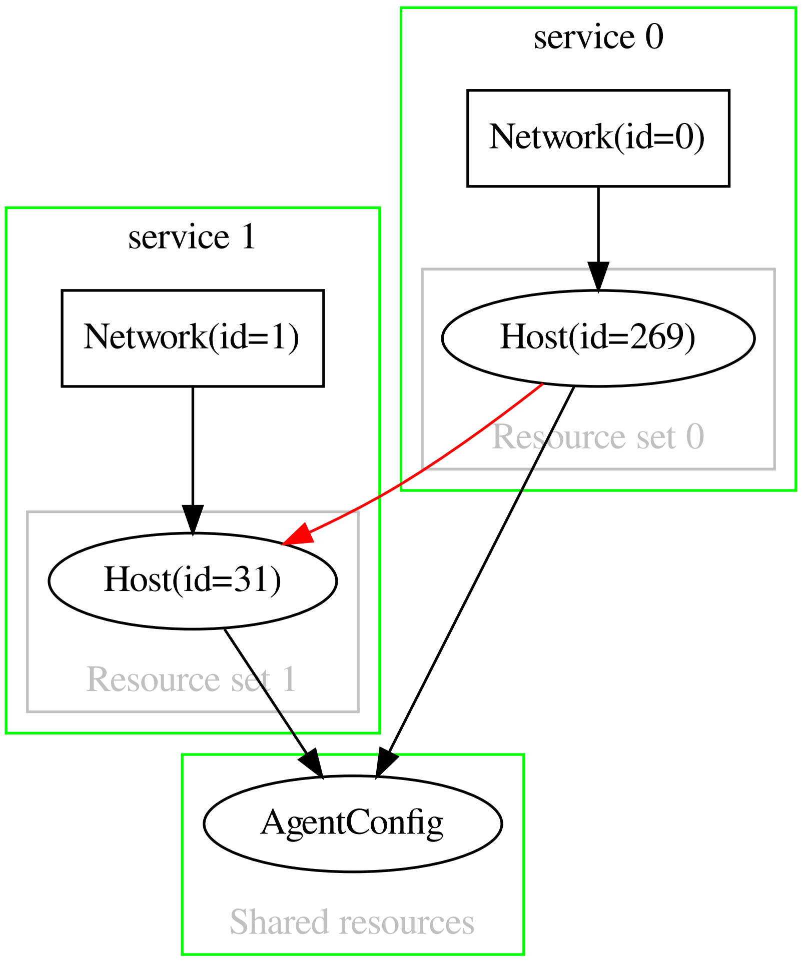


Fig. 8: Two resource sets violating the dependency constraints

Testing

While the guidelines outlined above suffice for safe use of partial compiles, a modeling error is easily made. In addition to the usual testing of behavior of both full and partial compiles, you should include tests that guard against incompatible resource sets and/or shared resources. These tests would generally be full compile tests with multiple service instances. As long as a full compile succeeds for any valid set of inputs, you can be confident the partial compile will behave the same. If on the other hand a set of valid service instances exist for which the full compile fails, you most likely have a modeling error that would allow sequential partial compiles for those same instances.

5.15 Unmanaged Resources

Unmanaged resources are resources that live in the network that are not yet managed by the orchestrator. They may be of the same type as other resources that are already managed, or something else entirely. The orchestrator can discover unmanaged resources in the network, given proper guidance. This discovery is driven by discovery resources in the model. They express the intent to discover resources of the associated type in the network.

5.15.1 Terminology

- **Discovery resource:** The category of resources that express the intent to discover resources in the network. This is an actual resource that is part of the configuration model.
- **Discovered resource:** This is the unit of data that is generated by the discovery process. Each time the discovery process discovers a resource, it creates a record in the inventory for discovered resources. This record contains the set of attributes that define the current state of the discovered resource. Discovered resources only exist in the discovered resources database. They don't exist in the configuration model.

5.15.2 Example

The code snippet below defines a discovery resource called `InterfaceDiscovery`. Instances of this resource will discover the interfaces present on a specific host. A discovery resource must always inherit from `std::DiscoveryResource`. Note that discovery resources are defined in exactly the same way as a regular resource, except that they inherit from `std::DiscoveryResource` instead of `std::PurgeableResource` or `std::Resource`.

Listing 20: my_project/main.cf

```
import ip

entity InterfaceDiscovery extends std::DiscoveryResource:
    """
    A discovery resource that discovers interfaces on a specific host.

    :attr name_filter: If not null, only discover interfaces for which the name␣
↪matches this regular expression.
                       Otherwise discover all the interfaces on the host.
    """
    string? name_filter = null
end

InterfaceDiscovery.host [1] -- ip::Host

index InterfaceDiscovery(host)

implement InterfaceDiscovery using parents, std::none
```

The associated handler code is shown below:

Listing 21: my_module/inmanta_plugins/__init__.py

```python
import re
from collections import abc

import pydantic

from inmanta import resources
from inmanta.data.model import ResourceIdStr
from inmanta.agent.handler import provider, DiscoveryHandler, HandlerContext
from inmanta.resources import resource, DiscoveryResource


@resource("my_module::InterfaceDiscovery", agent="host.name", id_attribute="host")
class InterfaceDiscovery(DiscoveryResource):
    fields = ("host", "name_filter")

    host: str
    name_filter: str

    @staticmethod
    def get_host(exporter, resource):
        return resource.host.name


class UnmanagedInterface(pydantic.BaseModel):
    """
    Datastructure used by the InterfaceDiscoveryHandler to return the attributes
    of its discovered resources.
    """

    host: str
    interface_name: str
    ip_address: str


@provider("my_module::InterfaceDiscovery", name="interface_discovery_handler")
class InterfaceDiscoveryHandler(DiscoveryHandler[InterfaceDiscovery, UnmanagedInterface]):
    def discover_resources(
        self, ctx: HandlerContext, discovery_resource: InterfaceDiscovery
    ) -> dict[ResourceIdStr, UnmanagedInterface]:
        """
        Entrypoint that is called by the agent when the discovery resource is deployed.
        """
        discovered: abc.Iterator[UnmanagedInterface] = (
            UnmanagedInterface(**attributes)
            for attributes in self._get_discovered_interfaces(discovery_resource)
            if discovery_resource.name_filter is None or re.match(discovery_resource.name_filter, attributes["interface_name"])
        )
        return {
            resources.Id(
                entity_type="my_module::Interface",
                agent_name=res.host,
                attribute="interface_name",
```

(continues on next page)

(continued from previous page)

```
53                 attribute_value=res.interface_name,
54             ).resource_str(): res
55             for res in discovered
56         }
57 
58     def _get_discovered_interfaces(self, discovery_resource: InterfaceDiscovery) -> list[dict[str, object]]:
59         """
60         A helper method that contains the logic to discover the unmanaged interfaces in the network.
61         It returns a list of dictionaries where each dictionary contains the attributes of an unmanaged resource.
62         """
63         raise NotImplementedError()
```

The handler code consists of three parts:

- Lines 12-21: The class that describes how the discovery resource `InterfaceDiscovery` should be serialized. This resource definition is analogous to the definition of a regular `PurgeableResource` or `Resource`, except that the class inherits from `DiscoveryResource`.
- Lines 24-31: A Pydantic BaseModel that represents the datastructure that will be used by the discovery handler to return the attributes of the discovered resources. This specific example uses a Pydantic BaseModel, but discovery handlers can use any json serializable datastructure.
- Line: 34-61: This is the handler for the discovery resource. A discovery handler class must satisfy the following requirements:
  - It must be annotated with the `@provider` annotation, like a regular `CRUDHandler` or `ResourceHandler`.
  - It must inherit from the `DiscoveryHandler` class. This is a generic class with two parameters. The first parameter is the class of the associated `DiscoveryResource` and the second parameter is the type of datastructure that the discovery handler will use to return the attributes of discovered resources.
  - It must implement a method called `discover_resources` that contains the logic to discover the resources in the network. This method returns a dictionary. The keys of this dictionary contain the resource ids of the discovered resources and the values the associated attributes.

5.15.3 Sharing attributes

In some situations there is a need to share behavior or attributes between a resource X and the discovery resource for X. For example, both might require credentials to authenticate to their remote host. This can be done by making both entities inherit from a shared parent entity. An example is provided below.

Listing 22: my_project/main.cf

```
import ip

entity Credentials:
    """
    An entity that holds the shared attributes between the Interface and InterfaceDiscovery entity.
    """
    string username
    string password
end

implement Credentials using std::none
```

(continues on next page)

(continued from previous page)

```

entity InterfaceBase:
    """
    Base entity for the Interface and InterfaceDiscovery handler.
    """
end

InterfaceBase.credentials [1] -- Credentials
InterfaceBase.host [1] -- ip::Host

implement InterfaceBase using std::none

entity Interface extends InterfaceBase, std::PurgeableResource:
    """
    An entity that represents an interface that is managed by the Inmanta server.
    """
    string name
    std::ipv4_address ip_address
end

index Interface(host, name)

implement Interface using parents, std::none

entity InterfaceDiscovery extends InterfaceBase, std::DiscoveryResource:
    """
    A discovery resource used to discover interfaces that exist on a specific host.

    :attr name_filter: If not null, this resource only discovers the interfaces for␣
↪which the name matches this
                       regular expression. Otherwise discover all the interfaces on␣
↪the host.
    """
    string? name_filter = null
end

index InterfaceDiscovery(host)

implement InterfaceDiscovery using parents, std::none
```

The `Credentials` entity, in the above-mentioned snippet, contains the shared attributes between the PurgeableResource `Interface` and the DiscoveryResource `InterfaceDiscovery`.

The associated handler code is provided below:

Listing 23: my_module/inmanta_plugins/__init__.py

```
import re
from collections import abc
from typing import Optional

import pydantic

from inmanta import resources
from inmanta.data.model import ResourceIdStr
from inmanta.agent.handler import provider, DiscoveryHandler, HandlerContext,␣
↪CRUDHandler
```

(continues on next page)

(continued from previous page)

```python
from inmanta.resources import resource, DiscoveryResource, PurgeableResource


class InterfaceBase:
    fields = ("host", "username", "password")

    host: str
    username: str
    password: str

    @staticmethod
    def get_host(exporter, resource):
        return resource.host.name

    @staticmethod
    def get_username(exporter, resource):
        return resource.credentials.username

    @staticmethod
    def get_password(exporter, resource):
        return resource.credentials.password


@resource("my_module::Interface", agent="host.name", id_attribute="name")
class Interface(InterfaceBase, PurgeableResource):
    fields = ("name", "ip_address")

    name: str
    ip_address: str


@resource("my_module::InterfaceDiscovery", agent="host.name", id_attribute="host")
class InterfaceDiscovery(InterfaceBase, DiscoveryResource):
    fields = ("name_filter",)

    name_filter: Optional[str]


class UnmanagedInterface(pydantic.BaseModel):
    """
    Datastructure used by the InterfaceDiscoveryHandler to return the attributes
    of the discovered resources.
    """

    host: str
    interface_name: str
    ip_address: str


class Authenticator:
    """
    Helper class that handles the authentication to the remote host.
    """

    def login(self, credentials: InterfaceBase) -> None:
        raise NotImplementedError()
```

(continues on next page)

(continued from previous page)

```

    def logout(self, credentials: InterfaceBase) -> None:
        raise NotImplementedError()


@provider("my_module::Interface", name="interface_handler")
class InterfaceHandler(Authenticator, CRUDHandler[Interface]):
    """
    Handler for the interfaces managed by the orchestrator.
    """

    def pre(self, ctx: HandlerContext, resource: Interface) -> None:
        self.login(resource)

    def post(self, ctx: HandlerContext, resource: Interface) -> None:
        self.logout(resource)

    def read_resource(self, ctx: HandlerContext, resource: Interface) -> None:
        raise NotImplementedError()

    def create_resource(self, ctx: HandlerContext, resource: Interface) -> None:
        raise NotImplementedError()

    def delete_resource(self, ctx: HandlerContext, resource: Interface) -> None:
        raise NotImplementedError()

    def update_resource(self, ctx: HandlerContext, changes: dict, resource:␣
↪Interface) -> None:
        raise NotImplementedError()


@provider("my_module::InterfaceDiscovery", name="interface_discovery_handler")
class InterfaceDiscoveryHandler(Authenticator, DiscoveryHandler[InterfaceDiscovery,␣
↪UnmanagedInterface]):

    def pre(self, ctx: HandlerContext, resource: InterfaceDiscovery) -> None:
        self.login(resource)

    def post(self, ctx: HandlerContext, resource: InterfaceDiscovery) -> None:
        self.logout(resource)

    def discover_resources(
        self, ctx: HandlerContext, discovery_resource: InterfaceDiscovery
    ) -> abc.Mapping[ResourceIdStr, UnmanagedInterface]:
        """
        Entrypoint that is called by the agent when the discovery resource is␣
↪deployed.
        """
        discovered: abc.Iterator[UnmanagedInterface] = (
            UnmanagedInterface(**attributes)
            for attributes in self._get_discovered_interfaces(discovery_resource)
            if discovery_resource.name_filter is None or re.match(discovery_resource.
↪name_filter, attributes["interface_name"])
        )
        return {
            resources.Id(
```

(continues on next page)

(continued from previous page)

```
                entity_type="my_module::Interface",
                agent_name=res.host,
                attribute="interface_name",
                attribute_value=res.interface_name,
            ).resource_str(): res
            for res in discovered
        }

    def _get_discovered_interfaces(self, discovery_resource: InterfaceDiscovery) ->␣
↪list[dict[str, object]]:
        """
        A helper method that contains the logic to discover the unmanaged interfaces␣
↪in the network.
        It returns a list of dictionaries where each dictionary contains the␣
↪attributes of a discovered interface.
        """
        raise NotImplementedError()
```

In the above-mentioned code snippet the `Credentials` class contains the shared attributes between the `Interface` resource and the `InterfaceDiscovery` resource. The `Authenticator` class on the other hand contains the shared logic between the `InterfaceHandler` and the `InterfaceDiscoveryHandler` class.

5.16 Dict Path

DictPath is a library for navigating json data.

The DictPath library offers a convenient way to get a specific value out of a structure of nested dicts and lists.

5.16.1 Writing DictPath expressions

A DictPath expression is a `.`-separated path. The following elements are supported:

1. `.dkey`: Return the value under `dkey` in the dict. `dkey` cannot be an empty string. Use the * character to get all values of the dictionary.
2. **`lst[lkey=lvalue]`: Find a dictionary in a list of dictionaries. Find the dict with `lkey=lvalue`. `lvalue` can be an empty string. `lst` and `lkey` cannot be an empty string. If no or more than one dict matches the filter, a LookupError is raised. The * character can be used for `lkey` and `lvalue` to match respectively any key or value. `\0` can be used for `lvalue` to match against the value``None``.**
   If no single key uniquely identifies an object, multiple keys can be used: `lst[lkey1=lvalue1][lkey2=lvalue2]`.

Each element of the path (keys or values) must escape the following special characters with a single backslash: `\`, `[`, `]`, `.`, * and =. Other characters must not be escaped.

A leading `.` character represent the entire data structure provided to the dict path library. As such, the following dict paths are logically equivalent to each other: `a.b.c` and `.a.b.c`. A dict path can also consist of a single dot (`.`). This expression represents the identity function.

5.16.2 Using DictPath in code

> **Warning:** The dict path library only works correctly when the keys and values, referenced in a dict path expression, are of a primitive type and the type is the same for all keys and values at the same level. For example, `{"True": 1, True: 2}` is not a valid dictionary.

- To convert a dictpath expression to a `DictPath` instance, use `dict_path.to_path`. Use `dict_path.to_wild_path` in order to allow wildcards (`*`) to be used in the dict path expression.
- To get the element from a collection use `DictPath.get_element(collection)`
- To set an element in a collection use `DictPath.set_element(collection, value)`

`class inmanta.util.dict_path.`**`DictPath`**

A base class for all non-wild dict paths segments. The key difference between WildDictPath and DictPath subclasses are:

1. WildDictPath can only get a list of elements, with get_elements. If no element is found, an empty list is returned, no error is raised.
2. DictPath can not use get_elements as it is always expected to have exactly one match.
3. DictPath can use get_element, which will return the matching element, or raise an exception if more or less than one is found.
4. DictPath can set values, using set_element, and can build the dict structure expected by the path by using the construct flag in the get_element method.

`abstract` **`get_element`**(*container: object*, *construct: bool = False*) → object

Get the element identified by this Path from the given collection

Parameters

- **`container`** – the container to search in
- **`construct`** – construct a dict on the location identified by this path in the container if the element doesn't exist. Return this new dict.

Raises

`KeyError` – if the element is not found or if more than one occurrence was found.

`get_elements`(*container: object*) → list[object]

Get the elements identified by this Path from the given collection. If no element is matched, an empty list is returned.

Parameters

`container` – the container to search in

`abstract` **`get_key`**() → str

Return the dictionary key referenced by this element in the dict path.

`get_path_sections`() → Sequence[*DictPath*]

Get the individual parts of this path

`abstract` **`remove`**(*container: object*) → None

Remove an element if it exists:

- On an InDict or a WildInDict: Remove the referenced key from the dictionary.
- On a KeyedList or a WildKeyedList: Remove the referenced element from the list.
- On a NullPath: This operation is not supported on a NullPath.

abstract set_element(*container: object*, *value: object*, *construct: bool = True*) → None

Set the element identified by this Path from the given collection.

If construct is True, all containers on the path towards the value are constructed if absent.

Raises
LookupError – if the path leading to the element is not found or if more than one occurrence was found.

inmanta.util.dict_path.**to_path**(*inp: str*) → *DictPath*

Convert a string to a DictPath

Raises
InvalidPathException – the path is not valid

inmanta.util.dict_path.**to_wild_path**(*inp: str*) → WildDictPath

Convert a string to a WildDictPath

Raises
InvalidPathException – the path is not valid

5.16.3 Example

```
from inmanta.util import dict_path

container = {
    "a": "b",
    "c": {
        "e": "f"
    },
    "g": [
        {"h": "i", "j": "k"},
        {"h": "a", "j": "b"}
    ]
}

assert dict_path.to_path("a").get_element(container) == "b"
assert dict_path.to_path("c.e").get_element(container) == "f"
assert dict_path.to_path("g[h=i]").get_element(container) == {"h": "i", "j": "k"}

assert dict_path.to_wild_path("c.*").get_elements(container) == ["f"]
assert sorted(dict_path.to_wild_path("g[h=i].*").get_elements(container)) == ["i", "k
↪"]
assert dict_path.to_wild_path("g[*=k]").get_elements(container) == [{"h": "i", "j": "k
↪"}]

dict_path.to_path("g[h=b].i").set_element(container, "z")
assert dict_path.to_path("g[h=b]").get_element(container) == {"h": "b", "i": "z"}
assert dict_path.to_path("g[h=b].i").get_element(container) == "z"
```

CHAPTER

SIX

PLATFORM DEVELOPER DOCUMENTATION

6.1 Creating a new server extension

Inmanta server extensions are separate Python packages with their own release cycle that can add additional server slices and Inmanta environment settings to the orchestrator. Server slices are components in the service orchestrator. A slice can be responsible for API endpoints or provide internal services to other slices. The core server extension provides all slices of the core service orchestrator.

6.1.1 The package layout of a server extension

Each Inmanta server extension is defined as a subpackage of the `inmanta_ext` package. `inmanta_ext` is a namespace package used by the service orchestrator to discover new extensions. The following directory structure is required for a new extension called `new_extension`.

```
inmanta_ext
|
|__ new_extension
|     |__ __init__.py
|     |__ extension.py
```

- The `__init__.py` file can be left empty. This file is only required to indicate that `new_extension` is a python package.
- The `extension.py` file must contain a `setup` function that registers the necessary server slices to the application context. An example `extension.py` file is shown below. The parameter `<server-slice-instance>` should be replaced with an instance of the server slice that belongs to the extension. Multiple server slices can be registered.
- The `extension.py` file can contain an optional `register_environment_settings` function that allows an extension to register extension-specific settings that can be used to customize an Inmanta environment.

```python
# File: extension.py
from inmanta.server.extensions import ApplicationContext
from inmanta import data

def setup(application: ApplicationContext) -> None:
    application.register_slice(<server-slice-instance>)

def register_environment_settings(application: ApplicationContext) -> None:
    application.register_environment_setting(
        data.Setting(
            name="my_environment_setting",
            default=False,
            typ="bool",
            validator=data.convert_boolean,
```

(continues on next page)

(continued from previous page)

```
            doc="Explain what the setting does.",
        )
    )
```

Tip: Indicate which version of the Inmanta core is compatible with the developed extension by constraining the version of the Inmanta core package to a valid range in the `setup.py` file of the extension.

6.1.2 Adding server slices to the extension

A server slice is defined by creating a class that extends from *inmanta.server.protocol.ServerSlice*.

class inmanta.server.protocol.**ServerSlice**(*name: str*)

Base class for server extensions offering zero or more api endpoints

Extensions developers should override the lifecycle methods:

- *ServerSlice.prestart()*
- *ServerSlice.start()*
- *ServerSlice.prestop()*
- *ServerSlice.stop()*
- *ServerSlice.get_dependencies()*

To register endpoints that server static content, either use :func:'add_static_handler' or :func:'add_static_content' To create endpoints, use the annotation based mechanism

To schedule recurring tasks, use `schedule()` or *self._sched* To schedule background tasks, use `add_background_task()`

get_depended_by() → list[str]

List of names of slices that must be started after this one.

get_dependencies() → list[str]

List of names of slices that must be started before this one.

async prestart(*server: Server*) → None

Called by the RestServer host prior to start, can be used to collect references to other server slices Dependencies are not up yet.

async prestop() → None

Always called before stop

Stop producing new work: - stop timers - stop listeners - notify shutdown to systems depending on us (like agents)

sets is_stopping to true

But remain functional

All dependencies are up (if present)

async start() → None

Start the server slice.

This method *blocks* until the slice is ready to receive calls

Dependencies are up (if present) prior to invocation of this call `async stop()` → None

Go down

All dependencies are up (if present)

This method *blocks* until the slice is down

- The constructor of the ServerSlice class expects the name of the slice as an argument. This name should have the format `"<extension-name>.<server-slice-name>"`. `<extension-name>` is the name of the package that contains the `extension.py` file. `<server-slice-name>` can be chosen by the developer.
- The `prestart()`, `start()`, `prestop()`, `stop()`, `get_dependencies()` and `get_depended_by()` methods can be overridden when required.

6.1.3 Enable the extension

By default, no extensions are enabled on the Inmanta server. Extensions can be enabled by specifying them in the *`server.enabled-extensions`* option of the Inmanta configuration file. This option accepts a comma-separated list of extensions that should be enabled.

```
# File: /etc/inmanta/inmanta.d/0-extensions.cfg
[server]
enabled_extensions=new_extension
```

6.1.4 The Inmanta extension template

A new Inmanta extension can be created via the Inmanta extension template. This is a cookiecutter template to generate the initial Python project for a new Inmanta extension. The documentation regarding this template is available on https://github.com/inmanta/inmanta-extension-template.

6.2 Database Schema Management

In some situation, a change to the database schema is required. To perform these database schema migrations, we implemented a migration tool and associated testing framework. This page describes how to create a new version of the database schema and test the migration script.

6.2.1 New schema version definition

The version number of the database schema evolves independently from any other versioned Inmanta element (product version, extension version, etc.). Each commit can introduce changes to the database schema. When that happens the commit creates a new database schema version. This means that multiple schema version can exist between two consecutive stable releases of the orchestrator.

A new version of the database schema is defined by adding a new Python module to the `inmanta.db.versions` package. The name of this module should have the format `v<timestamp><i>.py`, where the timestamp is in the form `YYYYMMDD` and `i` is an index to allow more than one schema update per day (e.g. `v202102220.py`).

Each of these Python modules should implement an asynchronous function `update` that accepts a database connection object as an argument. This function should execute all database queries required to update from the previous version of the database schema to the new version of the database schema.

An example is given in the code snippet below:

```
# File: src/inmanta/db/versions/v202102220.py
from asyncpg import Connection

```

(continues on next page)

(continued from previous page)

```python
async def update(connection: Connection) -> None:
    schema = """
    ALTER TABLE public.test
    ADD COLUMN new_column;
    """
    await connection.execute(schema)
```

6.2.2 Executing schema updates

Schema updates are applied automatically when the Inmanta server starts. The following algorithm is used to apply schema updates:

1. Retrieve the current version of the database schema from the `public.schemamanager` table of the database.
2. Check if the `inmanta.db.versions` package contains any schema updates.
3. When schema updates are available, each `update` function between the current version and the latest version is executed in the right order.

When a schema update fails, the database schema is rolled-back to the state before the start of the Inmanta server. In that case the Inmanta server will fail to start.

6.2.3 Testing database migrations

Each database migration script should be tested using an automated test case. The tests that verify the migration from schema version <old_version> to <new_version> are stored in a file named `tests/db/test_v<old_version>_to_v<new_version>.py`.

In general, a database schema migration test has the following flow:

1. Load a database dump that uses the database schema version directly preceding the version being tested.
2. Perform assertions that verify the database schema before the migration.
3. Start the inmanta server to trigger the database migration scripts.
4. Perform assertions to verify that the migration was done correctly.

The example below shows a test for the above-mentioned database migration script.

```python
# File: tests/db/test_v202101010_to_v202102220.py
@pytest.mark.db_restore_dump(os.path.join(os.path.dirname(__file__), "dumps",
↪"v202101010.sql"))
async def test_add_new_column_to_test_table(
    migrate_db_from: abc.Callable[[], abc.Awaitable[None]],
    get_columns_in_db_table: abc.Callable[[str], list[str]],
) -> None:
    """
    Verify that the database migration script v202102220.py correctly adds the column␣
↪new_column to the table test.
    """
    # Assert state before migration
    assert "new_column" not in await get_columns_in_db_table(table_name="test")
    # Migrate DB schema
    await migrate_db_from()
    # Assert state after migration
    assert "new_column" in await get_columns_in_db_table(table_name="test")
```

The most important elements of the test case are the following:

- Line 2: The `db_restore_dump` annotation makes the `migrate_db_from` fixture load the database dump `tests/db/dumps/v202101010.sql` in the database used by the test case. This happens in the setup stage of the fixture. As such, the database contains the old version of the database schema at the beginning of the test case.
- Line 11: Verifies that the column `new_column` doesn't exist in the table test. The test case uses the fixture `get_columns_in_db_table` to obtain that information, but the `postgresql_client` fixture can be used as well to run arbitrary queries against the database.
- Line 13: Invokes the callable returned by the `migrate_db_from` fixture. This function call starts an Inmanta server against the database used by the test case, which runs the migration script being tested.
- Line 15: Verifies whether the migration script correctly added the column `new_column` to the table test.

Each commit that creates a new database version should also add a database dump for that new version to the `tests/db/dumps/` directory. Generating this dump can be done using the `tests/db/dump_tool.py` script. This script does the following:

1. Start an Inmanta server using the latest database schema available in `inmanta.db.versions` package.
2. Execute some API calls against the server to populate the database tables with some dummy data.
3. Dump the content of the database to `tests/db/dumps/v<latest_version>.sql`.

The actions to be taken after generating a new dump file are described in the docstring of the `dump_tool.py` file. If a new table or column is added using a database migration script, the developer should make sure to adjust the `dump_tool.py` script with the necessary API calls to populate the table or column if required.

6.3 Define API endpoints

This page describes how to add an API endpoint to the Inmanta server. Adding a new API endpoint requires two methods: an API method and an API handle. The API method provides the specification of the endpoint. This includes the HTTP request method, the path to the endpoint, etc. The API handle on the other hand provides the actual implementation of the endpoint.

6.3.1 API Method

The Python function that acts as an API method should be annotated using the `method` decorator. The implementation of the method should be left empty.

An example is shown in the code snippet below.

```python
import uuid
from inmanta.const import ClientType
from inmanta.protocol.decorators import method

@method(path="/project/<id>", operation="GET", client_types=[ClientType.api])
def get_project(id: uuid.UUID):
    """
        Get a project and a list of the ids of all environments.

        :param id: The id of the project to retrieve.
        :return: The project and a list of environment ids.
        :raises NotFound: The project with the given id doesn't exist.
    """
```

This API method defines an HTTP GET operation at the path `/project/<id>` which can be used by a client of type api (cli, web-console and 3rd party service). The id parameter in the path will be passed to the associate API handle. A docstring can be associated with the API method. This information will be included in the OpenAPI documentation, available via the `/docs` endpoint of the Inmanta server.

A complete list of all the arguments accepted by the `method` decorator is given below.

`decorators.method`(*operation: str = 'POST'*, *reply: bool = True*, *arg_options: dict[str, ArgOption] = {}*, *timeout: int | None = None*, *server_agent: bool = False*, *api: bool | None = None*, *agent_server: bool = False*, *validate_sid: bool | None = None*, *client_types: list[ClientType] = [ClientType.api]*, *api_version: int = 1*, *api_prefix: str = 'api'*, *envelope: bool = False*, *envelope_key: str = 'data'*) → Callable[[...], Callable]

Decorator to identify a method as a RPC call. The arguments of the decorator are used by each transport to build and model the protocol.

Parameters

- **`path`** – The url path to use for this call. This path can contain parameter names of the function. These names should be enclosed in < > brackets.
- **`operation`** – The type of HTTP operation (verb).
- **`timeout`** – nr of seconds before request it terminated.
- **`api`** – This is a call from the client to the Server (True if not server_agent and not agent_server).
- **`server_agent`** – This is a call from the Server to the Agent (reverse http channel through long poll).
- **`agent_server`** – This is a call from the Agent to the Server.
- **`validate_sid`** – This call requires a valid session, true by default if agent_server and not api
- **`client_types`** – The allowed client types for this call. The valid values are defined by the `inmanta.const.ClientType` enum.
- **`arg_options`** – Options related to arguments passed to the method. The key of this dict is the name of the arg to which the options apply. The value is another dict that can contain the following options:

  header: Map this argument to a header with the following name. reply_header: If the argument is mapped to a header, this header will also be included in the reply getter: Call this method after validation and pass its return value to the method call. This may change the type of the argument. This method can raise an HTTPException to return a 404 for example.
- **`api_version`** – The version of the api this method belongs to.
- **`api_prefix`** – The prefix of the method: /<prefix>/v<version>/<method_name>.
- **`envelope`** – Put the response of the call under an envelope with key envelope_key.
- **`envelope_key`** – The envelope key to use.

6.3.2 API Handle

An API handle function should be annotated with the `handle` decorator and should contain all the arguments of the associated API method and the parameters defined in the path of the endpoint. The names these arguments can be mapped onto a different name by passing arguments to the `handle` decorator.

An example is shown in the code snippet below.

```
import uuid
from inmanta.server import protocol
from inmanta.types import Apireturn
from inmanta import data
from inmanta.protocol import methods

```

(continues on next page)

(continued from previous page)

```python
@protocol.handle(methods.get_project, project_id="id")
async def get_project(self, project_id: uuid.UUID) -> Apireturn:
    try:
        project = await data.Project.get_by_id(project_id)
        environments = await data.Environment.get_list(project=project_id)

        if project is None:
            return 404, {"message": "The project with given id does not exist."}

        project_dict = project.to_dict()
        project_dict["environments"] = [e.id for e in environments]

        return 200, {"project": project_dict}
    except ValueError:
        return 404, {"message": "The project with given id does not exist."}

    return 500
```

The first argument of the `handle` decorator defines that this is the handle function for the `get_project` API method. The second argument remaps the `id` argument of the API method to the `project_id` argument in the handle function.

The arguments and the return type of the handle method can be any built-in Python type or a user-defined object. The input format of an API call be verified automatically using Pydantic.

An overview of all the arguments of the `handle` decorator are shown below.

`class inmanta.protocol.decorators.`**`handle`**(*method: Callable[[...], int | tuple[int, dict[str, Any] | None] | ReturnValue[ReturnTypes] | ReturnValue[None] |* BaseModel *| UUID | bool | float | datetime | str | Sequence[*BaseModel *| UUID | bool | int | float | datetime | str] | Mapping[str,* BaseModel *| UUID | bool | int | float | datetime | str] | None], api_version: int | None = None, **kwargs: str*)

Decorator for subclasses of an endpoint to handle protocol methods

Parameters

- **`method`** – A subclass of method that defines the method
- **`api_version`** – When specific this handler is only associated with a method of the specific api version. If the version is not defined, the handler is not associated with a rest endpoint.
- **`kwargs`** – Map arguments in the message from one name to an other

6.4 Documentation writing

Inmanta uses Sphinx to generate documentation.

6.4.1 Inmanta code documentation

Modules

Python core

6.4.2 Sphinx tooling

The inmanta-sphinx package provides additional sphinx directives. The directives can render inmanta module documentation and configuration documentation.

Install inmanta sphinx extension

Install the inmanta sphinx extension by installing the inmanta-sphinx package from pypi. Adding the extensions to the extension list in conf.py enables the extensions. The names are `` `sphinxcontrib.inmanta.config` `` and `` `sphinxcontrib.inmanta.dsl` ``.

This module also install the sphinx-inmanta-api script. This script can be used to generate an RST file with the full API documentation from a module. This script is used to generate for example the API docs included in the documentation on https://docs.inmanta.com

sphinxcontrib.inmanta.config

This extension loads all the defined configuration options in the Inmanta core and uses the embedded documentation to generate a config reference.

It adds the show-options directive and a number of config objects to sphinx. Use it like this to generate documentation:

```
.. show-options::

    inmanta.server.config
    inmanta.agent.config
```

sphinxcontrib.inmanta.dsl

This exention adds objects and directives to add documentation for Inmanta dsl objects such as entities, relations, …

RST files can reference to inmanta configuration code with ``` `:inmanta:entity:`std::File`` ```. This renders to *`std::File`*

sphinx-inmanta-api

This scripts generates an RST file that provides the API documentation of a module. The documentation is generated by compiling an empty project with this module included. The generator then uses the compiler representation to emit RST code, using the directives from the inmanta.dsl domain extension. This script has the following options:

- `` `--module_repo` `` A local directory that function as the repo where all modules are stored that are required to generate the API documentation.
- `` `--module` `` The name of the module to generate api docs for.
- `` `-m` `` or `` `--extra-modules` `` An optional argument that can be provided multiple times. This is a list of modules that should be loaded as well when the API docs are generated. This might be required when other modules also provided implementations that have to be listed.
- `` `--source-repo` `` The repo where the upstream source is located. This is used to include a url in the documentation.
- `` `-f` `` or `` `--file` `` The file to save the generated documentation in.

6.5 Exceptions

For more details about Compiler Exceptions, see *Compiler exceptions*

6.5.1 HTTP Exceptions

HTTP Exceptions are raised when a server request can't be completed successfully. Each exception specifies what the HTTP status code of the response should be. By using the correct exception type (and a descriptive error message) the clients can get more information about what went wrong.

`class inmanta.protocol.exceptions.`**`BaseHttpException`**`(`*status_code: int = 500, message: str | None = None, details: dict[str, Any] | None = None*`)`

Bases: `HTTPError`

A base exception for errors in the server.

Classes which extend from the BaseHttpException class cannot have mandatory arguments in their constructor. This is required to determine the status_code of the exception in `inmanta.protocol.common.MethodProperties._get_http_status_code_for_exception()`

`class inmanta.protocol.exceptions.`**`Forbidden`**`(`*message: str | None = None, details: dict[str, Any] | None = None*`)`

Bases: *`BaseHttpException`*

An exception raised when access is denied (403)

`class inmanta.protocol.exceptions.`**`UnauthorizedException`**`(`*message: str | None = None, details: dict[str, Any] | None = None*`)`

Bases: *`BaseHttpException`*

An exception raised when access to this resource is unauthorized

`class inmanta.protocol.exceptions.`**`BadRequest`**`(`*message: str | None = None, details: dict[str, Any] | None = None*`)`

Bases: *`BaseHttpException`*

This exception is raised for a malformed request `class inmanta.protocol.exceptions.`**`NotFound`**(*message: str | None = None*, *details: dict[str, Any] | None = None*)

Bases: *BaseHttpException*

This exception is used to indicate that a request or reference resource was not found.

`class inmanta.protocol.exceptions.`**`Conflict`**(*message: str | None = None*, *details: dict[str, Any] | None = None*)

Bases: *BaseHttpException*

This exception is used to indicate that a request conflicts with the current state of the resource.

`class inmanta.protocol.exceptions.`**`ServerError`**(*message: str | None = None*, *details: dict[str, Any] | None = None*)

Bases: *BaseHttpException*

An unexpected error occurred in the server

`class inmanta.protocol.exceptions.`**`ShutdownInProgress`**(*message: str | None = None*, *details: dict[str, Any] | None = None*)

Bases: *BaseHttpException*

This request can not be fulfilled because the server is going down

6.5.2 Database Schema Related Exceptions

For more details, see *Database Schema Management*

`class inmanta.data.schema.`**`TableNotFound`**

Bases: `Exception`

Raised when a table is not found in the database

`class inmanta.data.schema.`**`ColumnNotFound`**

Bases: `Exception`

Raised when a column is not found in the database

6.6 Features

A default Inmanta install comes with all features enabled by default. *config.feature-file* points to a yaml file that enables or disables features. The format of this file is:

```
slices:
    slice_name:
        feature_name: bool
```

6.7 Model Export Format

1. top level is a dict with one entry for each instance in the model
2. the key in this dict is the object reference handle
3. the value is the serialized instance
4. the serialized instance is a dict with three fields: type, attributes and relation.
5. type is the fully qualified name of the type
6. attributes is a dict, with as keys the names of the attributes and as values a dict with one entry.

7. **An attribute can have one or more of tree keys: unknows, nones and values. The "values" entry has as value a list with the attribute values.**
   If any of the values is Unknown or None, it is removed from the values array and the index at which it was removed is recorded in respective the unknowns or nones value
8. relations is like attributes, but the list of values contains the reference handles to which this relations points

Basic structure as pseudo jinja template

```
{
{% for instance in instances %}
 '{{instance.handle}}':{
        "type":"{{instance.type.fqn}}",
        "attributes":[
                {% for attribute in instance.attributes %}
                "{{attribute.name}}": [ {{ attribute.values | join(",") }} ]
                {% endfor %}
        ]
        "relations" : [
                {% for relation in instance.relations %}
                "{{relation.name}}": [
                        {% for value in relation.values %}
                                {{value.handle}}
                        {% endfor %}
                ]
                {% endfor %}
        ]

{% endif %}
}
```

6.8 Type Export Format

class inmanta.model.**Attribute**(*mytype: str*, *nullable: bool*, *multi: bool*, *comment: str*, *location: Location*)

Attribute defined on an entity

Parameters

- **mytype** (*str*) – fully qualified name of the type of this attribute
- **nullable** (*bool*) – can this attribute be null
- **multi** (*bool*) – is this attribute a list
- **comment** (*str*) – docstring for this attribute
- **location** (inmanta.model.Location) – source location where this attribute is defined

to_dict() → dict[str, Any]

Convert to serialized form:

```
{
    "type": self.type,
    "multi": self.multi,
    "nullable": self.nullable,
    "comment": self.comment,
    "location": self.location.to_dict()
}
```

class inmanta.model.DirectValue(*value: Value*)

A primitive value, directly represented in the serialized form.

Parameters
value – the value itself, as string or number

to_dict() → dict[str, Any]

Convert to serialized form:

```
{"value": self.value}
```

class inmanta.model.Entity(*parents: list[str], attributes: dict[str, Attribute], relations: dict[str, Relation], location: Location*)

An entity type

Parameters

- **parents** (*List[str]*) – parent types
- **Attribute]** (*Dict[str,*) – all attributes declared on this entity directly, by name
- **Relation]** (*Dict[str,*) – all relations declared on this entity directly, by name
- **location** (*inmanta.model.Location*) – source location this entity was defined at

to_dict() → dict[str, Any]

Convert to serialized form:

```
{
"parents": self.parents,
"attributes": {n: a.to_dict() for n, a in self.attributes.items()},
"relations": {n: r.to_dict() for n, r in self.relations.items()},
"location": self.location.to_dict(),
}
```

class inmanta.model.Location(*file: str, lnr: int*)

Position in the source

Parameters

- **file** (*str*) – source file name
- **lnr** (*int*) – line in the source file

to_dict() → dict[str, Any]

Convert to serialized form:

```
{
    "file": self.file,
    "lnr": self.lnr
}
```

class inmanta.model.ReferenceValue(*reference*)

A reference to an instance of an entity.

Parameters
reference (*str*) – the handle for the entity this value refers to

to_dict() → dict[str, Any]

Convert to serialized form:

```
{"reference": self.reference}
```

class inmanta.model.Relation(*mytype: str*, *multi: tuple[int, int | None]*, *reverse: str*, *comment: str*, *location: Location*, *source_annotations: list[Value]*, *target_annotations: list[Value]*)

A relation between two entities.

Parameters

- **mytype** (*str*) – the type this relation refers to
- **multi** (*Tuple[int, int]*) – the multiplicity of this relation in the form (lower,upper), -1 for unbounded
- **reverse** (*str*) – the fully qualified name of the inverse relation
- **location** (*inmanta.model.Location*) – source location this relation was defined at
- **source_annotations** (*List[Value]*) – annotations on this relation on the source side
- **target_annotations** (*List[Value]*) – annotations on this relation on the target side

to_dict() → dict[str, Any]

Convert to serialized form:

```
{
 "type": self.type,
 "multi": [self.multi[0], self.multi[1]],
 "reverse": self.reverse,
 "comment": self.comment,
 "location": self.location.to_dict(),
 "source_annotations": [x.to_dict() for x in self.source_annotations],
 "target_annotations": [x.to_dict() for x in self.target_annotations]
 }
```

class inmanta.model.Value

A value reference from a type either *DirectValue* or *ReferenceValue*

6.9 Platform Developers Guide

6.9.1 Dependencies

All dependencies in this project need to be pinned to specific version. These versions are pinned in requirements.txt. This file can be used to install all dependencies at once or use it as a constraint file for tox or pip install. requirements.txt contains all dependencies for the core platform, for running tests and for generating documentation.

```
# Install inmanta from current checkout
pip install -c requirements.txt .
```

https://dependabot.com monitors each dependency for updates and security issues. The inmanta development policy is to track the latest version of all dependencies.

6.9.2 Versioning

A release gets its version based on the current year and an index for the release. The release schedule targets a release every two months but this tends to slip. The latest stable release (e.g. 2017.1) gets backported bugfixes, these release get a micro version number (e.g. 2017.1.4). All versions get a tag in the git repo prefixed with v (e.g. v2017.1. Supported versions are available in a branch under stable/ for backports and bugfixes (e.g. stable/v2017.1).

Development is done in the master branch. The version of the master branch is set to the next release version, but tagged with dev. This is configured in setup.cfg with the tag_build setting. The CI/build server can generate snapshots. Snapshots also need to have the dev tag (for correct version comparison) appended with the current date in +%Y%m%d%H%M format.

```
# Tag the code and build a source dist
python setup.py egg_info -b "dev$(date +%Y%m%d%H%M)" sdist
```

6.9.3 Running tests

Inmanta unit tests are executed with pytest. In tests/conftest.py provides numerous fixtures for tests. Use python functions for new tests. If setup and teardown is required, use fixtures instead of class based tests. Currently a number of tests are still class based and are in progress of being ported to function based tests.

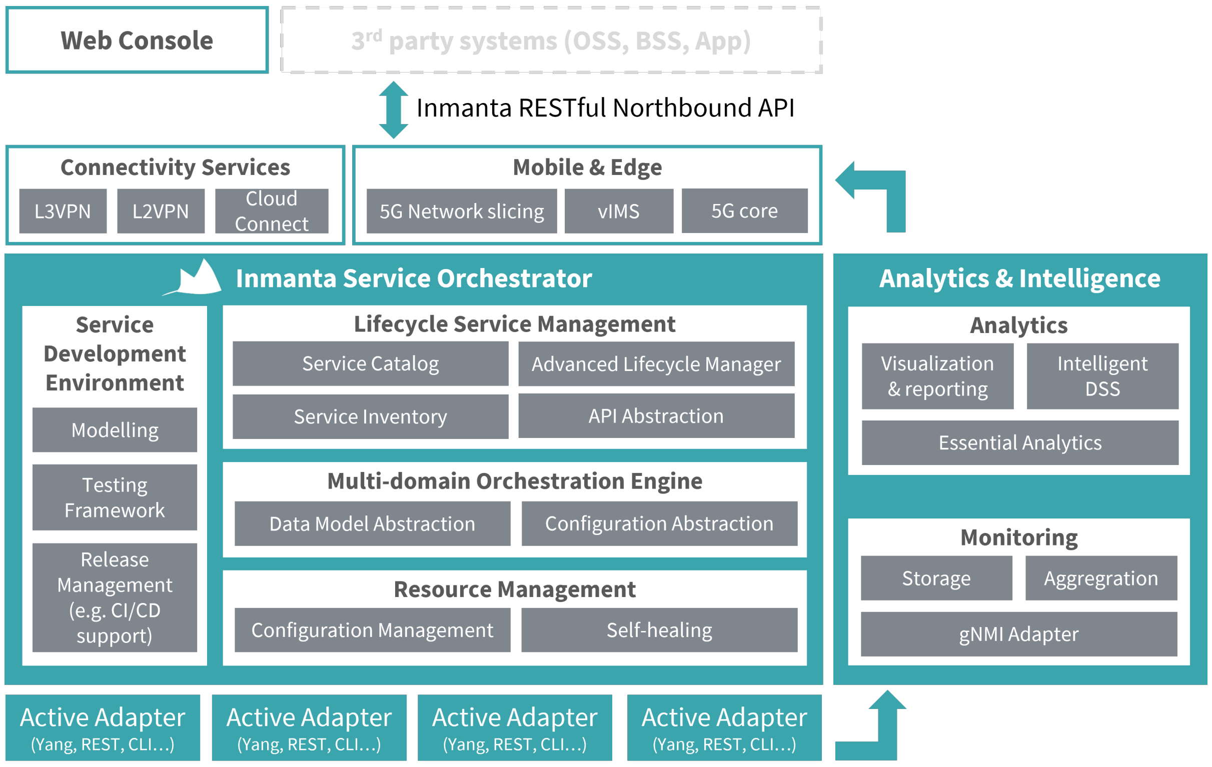
To make sure the tests run with correct dependencies installed, use tox as a testrunner. This is as simple as installing tox and executing tox in the inmanta repo. This will first run unit tests and validate code guideliness as well.

CHAPTER

SEVEN

INMANTA LIFECYCLE SERVICE MANAGER

The Inmanta LSM is an active component that governs the lifeycle of services in the orchestration model. LSM extends the Orchestration Engine (OrE) and Resource Controller (ResC) with a service catalog, a service inventory and a lifecycle manager.


Web Console
3rd party systems (OSS, BSS, App)
Inmanta RESTful Northbound API
Connectivity Services
L3VPN
L2VPN
Cloud Connect
Mobile & Edge
5G Network slicing
vIMS
5G core
Inmanta Service Orchestrator
Service Development Environment
Modelling
Testing Framework
Release Management (e.g. CI/CD support)
Lifecycle Service Management
Service Catalog
Advanced Lifecycle Manager
Service Inventory
API Abstraction
Multi-domain Orchestration Engine
Data Model Abstraction
Configuration Abstraction
Resource Management
Configuration Management
Self-healing
Analytics & Intelligence
Analytics
Visualization & reporting
Intelligent DSS
Essential Analytics
Monitoring
Storage
Aggregration
gNMI Adapter
Active Adapter (Yang, REST, CLI…)
Active Adapter (Yang, REST, CLI…)
Active Adapter (Yang, REST, CLI…)
Active Adapter (Yang, REST, CLI…)

The Resource Controller manages the low level desired state of individual resources in the managed infrastructure. This desired state is defined by the orchestration engine. The orchestration engine is responsible for the translation of high level desired state into low level desired state, based on a library of refinements.

The LSM makes the orchestator service aware: the orchestration engine can control the refinement process per service instance and change the refinement process based on external events.

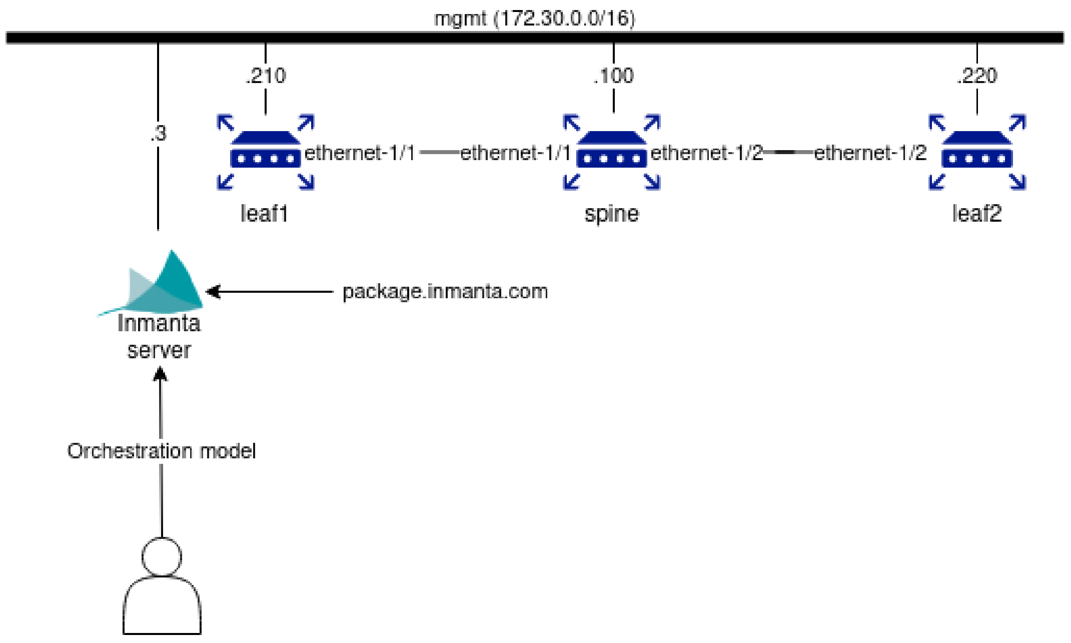
7.1 LSM quickstart

This document provides a quickstart for the Inmanta lifecycle service manager (LSM). A high-level overview will be given on how Inmanta LSM can be used to model and provision new services within a certain infrastructure. This quickstart considers a basic service, which creates and updates the IP of a given interface.

7.1.1 Overview setup

The figure shown below, gives an overview of the infrastructure required to execute this quickstart. The infrastructure consists of:

1. An Inmanta Service Orchestrator with LSM
2. Three SR Linux routers (spine, leaf1, leaf2)
3. An internet connection



The SR Linux routers in this guide are setup as a 3-node CLOS network with a spine and two leaf switches. `mgmt` is the management interface of the SR Linux routers and the Inmanta Service Orchestrator making them reachable over the management network (172.30.0.0/24).

The service modelled in the following section manages the association of IP-addresses to any interface of the SR Linux routers.

7.1.2 Prerequisites

This guide assumes that you already finished the *quickstart*, so if you haven't followed that one, please start with it.

Make sure that you have the necessary license information, namely:

- Credentials to the package repository;
- Entitlement file;
- License file.

System requirements:

- Python version 3.9 needs to be installed on your machine.

- Minimal 8GB of RAM.

Setup procedure:

1. Install Docker.
2. Install Containerlab.
3. Prepare a development environment by creating a python virtual environment and installing Inmanta:

```
$ mkdir -p ~/.virtualenvs
$ python3 -m venv ~/.virtualenvs/lsm-srlinux
$ source ~/.virtualenvs/lsm-srlinux/bin/activate
$ pip install inmanta
```

4. Change directory to the LSM SR Linux example of the examples repository:

```
$ cd examples/lsm-srlinux
```

This folder contains a project.yml, which looks like this:

```
name: LSM SR Linux Example
description: Provides an example of a LSM use case with SR Linux.
author: Inmanta
author_email: code@inmanta.com
license: ASL 2.0
copyright: 2022 Inmanta
modulepath: libs
downloadpath: libs
# This example requires licensed modules,
# replace <token> with inmanta access token you received with your license
pip:
  index_url: https://packages.inmanta.com/<token>/inmanta-service-orchestrator-8-
↪stable/python/simple/
```

Note: Additional explanation of each field can be found on the quickstart.

5. Change the <token> in the repo url to the credentials to the package repository (see *Prerequisites section*).
6. Go to the `containerlab` directory.

```
$ cd containerlab
```

7. Create a folder called `resources` in the `containerlab` folder and place your license and entitlement files there. The names of the files have to be `com.inmanta.jwe` for the entitlement file and `com.inmanta.license` for the license file.
8. *Spin-up the containers*.

```
$ docker login containers.inmanta.com
    Username: containers
    Password: <token>

    Login Succeeded
$ sudo clab deploy -t topology.yml
```

Note: Additional information about this command and how to connect to these containers can be found on the *quickstart*.

7.1.3 Orchestration model

The full orchestration model to assign an IP-address to an interface of a SR Linux router, is shown below.

```
import srlinux
import srlinux::interface as srinterface
import srlinux::interface::subinterface as srsubinterface
import srlinux::interface::subinterface::ipv4 as sripv4
import yang
import lsm
import lsm::fsm

entity InterfaceIPAssignment extends lsm::ServiceEntity:
    """
        Interface details.

        :attr router_ip: The IP address of the SR linux router that should be␣
↪configured.
        :attr router_name: The name of the SR linux router that should be configured.
        :attr interface_name: The name of the interface of the router that should be␣
↪configured.
        :attr address: The IP-address to assign to the given interface.
    """

    std::ipv_any_address router_ip
    string router_name
    string interface_name

    std::ipv_any_interface address
    lsm::attribute_modifier address__modifier="rw+"

end

implement InterfaceIPAssignment using parents, interfaceIPAssignment

implementation interfaceIPAssignment for InterfaceIPAssignment:

    device = srlinux::GnmiDevice(
            auto_agent = true,
            name = self.router_name,
            mgmt_ip = self.router_ip,
            yang_credentials = yang::Credentials(
                username = "admin",
                password = "NokiaSrl1!",
            )
        )

    resource = srlinux::Resource(
        device=device,
        identifier = self.instance_id
    )

    self.resources += resource.yang_resource

    interface = srlinux::Interface(
        device = device,
        name = self.interface_name,
```

(continues on next page)

(continued from previous page)

```
52         resource = resource,
53         mtu = 9000,
54         subinterface = srinterface::Subinterface(
55             x_index = 0,
56             ipv4=srsubinterface::Ipv4(
57                 address = sripv4::Address(
58                     ip_prefix = self.address
59                 ),
60             ),
61         ),
62         comanaged = false
63     )
64 
65 end
66 
67 
68 binding = lsm::ServiceEntityBinding(
69     service_entity="__config__::InterfaceIPAssignment",
70     lifecycle=lsm::fsm::simple,
71     service_entity_name="interface-ip-assignment",
72 )
73 
74 
75 for assignment in lsm::all(binding):
76     InterfaceIPAssignment(
77         instance_id=assignment["id"],
78         router_ip=assignment["attributes"]["router_ip"],
79         router_name=assignment["attributes"]["router_name"],
80         interface_name=assignment["attributes"]["interface_name"],
81         address=assignment["attributes"]["address"],
82         entity_binding=binding,
83     )
84 end
```

- Lines 1 to 7 import several modules required by this configuration model.
- Lines 9 to 26 define the API of the new service, i.e. the attributes required to instantiate a new instance of the service. The *InterfaceIPAssignment* entity defines four attributes: *router_ip*, *router_name*, *interface_name* and *address*. Each attribute has a description defined in the docstring above. The docstring provides documentation on the meaning of a specific service attribute. The "<attribute>__modifier" fields are meta-data fields. They defines whether the attribute can be modified or not. In the above-mentioned orchestration model, the *router_ip*, *router_name* and the *interface_name* attribute can only be set upon instantiation of the model, while the *address* attribute can be changed during the lifetime of the service. More information on attribute modifiers can be found *here*.
- Line 28 defines which implementation should be used to instantiate the *InterfaceIPAssignment* service entity.
- Lines 30 to 65 provide the actual implementation for the *InterfaceIPAssignment* service entity. If an instance is created of the *InterfaceIPAssignment* service entity, this implementation will make sure that the *address* specified in the attributes of the service instance, will be configured on the requested interface and SR Linux router.
- Lines 42 to 47 in particular, is where the resource is instantiated and assigned to the *resources* field. The *resources* field should contain the list of *resources* that need to be deployed before the state of the instance can be moved from *creating* to *up*.
- Lines 68 to 72 create a service entity binding. It associates a name and a lifecycle to the *InterfaceIPAssignment* service entity and registers it in the Inmanta Service Orchestrator via its northbound API. More information on service lifecycles can be found *here*.

- Lines 75 to 83 create an instance of the *InterfaceIPAssignment* entity for each service instance. The `lsm::all()` plugin retrieves all the service instances via the Inmanta Service Orchestrator API.

7.1.4 Install the orchestration model onto the Inmanta server

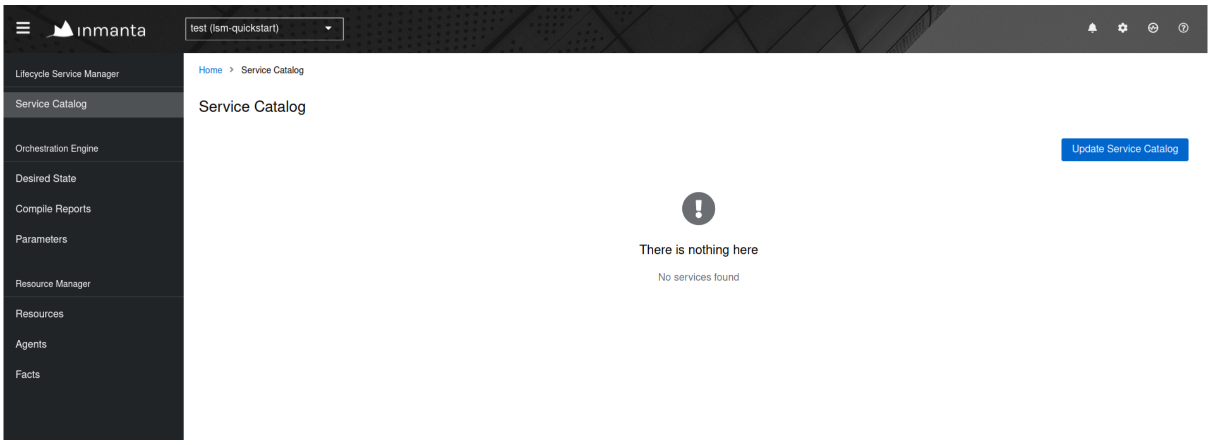
Go back to the previous folder and *create an Inmanta project and environment*.

```
# Go back to previous folder
$ cd ..
# Create a project called test
$ inmanta-cli --host 172.30.0.3 project create -n test
# Create an environment called lsm-srlinux
$ inmanta-cli --host 172.30.0.3 environment create -p test -n lsm-srlinux --save
```

The following command executes a script to copy the required resources to a specific folder inside the container.

```
$ docker exec -ti -w /code clab-srlinux-inmanta-server  /code/setup.sh
```

Afterwards, open the web-console, in this example it is on http://172.30.0.3:8888/console/.



Service Catalog

There is nothing here

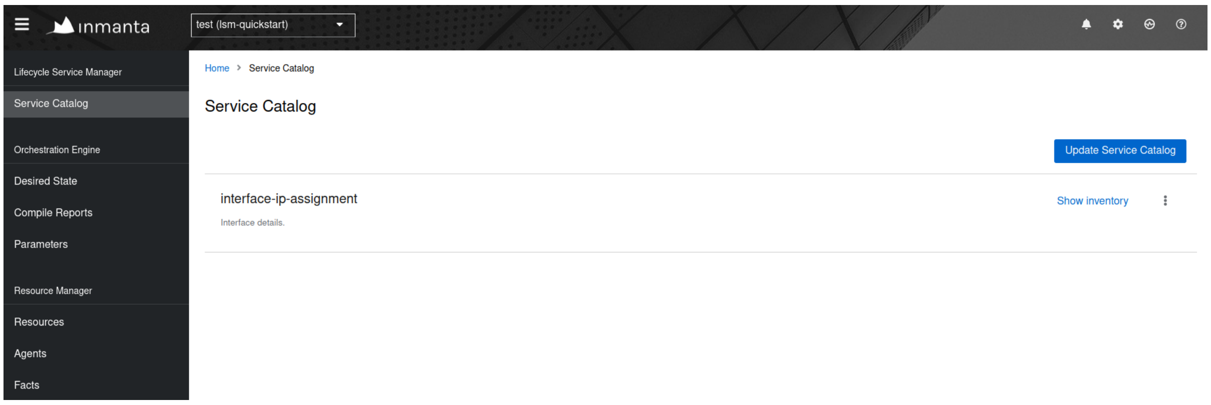
No services found

Click on the `Update Service Catalog` button. This will make the new `interface-ip-assignment` service known by the Inmanta orchestrator, making it possible to create new instances of this service via the LSM API or via the Inmanta web-console.

Clicking on the button will:

- Download all required code onto the orchestrator;
- Install the project;
- Export the service entity bindings to the service catalog.

After executing these commands, the `interface-ip-assignment` service will appear in the service catalog of the Inmanta web-console as shown in the figure below.



7.1.5 Check that the router is empty

Login into the SR Linux router named "spine" using the username "admin" and password "NokiaSrl1!".

```
$ ssh admin@clab-srlinux-spine
```

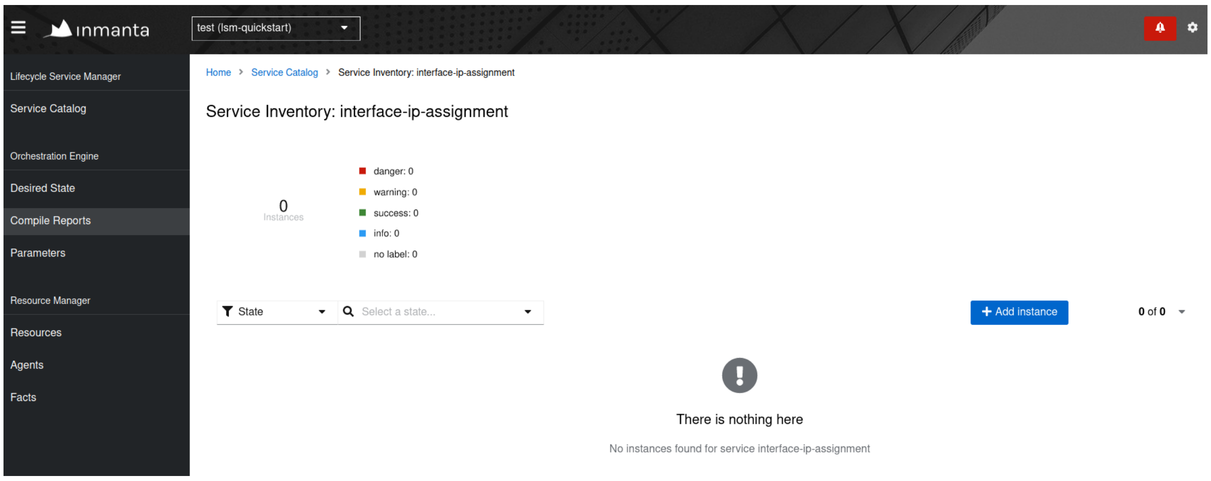
Note: Additional information on how to connect to these containers can be found on the *quickstart*. In this guide we will only do certain commands to show the changes.

Check the interface configuration via the following command.

```
A:spine# list interface
    interface ethernet-1/1 {
    }
    interface ethernet-1/2 {
    }
    interface mgmt0 {
        subinterface 0 {
            ipv4 {
                dhcp-client {
                }
            }
            ipv6 {
                dhcp-client {
                }
            }
        }
    }
```

7.1.6 Create a new service instance

Now, we will provision a new instance of the interface-ip-assignment service via the Inmanta web-console. Click on the *Show inventory* button after the vlan-assignment service and click on the *Add instance* button.

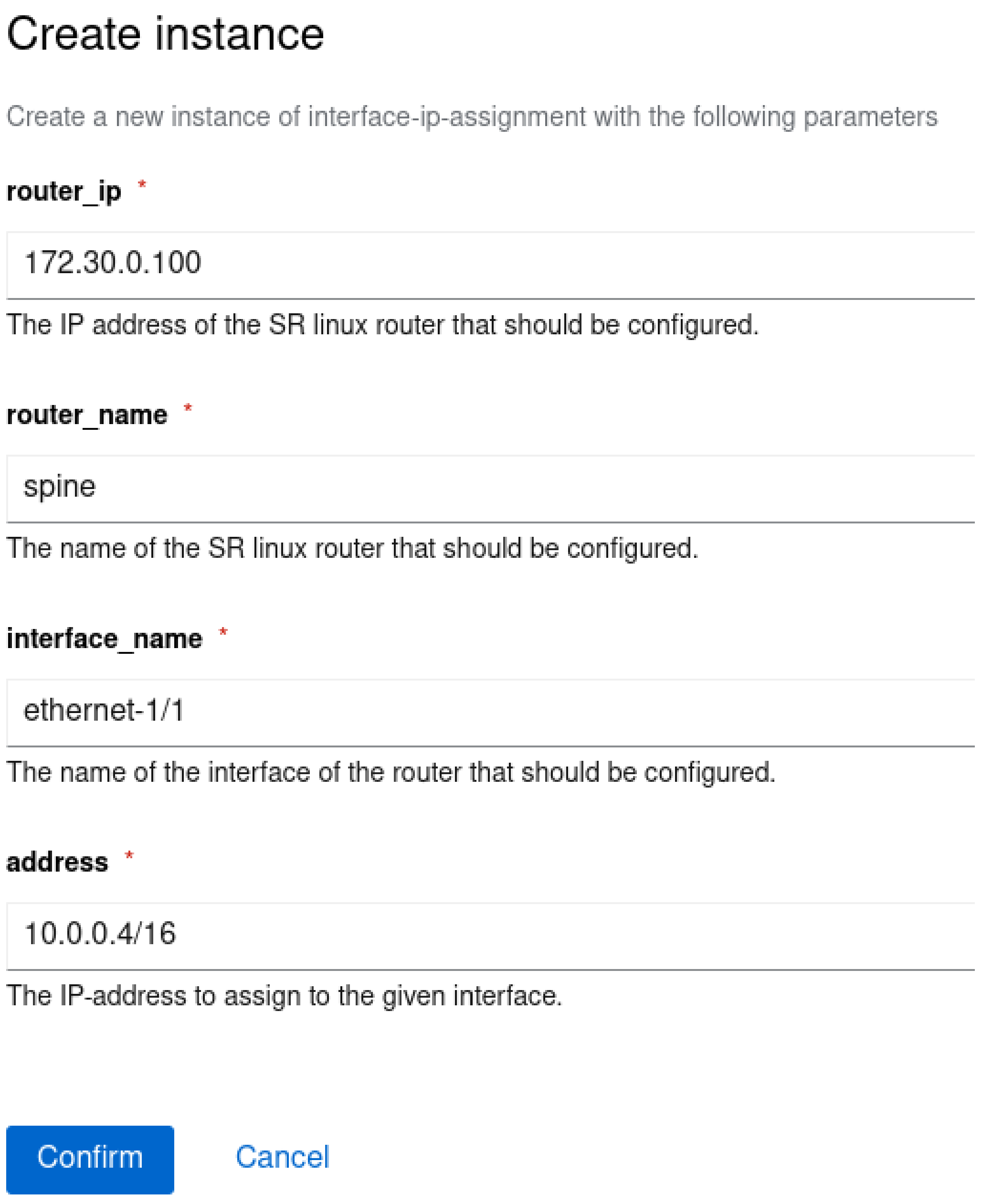


Service Inventory: interface-ip-assignment

There is nothing here

No instances found for service interface-ip-assignment

Fill in the required attributes and click on confirm.



Create instance

Create a new instance of interface-ip-assignment with the following parameters

router_ip *

172.30.0.100

The IP address of the SR linux router that should be configured.

router_name *

spine

The name of the SR linux router that should be configured.

interface_name *

ethernet-1/1

The name of the interface of the router that should be configured.

address *

10.0.0.4/16

The IP-address to assign to the given interface.

The service will be deployed automatically after clicking the *confirm* button. During the deployment, the service instance will move through different states of its lifecycle: start -> acknowledged -> creating -> up. When the service is in the up state, the interface is configured successfully. Verify the configuration on the SR Linux "spine" router.

```
A:spine# list interface
    interface ethernet-1/1 {
        subinterface 0 {
            ipv4 {
                address 10.0.0.4/16 {
                }
            }
        }
    }
    interface ethernet-1/2 {
    }
    interface mgmt0 {
        subinterface 0 {
            ipv4 {
                dhcp-client {
                }
            }
            ipv6 {
                dhcp-client {
                }
            }
        }
    }
```

7.2 Allocation

In a service lifecycle, allocation is the lifecycle stage where identifiers are allocated for use by a specific service instance.

For example a customer orders a virtual wire between two ports on two routers. The customer specifies router, port and vlan for both the A and Z side of the wire. In the network, this virtual wire is implemented as a VXlan tunnel, tied to both endpoints. Each such tunnel requires a "VXLAN Network Identifier (VNI)" that uniquely identifies the tunnel. In the allocation phase, the orchestrator selects a VNI and ensures no other customer is assigned the same VNI.

Correct allocation is crucial for the correct functioning of automated services. However, when serving multiple customers at once or when mediating between multiple inventories, correct allocation can be challenging, due to concurrency and distribution effects.

LSM offers a framework to perform allocation correctly and efficiently. The remainder of this document will explain how.

7.2.1 Types of Allocation

We distinguish several types of allocation. The next sections will explain each type, from simplest to most advanced. After the basic explanation, a more in-depth explanation is given for the different types. When first learning about LSM allocation (or allocation in general), it is important to have a basic understanding of the different types, before diving into the details.

LSM internal allocation

The easiest form of allocation is when no external inventory is involved. A range of available identifiers is assigned to LSM to distribute as it sees fit. For example, VNI range 50000-70000 is reserved to this service and can be used by LSM freely. This requires no coordination with external systems and is supported out-of-the-box.

The VNI example, allocation would look like this

Listing 1: main.cf

```
import lsm
import lsm::fsm

entity VlanAssignment extends lsm::ServiceEntity:
    string name

    int? vlan_id
    lsm::attribute_modifier vlan_id__modifier="r"
end

implement VlanAssignment using parents, do_deploy

binding = lsm::ServiceEntityBinding(
    service_entity="__config__::VlanAssignment",
    lifecycle=lsm::fsm::simple,
    service_entity_name="vlan-assignment",
    allocation_spec="allocate_vlan",
)

for assignment in lsm::all(binding):
    VlanAssignment(
        instance_id=assignment["id"],
        entity_binding=binding,
        **assignment["attributes"]
    )
end
```

The main changes in the model are:

1. the attributes that have to be allocated are added to the service definition as *r* (read only) attributes.
2. the service binding refers to an allocation spec (defined in python code)

Listing 2: plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""
```

(continues on next page)

(continued from previous page)

```

import inmanta_plugins.lsm.allocation as lsm

lsm.AllocationSpec(
    "allocate_vlan",
    lsm.LSM_Allocator(
        attribute="vlan_id", strategy=lsm.AnyUniqueInt(lower=50000, upper=70000)
    ),
)
```

The allocation spec specifies how to allocate the attribute:

1. Use the pure LSM internal allocation mechanism for *vlan_id*
2. To select a new value, use the *AnyUniqueInt* strategy, which selects a random number in the specified range

Internally, this works by storing allocations in read-only attributes on the instance. The lsm::all function ensures that if a value is already in the attribute, that value is used. Otherwise, the allocator gets an appropriate, new value, that doesn't collide with any value in any attribute-set of any other service instance.

In practice, this means that a value is allocated as long as it's in the active, candidate or rollback attribute sets of any non-terminated service instance. When a service instance is terminated, or clears one of its attribute sets, all identifiers are automatically deallocated.

Important note when designing custom lifecycles: allocation only happens during validating, and the result of the allocation is always written to the candidate attributes.

External lookup

Often, values received via the NorthBound API are not directly usable. For example, a router can be identified in the API by its name, but what is required is its management IP. The management IP can be obtained based on the name, through lookup in an inventory.

While lookup is not strictly allocation, it is in many ways similar.

The basic mechanism for external lookup is similar to internal allocation: the resolved value is stored in a read-only parameter. This is done to ensure that LSM remains stable, even if the inventory is down or corrupted. This also implies that if the inventory wants to change the value (i.e. router management IP is suddenly changed), it should notify LSM. LSM will not by itself pick up inventory changes. This notification mechanism is currently not supported yet.

An example with router management IP looks like this:

Listing 3: main.cf

```
import lsm
import lsm::fsm

entity VirtualWire extends lsm::ServiceEntity:
    string router_a
    int port_a
    int vlan_a
    string router_z
    int port_z
    int vlan_z
    int? vni
    std::ipv4_address?  router_a_mgmt_ip
    std::ipv4_address?  router_z_mgmt_ip
    lsm::attribute_modifier vni__modifier="r"
    lsm::attribute_modifier router_a_mgmt_ip__modifier="r"
```

(continues on next page)

(continued from previous page)

```
    lsm::attribute_modifier router_z_mgmt_ip__modifier="r"
    lsm::attribute_modifier router_a__modifier="rw+"
    lsm::attribute_modifier router_z__modifier="rw+"
end

implement VirtualWire using parents, do_deploy

for assignment in lsm::all(binding):
  VirtualWire(
      instance_id=assignment["id"],
      router_a = assignment["attributes"]["router_a"],
      port_a = assignment["attributes"]["port_a"],
      vlan_a = assignment["attributes"]["vlan_a"],
      router_z = assignment["attributes"]["router_z"],
      port_z = assignment["attributes"]["port_z"],
      vlan_z = assignment["attributes"]["vlan_z"],
      vni=assignment["attributes"]["vni"],
      router_a_mgmt_ip=assignment["attributes"]["router_a_mgmt_ip"],
      router_z_mgmt_ip=assignment["attributes"]["router_z_mgmt_ip"],
      entity_binding=binding,
  )
end

binding = lsm::ServiceEntityBinding(
    service_entity="__config__::VirtualWire",
    lifecycle=lsm::fsm::simple,
    service_entity_name="virtualwire",
    allocation_spec="allocate_for_virtualwire",
```

While the allocation implementation could look like the following

Listing 4: plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import os
from typing import Any, Optional

import inmanta_plugins.lsm.allocation as lsm
import psycopg2
from inmanta_plugins.lsm.allocation import (
    AllocationContext,
    ExternalAttributeAllocator,
    T,
)
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT


class PGRouterResolver(ExternalAttributeAllocator[T]):
    def __init__(self, attribute: str, id_attribute: str) -> None:
```

(continues on next page)

(continued from previous page)

```
        super().__init__(attribute, id_attribute)
        self.conn = None
        self.database = None

    def pre_allocate(self):
        """Connect to postgresql"""
        host = os.environ.get("db_host", "localhost")
        port = os.environ.get("db_port")
        user = os.environ.get("db_user")
        self.database = os.environ.get("db_name", "allocation_db")
        self.conn = psycopg2.connect(
            host=host, port=port, user=user, dbname=self.database
        )
        self.conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)

    def post_allocate(self) -> None:
        """Close connection"""
        self.conn.close()

    def needs_allocation(
        self, ctx: AllocationContext, instance: dict[str, Any]
    ) -> bool:
        attribute_not_yet_allocated = super().needs_allocation(ctx, instance)
        id_attribute_changed = self._id_attribute_changed(instance)
        return attribute_not_yet_allocated or id_attribute_changed

    def _id_attribute_changed(self, instance: dict[str, Any]) -> bool:
        if instance["candidate_attributes"] and instance["active_attributes"]:
            return instance["candidate_attributes"].get(self.id_attribute) !=␣
↪instance[
                "active_attributes"
            ].get(self.id_attribute)
        return False

    def _get_value_from_result(self, result: Optional[tuple[T]]) -> Optional[T]:
        if result and result[0]:
            return result[0]
        return None

    def allocate_for_attribute(self, id_attribute_value: Any) -> T:
        with self.conn.cursor() as cursor:
            cursor.execute(
                "SELECT mgmt_ip FROM routers WHERE name=%s", (id_attribute_value,)
            )
            result = cursor.fetchone()
            allocated_value = self._get_value_from_result(result)
            if allocated_value:
                return allocated_value
            raise Exception("No ip address found for %s", str(id_attribute_value))


lsm.AllocationSpec(
    "allocate_for_virtualwire",
    PGRouterResolver(id_attribute="router_a", attribute="router_a_mgmt_ip"),
    PGRouterResolver(id_attribute="router_z", attribute="router_z_mgmt_ip"),
    lsm.LSM_Allocator(
```

(continues on next page)

(continued from previous page)

```
        attribute="vni", strategy=lsm.AnyUniqueInt(lower=50000, upper=70000)
    ),
)
```

External inventory owns allocation

When allocating is owned externally, synchronization between LSM and the external inventory is crucial. If either LSM or the inventory fails, this should not lead to inconsistencies. In other words, LSM doesn't only have to maintain consistency between different service instances, but also between itself and the inventory.

The basic mechanism for external allocation is similar to external lookup. One important difference is that we also write our allocation to the inventory.

For example, consider that there is an external Postgres Database that contains the allocation table. In the model, this will look exactly the same as in the case of internal allocation, in the code, it will look as follows

Listing 5: plugins/__init__.py

```python
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import os
from typing import Optional
from uuid import UUID

import inmanta_plugins.lsm.allocation as lsm
import psycopg2
from inmanta_plugins.lsm.allocation import ExternalServiceIdAllocator, T
from psycopg2.extensions import ISOLATION_LEVEL_SERIALIZABLE


class PGServiceIdAllocator(ExternalServiceIdAllocator[T]):
    def __init__(self, attribute: str) -> None:
        super().__init__(attribute)
        self.conn = None
        self.database = None

    def pre_allocate(self):
        """Connect to postgresql"""
        host = os.environ.get("db_host", "localhost")
        port = os.environ.get("db_port")
        user = os.environ.get("db_user")
        self.database = os.environ.get("db_name", "allocation_db")
        self.conn = psycopg2.connect(
            host=host, port=port, user=user, dbname=self.database
        )
        self.conn.set_isolation_level(ISOLATION_LEVEL_SERIALIZABLE)

    def post_allocate(self) -> None:
        """Close connection"""
        self.conn.close()
```

(continues on next page)

(continued from previous page)

```python

    def _get_value_from_result(self, result: Optional[tuple[T]]) -> Optional[T]:
        if result and result[0]:
            return result[0]
        return None

    def allocate_for_id(self, serviceid: UUID) -> T:
        """Allocate in transaction"""
        with self.conn.cursor() as cursor:
            cursor.execute(
                "SELECT allocated_value FROM allocation WHERE attribute=%s AND owner=
↪%s",
                (self.attribute, serviceid),
            )
            result = cursor.fetchone()
            allocated_value = self._get_value_from_result(result)
            if allocated_value:
                return allocated_value
            cursor.execute(
                "SELECT max(allocated_value) FROM allocation where attribute=%s",
                (self.attribute,),
            )
            result = cursor.fetchone()
            current_max_value = self._get_value_from_result(result)
            allocated_value = current_max_value + 1 if current_max_value else 1
            cursor.execute(
                "INSERT INTO allocation (attribute, owner, allocated_value) VALUES (
↪%s, %s, %s)",
                (self.attribute, serviceid, allocated_value),
            )
            self.conn.commit()
            return allocated_value


lsm.AllocationSpec(
    "allocate_vlan",
    PGServiceIdAllocator(
        attribute="vlan_id",
    ),
)
```

What is important to notice is that the code first tries to see if an allocation has already happened. This is important in case there was a failure before LSM could commit the allocation. In general, LSM must be able to identify what has been allocated to it, in order to recover aborted operations. This is done either by attaching an identifier when performing allocation by knowing where the value will be stored in the inventory up front (e.g. the inventory contains a service model as well, LSM can find the VNI for a service by requesting the VNI for that service directly).

In the above example, the identifier is the same as the service instance id that LSM uses internally to identify an instance. An attribute of the instance can also be used to identify it in the external inventory, as the *name* attribute in the the example below.

Listing 6: plugins/__init__.py

```python
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
```

(continues on next page)

(continued from previous page)

```python
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import os
from typing import Any, Optional

import inmanta_plugins.lsm.allocation as lsm
import psycopg2
from inmanta_plugins.lsm.allocation import ExternalAttributeAllocator, T
from psycopg2.extensions import ISOLATION_LEVEL_SERIALIZABLE


class PGAttributeAllocator(ExternalAttributeAllocator[T]):
    def __init__(self, attribute: str, id_attribute: str) -> None:
        super().__init__(attribute, id_attribute)
        self.conn = None
        self.database = None

    def pre_allocate(self):
        """Connect to postgresql"""
        host = os.environ.get("db_host", "localhost")
        port = os.environ.get("db_port")
        user = os.environ.get("db_user")
        self.database = os.environ.get("db_name", "allocation_db")
        self.conn = psycopg2.connect(
            host=host, port=port, user=user, dbname=self.database
        )
        self.conn.set_isolation_level(ISOLATION_LEVEL_SERIALIZABLE)

    def post_allocate(self) -> None:
        """Close connection"""
        self.conn.close()

    def _get_value_from_result(self, result: Optional[tuple[T]]) -> Optional[T]:
        if result and result[0]:
            return result[0]
        return None

    def allocate_for_attribute(self, id_attribute_value: Any) -> T:
        """Allocate in transaction"""
        with self.conn.cursor() as cursor:
            cursor.execute(
                "SELECT allocated_value FROM allocation WHERE attribute=%s AND owner=
↪%s",
                (self.attribute, id_attribute_value),
            )
            result = cursor.fetchone()
            allocated_value = self._get_value_from_result(result)
            if allocated_value:
                return allocated_value
            cursor.execute(
                "SELECT max(allocated_value) FROM allocation where attribute=%s",
                (self.attribute,),
            )
            result = cursor.fetchone()
```

(continues on next page)

(continued from previous page)

```
            current_max_value = self._get_value_from_result(result)
            allocated_value = current_max_value + 1 if current_max_value else 1
            cursor.execute(
                "INSERT INTO allocation (attribute, owner, allocated_value) VALUES (
↪%s, %s, %s)",
                (self.attribute, id_attribute_value, allocated_value),
            )
            self.conn.commit()
            return allocated_value


lsm.AllocationSpec(
    "allocate_vlan",
    PGAttributeAllocator(attribute="vlan_id", id_attribute="name"),
)
```

Second, it is required that the inventory has a procedure to safely obtain ownership of an identifier. There must be some way LSM can definitely determine if it has correctly obtained an identifier. In the example, the database transaction ensures this. Many other mechanisms exist, but the inventory has to support at least one. Examples of possible transaction coordination mechanism are:

1. an API endpoint that atomically and consistently performs allocation,
2. database transaction
3. Compare-and-set style API (when updating a value, the old value is also passed along, ensuring no concurrent updates are possible)
4. API with version argument (like the LSM API itself, when updating a value, the version prior to update has to be passed along, preventing concurrent updates)
5. Locks and/or Leases (a value or part of the inventory can be locked or leased(locked for some time) prior to allocation, the lock ensures no concurrent modifications)

This scenario performs no de-allocation.

External inventory with deallocation

To ensure de-allocation on an external inventory is properly executed, it is not executed during compilation, but by a handler. This ensures that de-allocation is retried until it completes successfully.

The example below shows how allocation and de-allocation of a VLAN ID can be done using an external inventory. The handler of the PGAllocation entity performs the de-allocation. An instance of this entity is only constructed when the service instance is in the deallocating state.

Listing 7: vlan_assignment/model/_init.cf

```
import lsm
import lsm::fsm

entity VlanAssignment extends lsm::ServiceEntity:
    string name

    int? vlan_id
    lsm::attribute_modifier vlan_id__modifier="r"
end

implement VlanAssignment using parents, do_deploy
implement VlanAssignment using de_allocation when lsm::has_current_state(self,
```

(continues on next page)

(continued from previous page)

```
↪"deallocating")

entity PGAllocation extends std::PurgeableResource:
    """
        This entity ensures that an identifier allocated in PostgreSQL
        gets de-allocated when the service instance is removed.
    """
   string attribute
   std::uuid service_id
   string agent
end

implement PGAllocation using std::none

implementation de_allocation for VlanAssignment:
    """
        De-allocate the vlan_id identifier.
    """
    self.resources += PGAllocation(
        attribute="vlan_id",
        service_id=instance_id,
        purged=true,
        send_event=true,
        agent="internal",
        requires=self.requires,
        provides=self.provides,
    )
end

binding = lsm::ServiceEntityBinding(
    service_entity="vlan_assignment::VlanAssignment",
    lifecycle=lsm::fsm::simple_with_deallocation,
    service_entity_name="vlan-assignment",
    allocation_spec="allocate_vlan",
)

for assignment in lsm::all(binding):
    VlanAssignment(
        instance_id=assignment["id"],
        entity_binding=binding,
        **assignment["attributes"],
    )
end
```

The handler associated with the PGAllocation handler is shown in the code snippet below. Note that the handler doesn't have an implementation for the create_resource() and the update_resource() method since they can never be called. The only possible operation is a delete operation.

Listing 8: vlan_assignment/plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
```

(continues on next page)

(continued from previous page)

```
"""

import os
from typing import Optional
from uuid import UUID

import psycopg2
from inmanta.agent import handler
from inmanta.agent.handler import CRUDHandlerGeneric as CRUDHandler
from inmanta.agent.handler import ResourcePurged, provider
from inmanta.resources import PurgeableResource, resource
from inmanta_plugins.lsm.allocation import AllocationSpec, ExternalServiceIdAllocator
from psycopg2.extensions import ISOLATION_LEVEL_SERIALIZABLE


class PGServiceIdAllocator(ExternalServiceIdAllocator[int]):
    def __init__(self, attribute: str) -> None:
        super().__init__(attribute)
        self.conn = None
        self.database = None

    def pre_allocate(self) -> None:
        """Connect to postgresql"""
        host = os.environ.get("db_host", "localhost")
        port = os.environ.get("db_port")
        user = os.environ.get("db_user")
        self.database = os.environ.get("db_name", "allocation_db")
        self.conn = psycopg2.connect(
            host=host, port=port, user=user, dbname=self.database
        )
        self.conn.set_isolation_level(ISOLATION_LEVEL_SERIALIZABLE)

    def post_allocate(self) -> None:
        """Close connection"""
        self.conn.close()

    def _get_value_from_result(self, result: Optional[tuple[int]]) -> Optional[int]:
        if result and result[0]:
            return result[0]
        return None

    def allocate_for_id(self, serviceid: UUID) -> int:
        """Allocate in transaction"""
        with self.conn.cursor() as cursor:
            cursor.execute(
                "SELECT allocated_value FROM allocation WHERE attribute=%s AND owner=
↪%s",
                (self.attribute, serviceid),
            )
            result = cursor.fetchone()
            allocated_value = self._get_value_from_result(result)
            if allocated_value:
                return allocated_value
            cursor.execute(
                "SELECT max(allocated_value) FROM allocation where attribute=%s",
                (self.attribute,),
```

(continues on next page)

(continued from previous page)

```
            )
            result = cursor.fetchone()
            current_max_value = self._get_value_from_result(result)
            allocated_value = current_max_value + 1 if current_max_value else 1
            cursor.execute(
                "INSERT INTO allocation (attribute, owner, allocated_value) VALUES (
↪%s, %s, %s)",
                (self.attribute, serviceid, allocated_value),
            )
            self.conn.commit()
            return allocated_value

    def has_allocation_in_inventory(self, serviceid: UUID) -> bool:
        """
        Check whether a VLAN ID is allocated by the service instance with the given␣
↪id.
        """
        with self.conn.cursor() as cursor:
            cursor.execute(
                "SELECT allocated_value FROM allocation WHERE attribute=%s AND owner=
↪%s",
                (self.attribute, serviceid),
            )
            result = cursor.fetchone()
            allocated_value = self._get_value_from_result(result)
            if allocated_value:
                return True
            return False

    def de_allocate(self, serviceid: UUID) -> None:
        """
        De-allocate the VLAN ID allocated by the service instance with the given id.
        """
        with self.conn.cursor() as cursor:
            cursor.execute(
                "DELETE FROM allocation WHERE attribute=%s AND owner=%s",
                (self.attribute, serviceid),
            )
            self.conn.commit()


@resource("vlan_assignment::PGAllocation", agent="agent", id_attribute="service_id")
class PGAllocationResource(PurgeableResource):
    fields = ("attribute", "service_id")


@provider("vlan_assignment::PGAllocation", name="pgallocation")
class PGAllocation(CRUDHandler[PGAllocationResource]):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._allocator = PGServiceIdAllocator(attribute="vlan_id")

    def pre(self, ctx: handler.HandlerContext, resource: PGAllocationResource) ->␣
↪None:
        self._allocator.pre_allocate()

```

(continues on next page)

(continued from previous page)

```
    def post(self, ctx: handler.HandlerContext, resource: PGAllocationResource) ->␣
↪None:
        self._allocator.post_allocate()

    def read_resource(
        self, ctx: handler.HandlerContext, resource: PGAllocationResource
    ) -> None:
        if not self._allocator.has_allocation_in_inventory(resource.service_id):
            raise ResourcePurged()

    def delete_resource(
        self, ctx: handler.HandlerContext, resource: PGAllocationResource
    ) -> None:
        self._allocator.de_allocate(resource.service_id)


AllocationSpec("allocate_vlan", PGServiceIdAllocator(attribute="vlan_id"))
```

7.3 Allocation V2

Allocation V2 is a new framework, similar to allocation (v1). It happens in the same lifecycle stage and serves the same purpose: filling up read-only values of a service instance.

It comes to fill some gaps in the functionalities of allocation (v1) and takes advantages of the experience and learnings that using allocation (v1) taught us. It is a more complete, functional, and elegant framework.

7.3.1 Example

The example below show you the use case where a single allocator is used the same way on both the service instance and an embedded entity.

Listing 9: main.cf

```
import lsm
import lsm::fsm


entity ValueService extends lsm::ServiceEntity:
    string                          name
    lsm::attribute_modifier         name__modifier="rw"

    int?                            first_value
    lsm::attribute_modifier         first_value__modifier="r"
end
ValueService.embedded_values [0:] -- EmbeddedValue

entity EmbeddedValue extends lsm::EmbeddedEntity:
    string                          id
    lsm::attribute_modifier         id__modifier="rw"

    int?                            third_value
    lsm::attribute_modifier         third_value__modifier="r"

    string[]? __lsm_key_attributes = ["id"]
```

(continues on next page)

(continued from previous page)

```
end

index EmbeddedValue(id)

implement ValueService using parents
implement EmbeddedValue using std::none

binding = lsm::ServiceEntityBinding(
    service_entity="__config__::ValueService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="value-service",
    allocation_spec="value_allocation",
    strict_modifier_enforcement=true,
)

for assignment in lsm::all(binding):
    attributes = assignment["attributes"]

    service = ValueService(
        instance_id=assignment["id"],
        entity_binding=binding,
        name=attributes["name"],
        first_value=attributes["first_value"],
    )

    for embedded_value in attributes["embedded_values"]:
        service.embedded_values += EmbeddedValue(
            **embedded_value
        )
    end
end
```

Listing 10: plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2022 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

from inmanta.util import dict_path
from inmanta_plugins.lsm.allocation import AllocationSpecV2
from inmanta_plugins.lsm.allocation_v2.framework import AllocatorV2, ContextV2,␣
↪ForEach


class IntegerAllocator(AllocatorV2):
    def __init__(self, value: int, attribute: str) -> None:
        self.value = value
        self.attribute = dict_path.to_path(attribute)

    def needs_allocation(self, context: ContextV2) -> bool:
        try:
            if not context.get_instance().get(self.attribute):
```

(continues on next page)

(continued from previous page)

```
                # Attribute not present
                return True
        except IndexError:
            return True
        return False

    def allocate(self, context: ContextV2) -> None:
        context.set_value(self.attribute, self.value)


AllocationSpecV2(
    "value_allocation",
    IntegerAllocator(value=1, attribute="first_value"),
    ForEach(
        item="item",
        in_list="embedded_values",
        identified_by="id",
        apply=[
            IntegerAllocator(
                value=3,
                attribute="third_value",
            ),
        ],
    ),
)
```

7.3.2 Allocation V2 features

The two main additions to allocation v2 when compared to v1 are:

- The new *ContextV2* object (replacement for `AllocationContext` object), which goes in pair with *AllocatorV2* and *AllocationSpecV2*
- The support for allocating attributes in embedded entities.

Setting a read-only attribute on an embedded entity, like done in the above-mentioned example, is only possible when `strict_modifier_enforcement` is enabled. On legacy services, where `strict_modifier_enforcement` is not enabled, read-only attributes can be set on embedded entities using the workaround mentioned the Section *Legacy: Set attributes on embedded entities*.

Warning: To use allocation safely, allocators should not keep any state between invocations, but pass all state via the *ContextV2* object.

ContextV2

A context object that will be passed to each allocator and that should be used to set values. This context always shows the attributes the allocator should have access to, based on its level in the allocators tree. This means a top level allocator will see all the attributes, but an allocator used on embedded entities will only see the attributes of such embedded entity (as if it was a standalone entity). The context object can also be used to store values at each "level of allocation", reachable by all allocators at the same level.

In the *example* at the beginning of this page, the same allocator can be used to set a value on the service entity and an embedded entity. In `needs_allocation`, when calling `context.get_instance()`, we receive as dict the full service entity when allocating `first_value` and the embedded entity when allocating `third_value`.

AllocatorV2

A base class for all v2 allocators, they are provided with a `ContextV2` object for those two methods: `needs_allocation` and `allocate`. The main difference with v1, is that the allocate method doesn't return any value to allocate, it sets them using the context object: `context.set_value(name, value)`.

AllocationSpecV2

The collector for all `AllocatorV2`.

7.3.3 Legacy: Set attributes on embedded entities

The server doesn't have support to set read-only attributes on embedded entities when `strict_modifier_enforcement` is disabled. Thanks to the allocator `ContextV2Wrapper` and the plugin `lsm::context_v2_unwrapper` a workaround exists to do allocation on an embedded entity's attributes with `strict_modifier_enforcement` disabled. This workaround saves all the allocated values in a dict, in an attribute of the service instance (added to the instance for this single purpose). That way, the server accepts the update.

The `ContextV2Wrapper`, which has to be used at the root of the allocation tree, will collect and save all the allocated value in a single dict. And when getting all the service instances in your model, with `lsm::all`, you can simply wrap the call to `lsm::all` with a call to `lsm::context_v2_unwrapper`, which will place all the allocated values saved in the dict, directly where they belong, in the embedded entities.

When using the `ContextV2Wrapper` and the `lsm::context_v2_unwrapper` plugin, you will have to specify in which attributes all the allocated values should be saved.

Listing 11: main.cf

```
import lsm
import lsm::fsm


entity ValueService extends lsm::ServiceEntity:
    string                          name
    lsm::attribute_modifier         name__modifier="rw"

    int?                            first_value
    lsm::attribute_modifier         first_value__modifier="r"

    dict?                           allocated
    lsm::attribute_modifier         allocated__modifier="r"
end
ValueService.embedded_values [0:] -- EmbeddedValue

entity EmbeddedValue extends lsm::EmbeddedEntity:
    string                          id
    lsm::attribute_modifier         id__modifier="rw"

    int?                            third_value
    lsm::attribute_modifier         third_value__modifier="r"
end

implement ValueService using parents
implement EmbeddedValue using std::none

binding = lsm::ServiceEntityBinding(
```

(continues on next page)

(continued from previous page)

```
    service_entity="__config__::ValueService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="value-service",
    allocation_spec="value_allocation",
)

for assignment in lsm::context_v2_unwrapper(
    assignments=lsm::all(binding),
    fallback_attribute="allocated",
):
    attributes = assignment["attributes"]

    service = ValueService(
        instance_id=assignment["id"],
        entity_binding=binding,
        name=attributes["name"],
        first_value=attributes["first_value"],
        allocated=attributes["allocated"],
    )

    for embedded_value in attributes["embedded_values"]:
        service.embedded_values += EmbeddedValue(
            **embedded_value
        )
    end
end
```

Listing 12: plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2022 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

from inmanta.util import dict_path
from inmanta_plugins.lsm.allocation import AllocationSpecV2
from inmanta_plugins.lsm.allocation_v2.framework import (
    AllocatorV2,
    ContextV2,
    ContextV2Wrapper,
    ForEach,
)


class IntegerAllocator(AllocatorV2):
    def __init__(self, value: int, attribute: str) -> None:
        self.value = value
        self.attribute = dict_path.to_path(attribute)

    def needs_allocation(self, context: ContextV2) -> bool:
        try:
            if not context.get_instance().get(self.attribute):
                # Attribute not present
```

(continues on next page)

(continued from previous page)

```
                return True
        except IndexError:
            return True
        return False

    def allocate(self, context: ContextV2) -> None:
        context.set_value(self.attribute, self.value)


AllocationSpecV2(
    "value_allocation",
    IntegerAllocator(value=1, attribute="first_value"),
    ContextV2Wrapper(
        "allocated",
        ForEach(
            item="item",
            in_list="embedded_values",
            identified_by="id",
            apply=[
                IntegerAllocator(
                    value=3,
                    attribute="third_value",
                ),
            ],
        ),
    ),
)
```

To facilitate allocation on embedded entities, the ForEach allocator can be used.

Deleting of Embedded entities

When you want to support deletion of embedded entities during updates, a slightly different configuration is needed. Because all allocated values are stored in a single attribute, the deleted entities will be recreated when unwrapping.

To prevent this, use track_deletes=true on both the the allocator ContextV2Wrapper and the plugin lsm::context_v2_unwrapper

Additionally, to re-trigger allocation when an item is deleted, use SetSensitiveForEach instead of ForEach.

Listing 13: main.cf

```
import lsm
import lsm::fsm


entity ValueService extends lsm::ServiceEntity:
    string                      name
    lsm::attribute_modifier     name__modifier="rw"

    int?                        first_value
    lsm::attribute_modifier     first_value__modifier="r"

    dict?                       allocated
    lsm::attribute_modifier     allocated__modifier="r"
end
ValueService.embedded_values [0:] lsm::__rwplus__ EmbeddedValue
```

(continues on next page)

(continued from previous page)

```

entity EmbeddedValue extends lsm::EmbeddedEntity:
    string                          id
    lsm::attribute_modifier     id__modifier="rw"

    int?                            third_value
    lsm::attribute_modifier     third_value__modifier="r"

    string                          other_value
    lsm::attribute_modifier     other_value__modifier="rw"
end

index EmbeddedValue(id)

implement ValueService using parents
implement EmbeddedValue using std::none

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ValueService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="value-service",
    allocation_spec="value_allocation",
)

for assignment in lsm::context_v2_unwrapper(
    assignments=lsm::all(binding),
    fallback_attribute="allocated",
    track_deletes=true,
):
    attributes = assignment["attributes"]

    service = ValueService(
        instance_id=assignment["id"],
        entity_binding=binding,
        name=attributes["name"],
        first_value=attributes["first_value"],
        allocated=attributes["allocated"],
    )

    for embedded_value in attributes["embedded_values"]:
        service.embedded_values += EmbeddedValue(
            **embedded_value
        )
    end
end
```

Listing 14: plugins/__init__.py

```
"""
    Inmanta LSM

    :copyright: 2020 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

```

(continues on next page)

(continued from previous page)

```python
from inmanta.util import dict_path
from inmanta_plugins.lsm.allocation import AllocationSpecV2
from inmanta_plugins.lsm.allocation_v2.framework import (
    AllocatorV2,
    ContextV2,
    ContextV2Wrapper,
    SetSensitiveForEach,
)


class IntegerAllocator(AllocatorV2):
    def __init__(self, value: int, attribute: str) -> None:
        self.value = value
        self.attribute = dict_path.to_path(attribute)

    def needs_allocation(self, context: ContextV2) -> bool:
        try:
            if not context.get_instance().get(self.attribute):
                # Attribute not present
                return True
        except IndexError:
            return True
        return False

    def allocate(self, context: ContextV2) -> None:
        if self.needs_allocation(context):
            context.set_value(self.attribute, self.value)


AllocationSpecV2(
    "value_allocation",
    IntegerAllocator(value=1, attribute="first_value"),
    ContextV2Wrapper(
        "allocated",
        SetSensitiveForEach(
            item="item",
            in_list="embedded_values",
            identified_by="id",
            apply=[
                IntegerAllocator(
                    value=3,
                    attribute="third_value",
                ),
            ],
        ),
        track_deletes=True,
    ),
)
```

7.4 Allocation V3

Allocation V3 is a new framework that changes significantly compared to Allocation V2. The purpose is the same as V2: filling up the read-only values of a service instance during the first validation compile of the lifecycle. Allocation is now performed via a plugin call.

The advantage of this approach is that it simplifies greatly the process: you don't need anymore to write allocator classes and all the required functions (`needs_allocation`, `allocate`, etc.). You also don't need to instantiate many `AllocationSpecV2` with your allocators inside. Instead, you just need to write one plugin per attribute you want to allocate and register it as an `allocator`, it is less verbose and a much more straightforward approach. LSM comes with build-in allocators that can be used out of the box, e.g. `get_first_free_integer`.

7.4.1 Create an allocator

In the allocation V3 framework, an allocator is a python function returning the value to be set for a specific read-only attribute on a specific service instance. To register this function as an allocator, use the `allocation_helpers.allocator()` decorator:

```
from inmanta_plugins.lsm.allocation_helpers import allocator

@allocator()
def get_service_id(
    service: "lsm::ServiceEntity",
    attribute_path: "string",
) -> "int":
    return 5
```

An allocator must accept exactly two positional arguments:

1. `service`, the service instance for which the value is being allocated.

2. `attribute_path`, the attribute of the service instance in which the allocated value should be saved, as a *`DictPath`* expression. The decorated function can define a default value.

After those two positional arguments, the function is free of accepting any keyword argument it needs from the model and they will be passed transparently. The function can also define default values, that will be passed transparently as well.

Once an allocator is registered, it can be reused for other instances and attributes that require the same type of allocation by passing the appropriate parameters to the plugin call.

It is also possible to enforce an order in the allocators call by passing values that are returned by other plugins in the model:

Listing 15: main.cf (Plugin call ordering)

```
"""
    Inmanta LSM
    :copyright: 2024 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""


import lsm
import lsm::fsm

entity ServiceWithOrderedAllocation extends lsm::ServiceEntity:
    """
```

(continues on next page)

(continued from previous page)

```
    This service entity demonstrates how to enforce a specific order during
    the allocation process. Here we want to allocate some attributes in a
    specific order: value_allocated_first and then value_allocated_last.

    :attr name: The name identifying the service instance.
    :attr value_allocated_first: A read-only value, automatically assigned by the api
        before value_allocated_last.
    :attr value_allocated_last: A read-only value, automatically assigned by the api
        after value_allocated_first.
    """

    string                          name
    lsm::attribute_modifier         name__modifier="rw"
    string?                         value_allocated_first=null
    lsm::attribute_modifier         value_allocated_first__modifier="r"
    string?                         value_allocated_last=null
    lsm::attribute_modifier         value_allocated_last__modifier="r"
end

# Inherit parent entity's implementations
implement ServiceWithOrderedAllocation using parents


# Create a binding to enable service creation through the service catalog
ordered_allocation_binding = lsm::ServiceEntityBindingV2(
    service_entity="allocatorv3_demo::ServiceWithOrderedAllocation",
    lifecycle=lsm::fsm::simple,
    service_entity_name="allocation_order_enforcement",
)

# Collect all service instances
for assignment in lsm::all(ordered_allocation_binding):
    service = ServiceWithOrderedAllocation(
        instance_id=assignment["id"],
        entity_binding=ordered_allocation_binding,
        name=assignment["attributes"]["name"],

        # Regular allocation:
        value_allocated_first=ordered_allocation(
            service,
            "value_allocated_first"
        ),

        # Passing value_allocated_first as a parameter to this allocator
        # will enforce the ordering:
        value_allocated_last = ordered_allocation(
            service,
            "value_allocated_last",
            requires=[service.value_allocated_first]
        )

    )
end
```

On the plugin side, add an optional argument to enforce ordering:

Listing 16: __init__.py (Plugin call ordering)

```python
"""
    Copyright 2024 Inmanta

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    Contact: code@inmanta.com
"""

from datetime import datetime

from inmanta_plugins.lsm.allocation_helpers import allocator


@allocator()
def ordered_allocation(
    service: "lsm::ServiceEntity",
    attribute_path: "string",
    *,
    requires: "list?" = None
) -> "string":
    """
    For demonstration purposes, this allocator returns the current time.

    :param service: The service instance for which the attribute value
        is being allocated.
    :param attribute_path: DictPath to the attribute of the service
        instance in which the allocated value will be stored.
    :param requires: Optional list containing the results of allocator calls
        that should happen before the current call.
    """
    return str(datetime.now())
```

7.4.2 V2 to V3 migration

Moving from allocation V2 to allocation V3 boils down to the following steps:

In the plugins directory:

1. Create a specific allocator for each property of the service that requires allocation.
2. Make sure to register these allocators by decorating them with the `@allocator()` decorator.

In the model:

3. Call the relevant allocator plugin for each value requiring allocation in the `lsm::all` unwrapping.

Basic example

Here is an example of a V2 to V3 migration. For both the model and the plugin, first the old V2 version is shown and then the new version using V3 framework:

Plugin

Baseline V2 allocation in the plugins directory:

Listing 17: __init__.py (V2 allocation)

```python
"""
    Copyright 2024 Inmanta

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    Contact: code@inmanta.com
"""

from inmanta.util import dict_path
from inmanta_plugins.lsm.allocation import AllocationSpecV2
from inmanta_plugins.lsm.allocation_v2.framework import AllocatorV2, ContextV2,␣
↪ForEach


class IntegerAllocator(AllocatorV2):
    """
    Custom allocator class to set an integer value for an attribute.
    """

    def __init__(self, value: int, attribute: str) -> None:
        """
        :param value: The value to store for this attribute of this service.
        :param attribute: Attribute of the service instance in which the
            value will be stored.
        """
        self.value = value
        self.attribute = dict_path.to_path(attribute)

    def needs_allocation(self, context: ContextV2) -> bool:
        """
        Determine if this allocator has any work to do or if all
        values have already been allocated correctly for the instance
        exposed through the context object.

        :param context: Interface with the current instance
        being unwrapped in an lsm::all call.
```

(continues on next page)

(continued from previous page)

```python
        """
        try:
            if not context.get_instance().get(self.attribute):
                # Attribute not present
                return True
        except IndexError:
            return True

        return False

    def allocate(self, context: ContextV2) -> None:
        """
        Allocate the value for the attribute via the context object.

        :param context: Interface with the current instance
            being unwrapped in an lsm::all call.
        """
        context.set_value(self.attribute, self.value)


# In the allocation V2 framework, AllocationSpecV2 objects
# are used to configure the allocation process:
AllocationSpecV2(
    "value_allocation",
    IntegerAllocator(value=1, attribute="top_level_value"),
    ForEach(
        item="item",
        in_list="embedded_services",
        identified_by="id",
        apply=[
            IntegerAllocator(
                value=3,
                attribute="embedded_value",
            ),
        ],
    ),
)
```

When moving to V3, register an allocator in the plugin:

Listing 18: __init__.py (V3 allocation)

```python
"""
    Copyright 2024 Inmanta

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.
```

(continues on next page)

(continued from previous page)

```python

    Contact: code@inmanta.com
"""

from inmanta_plugins.lsm.allocation_helpers import allocator


@allocator()
def get_value(
    service: "lsm::ServiceEntity",
    attribute_path: "string",
    *,
    value: "any",
) -> "any":
    """
    Store a given value in the attributes of a service.

    :param service: The service instance for which the attribute value
        is being allocated.
    :param attribute_path: DictPath to the attribute of the service
        instance in which the allocated value will be stored.
    :param value: The value to store for this attribute of this service.
    """

    return value
```

Model

Baseline V2 allocation in the model:

Listing 19: main.cf (V2 allocation)

```
"""
    Inmanta LSM
    :copyright: 2024 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import lsm
import lsm::fsm

entity TopLevelService extends lsm::ServiceEntity:
    """
    Top-level service to demonstrate V2 allocation.

    :attr name: The name identifying the service instance.
    :attr top_level_value: A read-only value, automatically assigned by the api.
    """
    string                          name
    lsm::attribute_modifier         name__modifier="rw"
    int?                            top_level_value=null
    lsm::attribute_modifier         top_level_value__modifier="r"
end

```

(continues on next page)

(continued from previous page)

```
# Uniquely identify top level services through their name attribute
index TopLevelService(name)

# Each top level service may have zero or more embedded services attached to it
TopLevelService.embedded_services [0:] -- EmbeddedService

entity EmbeddedService extends lsm::EmbeddedEntity:
    """
    An embedded service, attached to a TopLevelService instance.

    :attr id: Identifier for this embedded service instance.
    :attr embedded_value: A read-only value, automatically assigned by the api.
    """
    string                       id
    lsm::attribute_modifier      id__modifier="rw"
    int?                         embedded_value=null
    lsm::attribute_modifier      embedded_value__modifier="r"
    string[]? __lsm_key_attributes = ["id"]
end

# Uniquely identify embedded services through their id attribute
index EmbeddedService(id)

# Inherit parent entity's implementations
implement TopLevelService using parents

implement EmbeddedService using parents

# Create a binding to enable service creation through the service catalog
value_binding = lsm::ServiceEntityBindingV2(
    service_entity="allocatorv3_demo::TopLevelService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="value-service",
    # V2 allocation requires passing the allocation_spec argument.
    # The value_allocation is defined in the plugin:
    allocation_spec="value_allocation",
    service_identity="name",
    service_identity_display_name="Name",
)

# Collect all service instances
for assignment in lsm::all(value_binding):
    attributes = assignment["attributes"]

    service = TopLevelService(
        instance_id=assignment["id"],
        entity_binding=value_binding,
        name=attributes["name"],
        top_level_value=attributes["top_level_value"],
        embedded_services=[
            EmbeddedService(
                **embedded_service
            )
            for embedded_service in attributes["embedded_services"]
        ],
    )
```

(continues on next page)

(continued from previous page)

```
end
```

When moving to V3 allocation, on the model side, call the allocators for the values requiring allocation:

Listing 20: main.cf (V3 allocation)

```
"""
    Inmanta LSM
    :copyright: 2024 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import lsm
import lsm::fsm

entity TopLevelService extends lsm::ServiceEntity:
    """
    This service entity demonstrates how a single allocator
    can be used for both a service entity and its embedded
    entities.

    :attr name: The name identifying the service instance.
    :attr top_level_value: A read-only value, automatically assigned by the api.
    """

    string                          name
    lsm::attribute_modifier         name__modifier="rw"
    int?                            top_level_value=null
    lsm::attribute_modifier         top_level_value__modifier="r"
end

# Uniquely identify top level services through their name attribute
index TopLevelService(name)

# Each top level service may have zero or more embedded services attached to it
TopLevelService.embedded_services [0:] -- EmbeddedService


entity EmbeddedService extends lsm::EmbeddedEntity:
    """
    An embedded service, attached to a TopLevelService instance.

    :attr id: Identifier for this embedded service instance.
    :attr embedded_value: A read-only value, automatically assigned by the api.
    """
    string                          id
    lsm::attribute_modifier         id__modifier="rw"
    int?                            embedded_value=null
    lsm::attribute_modifier         embedded_value__modifier="r"
    string[]? __lsm_key_attributes = ["id"]
end

# Uniquely identify embedded services through their id attribute
index EmbeddedService(id)

```

(continues on next page)

(continued from previous page)

```
# Inherit parent entity's implementations
implement TopLevelService using parents

implement EmbeddedService using parents


# Create a binding to enable service creation through the service catalog
top_level_service_binding = lsm::ServiceEntityBindingV2(
    service_entity="allocatorv3_demo::TopLevelService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="top-level-service",
    service_identity="name",
    service_identity_display_name="Name",
)


# Collect all service instances
for assignment in lsm::all(top_level_service_binding):
    attributes = assignment["attributes"]
    service = TopLevelService(
        instance_id=assignment["id"],
        entity_binding=top_level_service_binding,
        name=attributes["name"],
        # Allocator call
        top_level_value=get_value(service, "top_level_value", value=1),
        embedded_services=[
            EmbeddedService(
                id=embedded_service["id"],
                # Allocator call
                embedded_value=get_value(
                    service,
                    lsm::format(
                        "embedded_services[id={id}].embedded_value",
                        args=[],
                        kwargs=embedded_service,
                    ),
                    value=3,
                ),
            )
            for embedded_service in attributes["embedded_services"]
        ],
    )
end
```

In-depth example

This is a more complex example ensuring uniqueness for an attribute across instances within a given range of values:

Plugin

Baseline V2 allocation in the plugins directory:

Listing 21: __init__.py (V2 allocation)

```python
"""
    Copyright 2024 Inmanta

    Licensed under the Apache License, Version 2.0 (the "License");
    you may not use this file except in compliance with the License.
    You may obtain a copy of the License at

        http://www.apache.org/licenses/LICENSE-2.0

    Unless required by applicable law or agreed to in writing, software
    distributed under the License is distributed on an "AS IS" BASIS,
    WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
    See the License for the specific language governing permissions and
    limitations under the License.

    Contact: code@inmanta.com
"""

from inmanta_plugins.lsm.allocation import AllocationSpec, AnyUniqueInt, LSM_Allocator

# Define an AllocationSpec using the build-in LSM_Allocator allocator:
AllocationSpec(
    "allocate_vlan",
    LSM_Allocator(attribute="vlan_id", strategy=AnyUniqueInt(lower=50000,␣
↪upper=70000)),
)
```

This example will demonstrate how to use the `get_first_free_integer` allocator from the `lsm` module. Since we are using a plugin that is already defined, no extra plugin code is required. We will simply call this plugin from the model with the appropriate arguments.

Model

Baseline V2 allocation in the model:

Listing 22: main.cf (V2 allocation)

```
"""
    Inmanta LSM
    :copyright: 2024 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""

import lsm
```

(continues on next page)

(continued from previous page)

```
import lsm::fsm

entity VlanAssignment extends lsm::ServiceEntity:
    """
    This service entity demonstrates allocation using the LSM_Allocator
    build in lsm.

    :attr name: The name identifying the service instance.
    :attr vlan_id: A read-only value, automatically assigned by the api.
    """
    string name
    int? vlan_id=null
    lsm::attribute_modifier vlan_id__modifier="r"
end

# Inherit parent entity's implementations
implement VlanAssignment using parents

# Create a binding to enable service creation through the service catalog
vlan_binding = lsm::ServiceEntityBinding(
    service_entity="allocatorv3_demo::VlanAssignment",
    lifecycle=lsm::fsm::simple,
    service_entity_name="vlan-assignment",
    # V2 allocation requires passing the allocation_spec argument.
    # The allocate_vlan is defined in the plugin:
    allocation_spec="allocate_vlan",
)

# Collect all service instances
for assignment in lsm::all(vlan_binding):
    VlanAssignment(
        instance_id=assignment["id"],
        entity_binding=vlan_binding,
        **assignment["attributes"]
    )
end
```

When moving to V3 allocation, on the model side, call the allocators for the values requiring allocation:

Listing 23: main.cf (V3 allocation)

```
"""
    Inmanta LSM
    :copyright: 2024 Inmanta
    :contact: code@inmanta.com
    :license: Inmanta EULA
"""


import lsm
import lsm::fsm
import lsm::allocators


entity VlanAssignment extends lsm::ServiceEntity:
    """
```

(continues on next page)

(continued from previous page)

```
    This service entity demonstrates allocation using the get_first_free_integer
    allocator build in lsm.

    :attr name: The name identifying the service instance.
    :attr vlan_id: A read-only value, automatically assigned by the api.
    """

    string name
    int? vlan_id=null
    lsm::attribute_modifier vlan_id__modifier="r"
end


# Inherit parent entity's implementations
implement VlanAssignment using parents

# Create a binding to enable service creation through the service catalog
vlan_binding = lsm::ServiceEntityBindingV2(
    service_entity="allocatorv3_demo::VlanAssignment",
    lifecycle=lsm::fsm::simple,
    service_entity_name="vlan-assignment",
)

# Collect all service instances
for assignment in lsm::all(vlan_binding):
    service = VlanAssignment(
        instance_id=assignment["id"],
        entity_binding=vlan_binding,
        name=assignment["attributes"]["name"],
        # Allocator call
        vlan_id=lsm::allocators::get_first_free_integer(
            service,
            "vlan_id",
            range_start=50000,
            range_end=70000,
            # Retrieve the values already in use across services in the binding
            # and pass them as a parameter to the allocator call
            used_values=lsm::allocators::get_used_values(vlan_binding, "vlan_id"),
        )
    )
end
```

7.5 Embedded entities

In some situations, the attributes of a ServiceEntity contain a lot of duplication. Consider the following example:

Listing 24: main.cf

```
import lsm
import lsm::fsm

entity ServiceX extends lsm::ServiceEntity:
    """
        The API of ServiceX.
```

(continues on next page)

(continued from previous page)

```

        :attr service_id: A unique ID for this service.

        :attr customer_router_name: The name of the router on the customer side.
        :attr customer_router_system_ip: The system ip of the router on the customer␣
↪side.
        :attr customer_router_vendor: The vendor of the router on the customer side.
        :attr customer_router_chassis: The chassis of the router on the customer side.

        :attr provider_router_name: The name of the router on the provider side.
        :attr provider_router_system_ip: The system ip of the router on the provider␣
↪side.
        :attr provider_router_vendor: The vendor of the router on the provider side.
        :attr provider_router_chassis: The chassis of the router on the provider side.
    """
    string service_id

    string customer_router_name
    std::ipv4_address customer_router_system_ip
    lsm::attribute_modifier customer_router_system_ip__modifier="rw+"
    string customer_router_vendor
    string customer_router_chassis

    string provider_router_name
    std::ipv4_address provider_router_system_ip
    lsm::attribute_modifier provider_router_system_ip__modifier="rw+"
    string provider_router_vendor
    string provider_router_chassis
end

index ServiceX(service_id)

implement ServiceX using parents

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ServiceX",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_x",
)

for instance in lsm::all(binding):
    ServiceX(
        instance_id=instance["id"],
        entity_binding=binding,
        **instance["attributes"],
    )
end
```

Specifying the router details multiple times, results in code that is hard to read and hard to maintain. Embedded entities provide a mechanism to define a set of attributes in a separate entity. These attributes can be included in a ServiceEntity or in another embedded entity via an entity relationship. The code snippet below rewrite the above-mentioned example using the embedded entity Router:

Listing 25: main.cf

```
import lsm
```

(continues on next page)

(continued from previous page)

```
import lsm::fsm

entity ServiceX extends lsm::ServiceEntity:
    """
        The API of ServiceX.

        :attr service_id: A unique ID for this service.
    """
    string service_id
end

index ServiceX(service_id)

ServiceX.customer_router [1] -- Router
ServiceX.provider_router [1] -- Router

entity Router extends lsm::EmbeddedEntity:
    """
        Router details.

        :attr name: The name of the router.
        :attr system_ip: The system ip of the router.
        :attr vendor: The vendor of the router.
        :attr chassis: The chassis of the router.
    """
    string name
    std::ipv4_address system_ip
    lsm::attribute_modifier system_ip__modifier="rw+"
    string vendor
    string chassis
end

index Router(name)

implement ServiceX using parents
implement Router using parents

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ServiceX",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_x",
)

for instance in lsm::all(binding):
    ServiceX(
        instance_id=instance["id"],
        entity_binding=binding,
        service_id=instance["attributes"]["service_id"],
        customer_router=Router(**instance["attributes"]["customer_router"]),
        provider_router=Router(**instance["attributes"]["provider_router"]),
    )
end
```

Note, that the Router entity also defines an index on the name attribute.

7.5.1 Modelling embedded entities

This section describes the different parts of the model that are relevant when modelling an embedded entity.

Strict modifier enforcement

Each entity binding (`lsm::ServiceEntityBinding` and `lsm::ServiceEntityBindingV2`) has a feature flag called `strict_modifier_enforcement`. This flag indicates whether attribute modifiers should be enforced recursively on embedded entities or not. For new projects, it's recommended to enable this flag. Enabling it can be done in two different ways:

- Create a service binding using the `lsm::ServiceEntityBinding` entity and set the value of the attribute `strict_modifier_enforcement` explicitly to true.
- Or, create a service binding using the `lsm::ServiceEntityBindingV2` entity (recommended approach). This entity has the `strict_modifier_enforcement` flag enabled by default.

The remainder of this section assumes the `strict_modifier_enforcement` flag is enabled. If your project has `strict_modifier_enforcement` disabled for legacy reasons, consult the Section *Legacy: Embedded entities without strict_modifier_enforcement* for more information.

Defining an embedded entity

The following constraints should be satisfied for each embedded entity defined in a model:

- The embedded entity must inherit from *`lsm::EmbeddedEntity`*.
- When a bidirectional relationship is used between the embedding entity and the embedded entity, the variable name referencing the embedding entity should start with an underscore (See code snippet below).
- When a bidirectional relationship is used, the arity of the relationship towards the embedding entity should be 0 or 1.
- Relation attributes, where the other side is an embedded entity, should be prefixed with an underscore when the relation should not be included in the service definition.
- An index must be defined on an embedded entity if the relationship towards that embedded entity has an upper arity larger than one. This index is used to uniquely identify an embedded entity in a relationship. More information regarding this is available in section *Attribute modifiers on a relationship*.
- When an embedded entity is defined with the attribute modifier `__r__`, all sub-attributes of that embedded entity need to have the attribute modifier set to read-only as well. More information regarding attribute modifiers on embedded entities is available in section *Attribute modifiers on a relationship*.

The following code snippet gives an example of a bidirectional relationship to an embedded entity. Note that the name of the relationship to the embedding entity starts with an underscore as required by the above-mentioned constraints:

Listing 26: main.cf

```
import lsm
import lsm::fsm

entity ServiceX extends lsm::ServiceEntity:
    """
        The API of ServiceX.

        :attr service_id: A unique ID for this service.
    """
    string service_id
end
```

(continues on next page)

(continued from previous page)

```

index ServiceX(service_id)

ServiceX.router [1] -- Router._service [1]

entity Router extends lsm::EmbeddedEntity:
    """
        Router details.

        :attr name: The name of the router.
        :attr system_ip: The system ip of the router.
        :attr vendor: The vendor of the router.
        :attr chassis: The chassis of the router.
    """
    string name
    std::ipv4_address system_ip
    lsm::attribute_modifier system_ip__modifier="rw+"
    string vendor
    string chassis
end

index Router(name)

implement ServiceX using parents
implement Router using parents

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ServiceX",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_x",
)

for instance in lsm::all(binding):
    ServiceX(
        instance_id=instance["id"],
        entity_binding=binding,
        service_id=instance["attributes"]["service_id"],
        router=Router(**instance["attributes"]["router"]),
    )
end
```

7.5.2 Attribute modifiers on a relationship

Attribute modifiers can also be specified on relational attributes. The `--` part of the relationship definition can be replaced with either `lsm::__r__`, `lsm::__rw__` or `lsm::__rwplus__`. These attribute modifiers have the following semantics when set on a relationship:

- **__r__**: The embedded entity/entities can only be set by an allocator. If an embedded entity has this attribute modifier, all its sub-attributes should have the read-only modifier as well.
- **__rw__**: The embedded entities, part of the relationship, should be set on service instantiation. After creation, no embedded entities can be added or removed from the relationship anymore. Note that this doesn't mean that the attributes of the embedded entity cannot be updated. The latter is determined by the attribute modifiers defined on the attributes of the embedded entity.
- **__rwplus__**: After service instantiation, embedded entities can be added or removed from the relationship.

When the relationship definition contains a -- instead of one of the above-mentioned keywords, the default attribute modifier __rw__ is applied on the relationship. The code snippet below gives an example on the usage of attribute modifiers on relationships:

Listing 27: main.cf

```
import lsm
import lsm::fsm

entity ServiceX extends lsm::ServiceEntity:
    """
        The API of ServiceX.

        :attr service_id: A unique ID for this service.
    """
    string service_id
end

index ServiceX(service_id)

ServiceX.primary [1] -- SubService
ServiceX.secondary [0:1] lsm::__rwplus__ SubService

entity SubService extends lsm::EmbeddedEntity:
    """
        :attr ip: The IP address of the service
    """
    std::ipv4_address ip
end

index SubService(ip)

implement ServiceX using parents
implement SubService using parents

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ServiceX",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_x",
)

for instance in lsm::all(binding):
    service_x = ServiceX(
        instance_id=instance["id"],
        entity_binding=binding,
        service_id=instance["attributes"]["service_id"],
        primary=SubService(**instance["attributes"]["primary"]),
    )
    if instance["attributes"]["secondary"] != null:
        service_x.secondary=SubService(**instance["attributes"]["secondary"])
    end
end
```

In order to enforce the above-mentioned attribute modifiers, the inmanta server needs to be able to determine whether the embedded entities, provided in an attribute update, are an update of an existing embedded entity or a new embedded entity is being created. For that reason, each embedded entity needs to define the set of attributes that uniquely identify the embedded entity if the upper arity of the relationship is larger than one. This set of attributes is defined via an index on the embedded entity. The index should satisfy the following constraints:

- At least one non-relational attribute should be included in the index.
- Each non-relational attribute, part of the index, is exposed via the north-bound API (i.e. the name of the attribute doesn't start with an underscore).
- The index can include no other relational attributes except for the relation to the embedding entity.

The attributes that uniquely identify an embedded entity can never be updated. As such, they cannot have the attribute modifier `__rwplus__`.

If multiple indices are defined on the embedded entity that satisfy the above-mentioned constraints, one index needs to be selected explicitly by defining the `string[]? __lsm_key_attributes` attribute in the embedded entity. The default value of this attribute should contain all the attributes of the index that should be used to uniquely identify the embedded entity.

The example below defines an embedded entity `SubService` with two indices that satisfy the above-mentioned constraints. The `__lsm_key_attributes` attribute is used to indicate that the `name` attribute should be used to uniquely identify the embedded entity.

Listing 28: main.cf

```
import lsm
import lsm::fsm

entity ServiceX extends lsm::ServiceEntity:
    """
        The API of ServiceX.

        :attr service_id: A unique ID for this service.
    """
    string service_id
end

index ServiceX(service_id)

ServiceX.primary [1] -- SubService
ServiceX.secondary [0:1] lsm::__rwplus__ SubService

entity SubService extends lsm::EmbeddedEntity:
    """
        :attr name: The name of the sub-service
        :attr ip: The IP address of the service
    """
    string name
    std::ipv4_address ip
    string[]? __lsm_key_attributes = ["name"]
end

index SubService(name)
index SubService(ip)

implement ServiceX using parents
implement SubService using parents

binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::ServiceX",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_x",
)

```

(continues on next page)

(continued from previous page)

```
for instance in lsm::all(binding):
    service_x = ServiceX(
        instance_id=instance["id"],
        entity_binding=binding,
        service_id=instance["attributes"]["service_id"],
        primary=SubService(**instance["attributes"]["primary"]),
    )
    if instance["attributes"]["secondary"] != null:
        service_x.secondary=SubService(**instance["attributes"]["secondary"])
    end
end
```

If the upper arity of the relationship towards an embedded entity is one, it's not required to define an index on the embedded entity. In that case, the embedded entity will always have the same identity, no matter what the values of its attributes are. This means that there will be no difference in behavior whether the attribute modifier is set to `rw` or `rw+`. If an index is defined on the embedded entity, the attribute modifiers will be enforced in the same way as for relationships with an upper arity larger than one.

7.5.3 Legacy: Embedded entities without strict modifier enforcement

When the `strict_modifier_enforcement` flag is disabled on a service entity binding, the attribute modifiers defined on embedded entities are not enforced recursively. In that case, only the attribute modifiers defined on top-level service attributes are enforced. The following meaning applies to attribute modifiers associated with top-level relational attributes to embedded entities:

- **__r__**: The embedded entity/entities can only be set by an allocator.
- **__rw__**: The embedded entity/entities should be set on service instantiation. Afterwards the relationship object cannot be altered anymore. This means it will be impossible to add/remove entities from the relationship as well as modify any of the attributes of the embedded entity in the relationship.
- **__rwplus__**: After service instantiation, embedded entities can be updated and embedded entities can be added/removed from the relationship.

The modelling rules that apply when the `strict_modifier_enforcement` flag is disabled are less strict compared to the rules defined in *Defining an embedded entity*. The following changes apply:

- No index should be defined on an embedded entity to indicate the set of attributes that uniquely identify that embedded entity. There is also no need to set the `__lsm_key_attributes` attribute either.
- When the attribute modifier on an embedded entity is set to `__r__`, it's not required to set the attribute modifiers of all sub-attribute to read-only as well.

7.6 Inter-Service Relations

In some situations, it might be useful to specify relations between services. In the model, an inter-service-relation is indicated using a relation with the `lsm::__service__` annotation. One can also specify *Attribute modifiers on a relationship*. Consider the following example:

Listing 29: main.cf

```
import lsm
import lsm::fsm

entity Parent extends lsm::ServiceEntity:
    """
        Definition Parent
```

(continues on next page)

(continued from previous page)

```

        :attr name: The name of the parent
    """
    string name
end

index Parent(instance_id)

Child.parent_entity [1] lsm::__service__, lsm::__rwplus__ Parent

entity Child extends lsm::ServiceEntity:
    """
        Definition Child

        :attr name: The name of the child
    """
    string name
end

index Child(instance_id)

implement Parent using parents
implement Child using parents

binding_parent = lsm::ServiceEntityBinding(
service_entity="__config__::Parent",
lifecycle=lsm::fsm::service_with_delete_validate,
service_entity_name="parent_service",
)

binding_child = lsm::ServiceEntityBinding(
service_entity="__config__::Child",
lifecycle=lsm::fsm::service_with_delete_validate,
service_entity_name="child_service",
)
```

Here, an inter-service-relation is indicated for service `Child` in field `parent_entity` with arity `1` and modifier `rw+`.

7.6.1 delete-validating state

Using inter-service-relations can introduce some difficulties with deleting of instances. If we consider the previous example, deleting an instance of a `Parent` can make the configuration invalid if the instance is part of an inter-service-relation with a `Child` instance. The solution to deal with this, is to use an intermediate validation state. Some pre-constructed lifecycles also exist in the `lsm` module with additional validation states. Those lifecycles are:

- `service_with_delete_validate`
- `service_with_deallocation_and_delete_validate`
- `simple_with_delete_validate`
- `simple_with_deallocation_v2_and_delete_validate`

and use following validation states:

- `delete_validating_creating`
- `delete_validating_failed`

- `delete_validating_up`
- `delete_validating_update_failed`

To create a custom validation state, create a `State` with the `validate_self` attribute set to `null`.

If the compilation succeeds the deletion is accepted, if it fails, this means we are trying to delete an instance that is still in use in an inter-service relation. Lsm can then accordingly move the state of the service back to the original state or proceed with the delete operation.

7.7 Partial Compiles

Partial compilation is an approach to speed up compilation when the Service Inventory contains many instances.

Ordinarily, LSM re-compiles all instances on every update. This means that as the inventory grows, the compiles become slower. Partial compiles allow LSM to re-compile only those instances that are relevant to the current service instance, avoiding any slowdown.

7.7.1 Implementation guidelines

1. for every *`lsm::ServiceEntity`*,
   1. make sure to collect all resources it contains in the relation *`owned_resources`*
   2. make sure to always select the `parent` implementations (*implement … using parents*)
2. for every *Inter Service Relation*
   1. indicate if this is the relation to the owner by setting *`lsm::ServiceEntityBinding.relation_to_owner`* and *`lsm::ServiceEntityBinding.owner`*.

7.7.2 Supported scenarios

Partial compiles are possible when

1. Service Instances are unrelated: service instances don't share any resources and don't depend on each other in any way. This only requires correctly setting *`owned_resources`*.
2. Services form groups under a common owner.
   - Instances within the group can freely depend on each other and share resources, but nothing is shared across groups.
   - One specific instance is designated as the common owner of the group.
   - Instances can not be moved to another group. The model should prevent this type of update.
   - This additionally requires indicating what the owner of any service is, by setting *`lsm::ServiceEntityBinding.owner`* and *`lsm::ServiceEntityBinding.relation_to_owner`*. This does not immediately have to be the root owner, the ownership hierarchy is allowed to form a tree with intermediate owners below the root owner.
3. Service instances and groups can depend on shared resources, that are identical for all service instances and groups.
4. Any combination of the above

7.7.3 How it works for unrelated services

For unrelated services, LSM expands on the normal *resources set based partial compiles* by automatically creating a single resource set for each service instance.

To add resources to the instance's resource set, simply add them to its *lsm::ServiceBase.owned_resources* relation and make sure to select the `parents` implementation for your service entities. LSM will then make sure to populate the resource set and to correctly trigger related compiles and exports.

7.7.4 Example with Inter Service Relations

As an example, consider the following model for managing ports and routers. Both are independent services, but a port can only be managed in combination with its router and all its siblings. (This is not in general true, we often manage ports without managing the entire router, but we use it as an example.)

This model is not much different from normal *Inter Service Relations*, except for lines 29, 38, 58-59.

Listing 30: main.cf

```
1  import lsm
2  import lsm::fsm
3  import std::testing
4
5  entity Router extends lsm::ServiceEntity:
6      """
7          A service for managing routers
8      """
9      string mgmt_ip
10 end
11
12 index Router(instance_id)
13
14 entity Port extends lsm::ServiceEntity:
15     """
16         A service for managing ports on routers
17     """
18     string name
19 end
20
21 index Port(instance_id)
22
23 Port.router [1] lsm::__service__, lsm::__rwplus__ Router
24 """ An Inter Service Relation between Router and Port"""
25
26 implementation router_config for Router:
27     """ Add a dummy resource to the router to represent actual configuration """
28     self.resources += std::testing::NullResource(name=self.mgmt_ip)
29     self.owned_resources += self.resources # We own all our resources and nothing else
30 end
31
32 implementation port_config for Port:
33     """ Add a dummy resource to the Port to represent actual configuration """
34     self.resources += std::testing::NullResource(
35         name="{{self.router.mgmt_ip}}-{{self.name}}",
36         requires = self.router.resources
37     )
38     self.owned_resources += self.resources # We own all our resources and nothing else
39 end
```

(continues on next page)

(continued from previous page)

```

implement Router using router_config, parents
implement Port using port_config, parents

# Service binding for Router
binding_router = lsm::ServiceEntityBinding(
    service_entity="__config__::Router",
    lifecycle=lsm::fsm::simple_with_delete_validate,
    service_entity_name="router",
    service_identity="mgmt_ip",
)

# Service binding for Port
binding_port = lsm::ServiceEntityBinding(
    service_entity="__config__::Port",
    lifecycle=lsm::fsm::simple_with_delete_validate,
    service_entity_name="port",
    service_identity="name",
    relation_to_owner="router", # required for Partial Compile
    owner=binding_router, # required for Partial Compile
)

# Normal Service unrolling
for instance in lsm::all(binding_router):
    Router(
        instance_id = instance["id"],
        entity_binding = binding_router,
        **instance["attributes"],
    )
end

for instance in lsm::all(binding_port):
    Port(
        instance_id = instance["id"],
        entity_binding = binding_port,
        name = instance["attributes"]["name"],
        router = Router[instance_id=instance["attributes"]["router"]]
    )
end
```

7.7.5 How it works

To better understand how this works, there are two things to consider:

1. how to divide the resources into resource sets
2. how to get the correct instances into the model

Resource sets

The key mechanism behind partial compiles are `ResourceSets`: all resources in the desired state are divided into groups. When building a new desired state, instead of replacing the entire desired state, we only replace a specific `ResourceSet`. Resources in a `ResourceSet` can not depend on Resources in other `ResourceSets`.

To make this work, we have to assign every Service Instance to a `ResourceSet`, such that the set has no relations to any other `ResourceSet`.

In practice, we do this by putting all `Resources` in the `ResourceSet` of the owning entity.

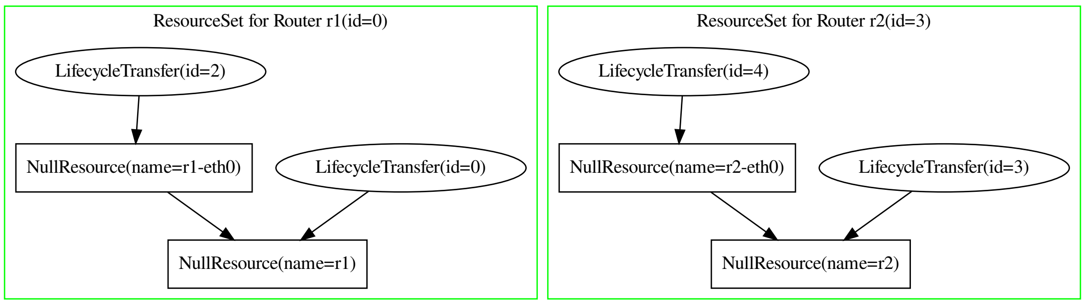


Fig. 1: Resource Sets for the Router example with 2 Routers with each 1 port. Arrows represent the requires relation.

In addition to the `ResourceSets` used by individual services, there are also `Resources` that are not in any set. These `Resources` can be shared by multiple services, with the limitation that any compile that produces them, has to produce them exactly the same. For more information see *Partial Compiles*.

Service Instance Selection

To have efficiency gains when recompiling, it is important to only build the model for all Service Instances that are in the `ResourceSet` we want to update and nothing else.

This selection is done automatically within `lsm::all`, based on the relations set between the service bindings as explained above.

The underlying mechanism is that when we recompile for a state change on any Service Instance, we first search its owner by traversing *`lsm::ServiceEntityBinding.relation_to_owner`* until we reach a single owner. Then we traverse back down the *`lsm::ServiceEntityBinding.relation_to_owner`* until we have all children. `lsm::all` will only return these children and nothing else.

7.7.6 Limitations

1. **When doing normal compiles, the model can very effectively find conflicts between services (e.g. using indexes), because it has an overview of all instances.**
   When using partial compile, conflicts between groups can not be detected, because the compiler never sees them together. This means that the model must be designed to be conflict free or rely on an (external) inventory to avoid conflicts. *This is why we always advice to run models in full compile mode until performance becomes an issue*: it gives the model time to mature and to detect subtle conflicts.
2. **Complex topologies (with multiple parents or cross-relations) are currently not supported out of the box.**
   However, complex interdependencies between service instances are often an operation risk as well. Overly entangled services are hard to reason about, debug and fix. While it is possible to develop more

complex topologies using the guidelines set out in *Partial Compiles*, it may be preferable to simplify the service design for less interdependence.

For more details, see *limitation section in the core documentation*

7.7.7 Further Reading

- *Partial Compiles*

7.8 Troubleshooting

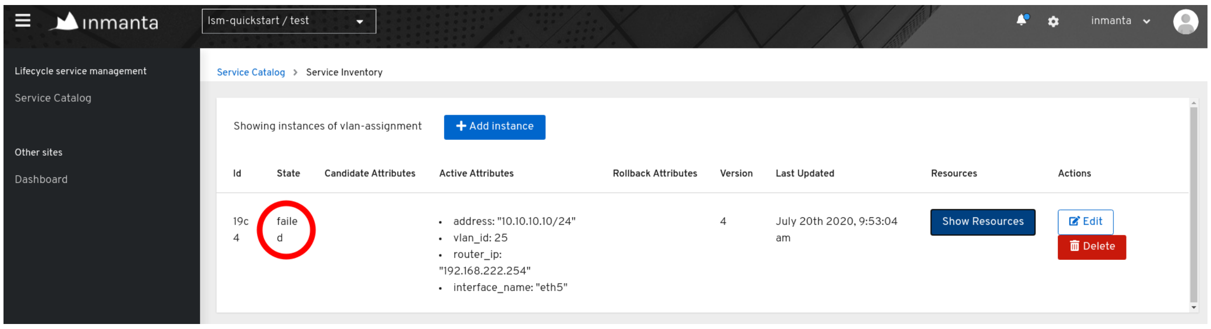
This page provides information on how to troubleshoot certain issues with Inmanta LSM in an efficient way.

7.8.1 Deployment failure

This section describes what should be done when one of the resources of a certain service instance fails. First we explain how deployment failure can be detected. The second section describes how a root cause analysis can be done via the Web Console, the CLI and the API.

Detect deployment failure

The easiest way to detect deployment failure is by looking at the state of a service instance. When the lifecycle of a service instance is modelled in such a way that the service instance enters a certain state when failure occurs, it will be easy to detect failure by looking at the state of the service instance. In the figure below, the service instance entered the failure state which indicates deployment failure.



Showing instances of vlan-assignment

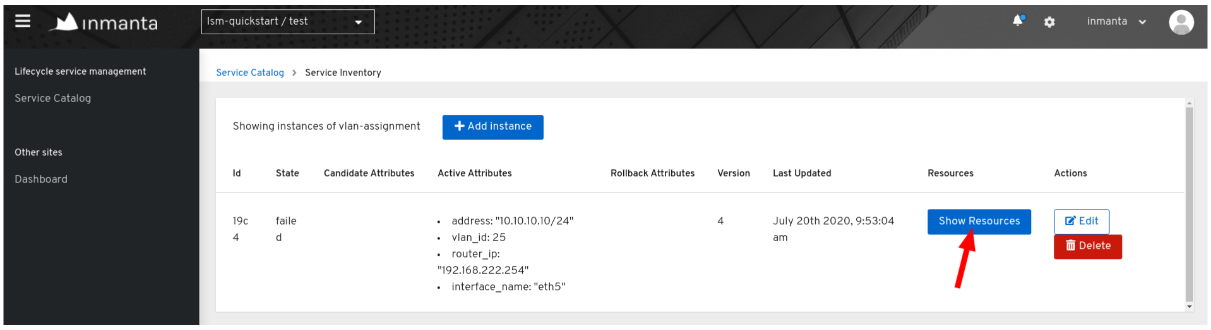
Id	State	Candidate Attributes	Active Attributes	Rollback Attributes	Version	Last Updated	Resources	Actions
19c4	failed		• address: "10.10.10.10/24" • vlan_id: 25 • router_ip: "192.168.222.254" • interface_name: "eth5"		4	July 20th 2020, 9:53:04 am	Show Resources	Edit Delete

Determine the root cause of the deployment failure

The root cause of a deployment failure can be examined via three different interfaces: the Web Console, the Inmanta client and the rest API. Each of the sections below discuss the procedure for a specific interface.

Web Console

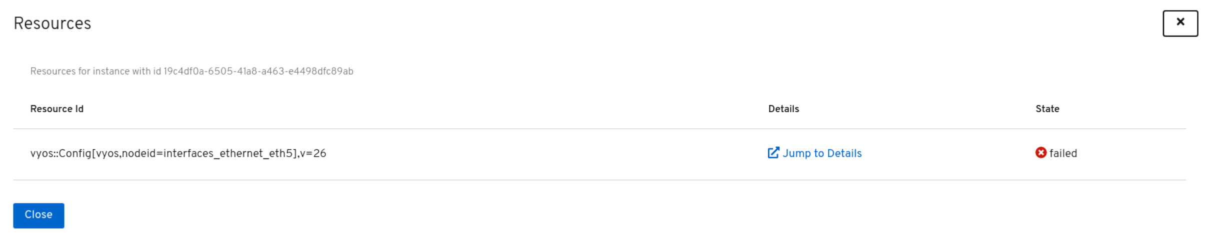
The resources that should be deployed to trigger a resource-based transfer for a specific service instance, can be obtained via the inventory of that service instance. The figure below shows an entry in the service inventory which contains service instances of the service type `vlan-assignment`. The service instance is in the state `failed` which indicates that a deployment failure has occurred.



Showing instances of vlan-assignment + Add instance

Id	State	Candidate Attributes	Active Attributes	Rollback Attributes	Version	Last Updated	Resources	Actions
19c4	failed		• address: "10.10.10.10/24" • vlan_id: 25 • router_ip: "192.168.222.254" • interface_name: "eth5"		4	July 20th 2020, 9:53:04 am	Show Resources	Edit Delete

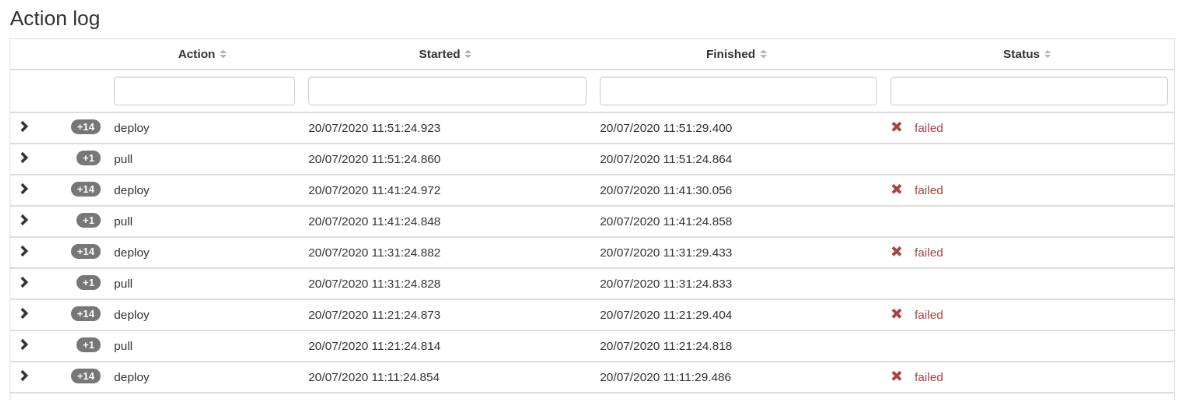
Click on the *Show Resources* button to get an overview of the different resources which are part of the resource-based transfer. This overview shows each resource, together with its deployment state. The example below only contains a single resource with the deployment state `failed`.


Resources

Resources for instance with id 19c4df0a-6505-41a8-a463-e4498dfc89ab

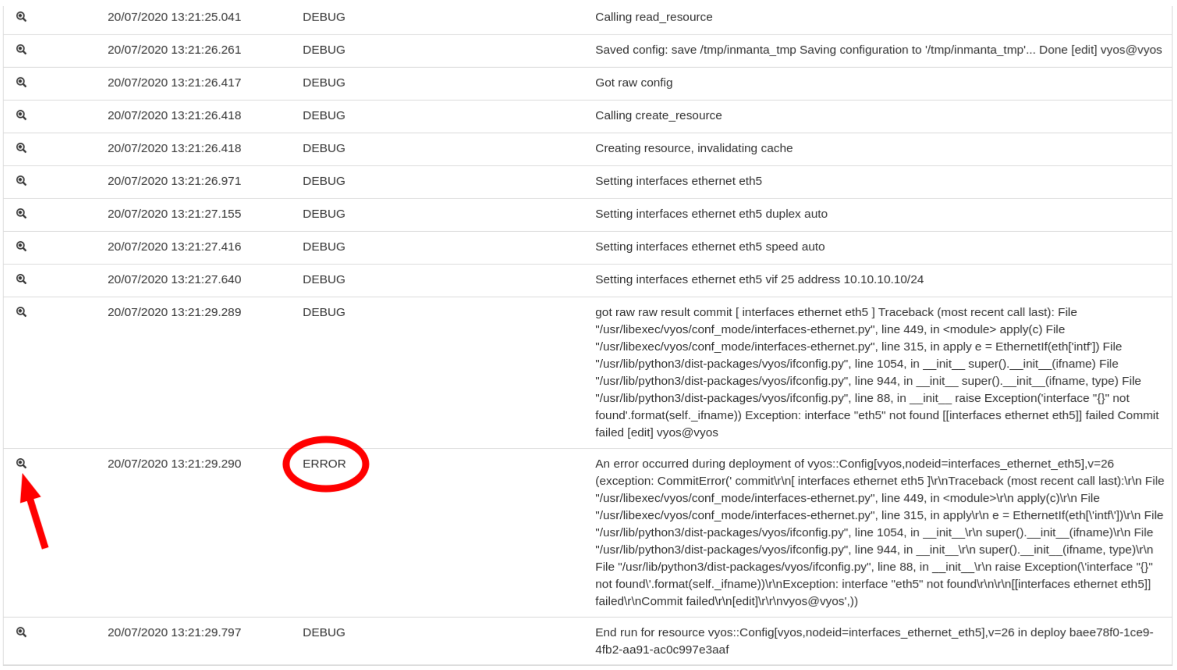
Resource Id	Details	State
vyos::Config[vyos,nodeid=interfaces_ethernet_eth5],v=26	Jump to Details	failed

When a resource ends up in the failed state, more information on the root cause of the failure can be obtained via the *Jump To Details* button. This button opens the Inmanta Dashboard and shows all the details of the failed resource, including the action log. The figure below shows the action log for the failed resource.


Action log

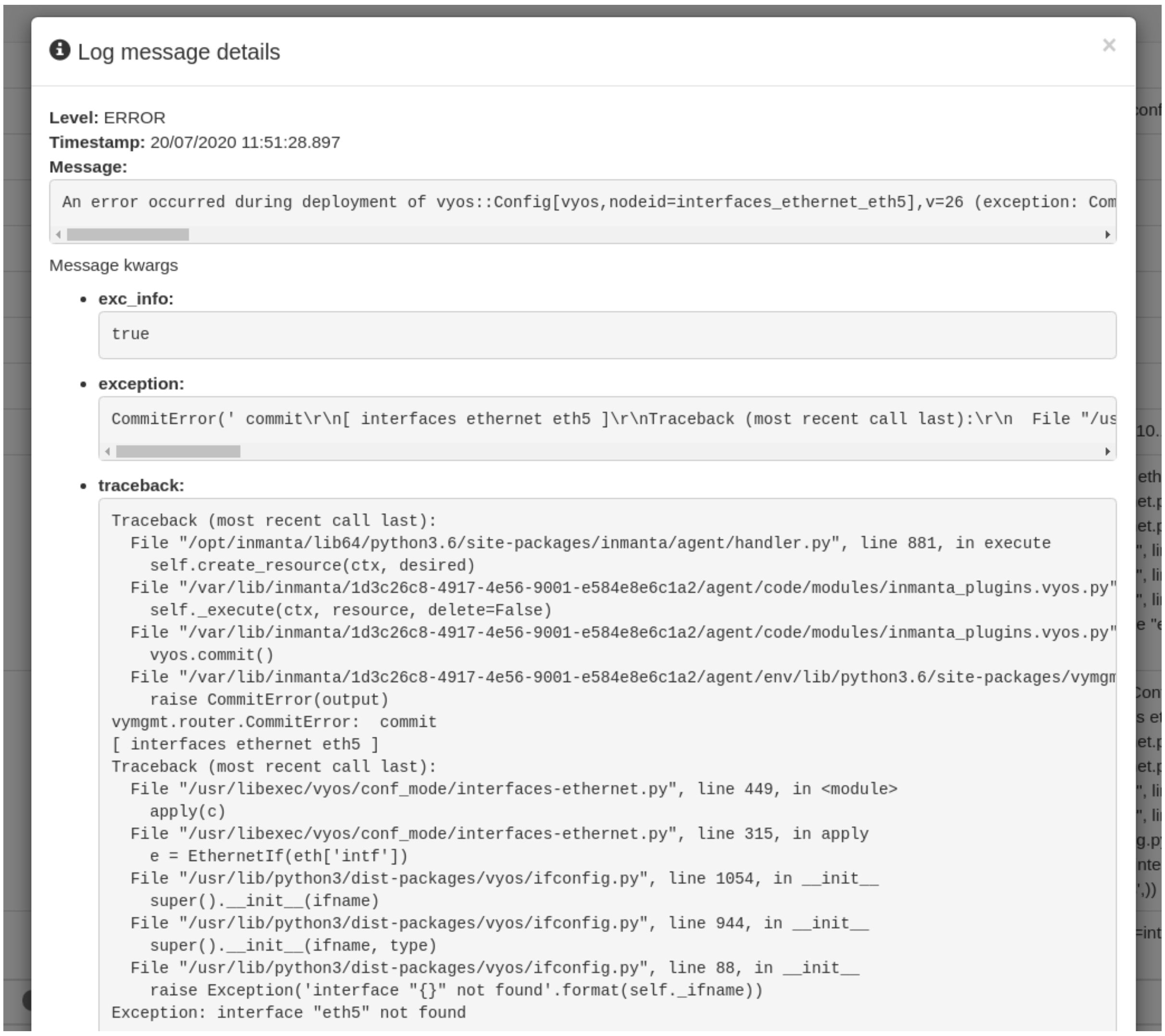
	Action	Started	Finished	Status
+14	deploy	20/07/2020 11:51:24.923	20/07/2020 11:51:29.400	failed
+1	pull	20/07/2020 11:51:24.860	20/07/2020 11:51:24.864	
+14	deploy	20/07/2020 11:41:24.972	20/07/2020 11:41:30.056	failed
+1	pull	20/07/2020 11:41:24.848	20/07/2020 11:41:24.858	
+14	deploy	20/07/2020 11:31:24.882	20/07/2020 11:31:29.433	failed
+1	pull	20/07/2020 11:31:24.828	20/07/2020 11:31:24.833	
+14	deploy	20/07/2020 11:21:24.873	20/07/2020 11:21:29.404	failed
+1	pull	20/07/2020 11:21:24.814	20/07/2020 11:21:24.818	
+14	deploy	20/07/2020 11:11:24.854	20/07/2020 11:11:29.486	failed

Each action in the action log can contains several log entries. Open the logs for the latest *deploy* action by clicking on the arrow in front of the resource action. These log entries are produced by the orchestrator itself as well as by the handler performing the deploy operation for that specific resource. The figure below shows the log entries for the deploy action.



🔍	20/07/2020 13:21:25.041	DEBUG	Calling read_resource
🔍	20/07/2020 13:21:26.261	DEBUG	Saved config: save /tmp/inmanta_tmp Saving configuration to '/tmp/inmanta_tmp'... Done [edit] vyos@vyos
🔍	20/07/2020 13:21:26.417	DEBUG	Got raw config
🔍	20/07/2020 13:21:26.418	DEBUG	Calling create_resource
🔍	20/07/2020 13:21:26.418	DEBUG	Creating resource, invalidating cache
🔍	20/07/2020 13:21:26.971	DEBUG	Setting interfaces ethernet eth5
🔍	20/07/2020 13:21:27.155	DEBUG	Setting interfaces ethernet eth5 duplex auto
🔍	20/07/2020 13:21:27.416	DEBUG	Setting interfaces ethernet eth5 speed auto
🔍	20/07/2020 13:21:27.640	DEBUG	Setting interfaces ethernet eth5 vif 25 address 10.10.10.10/24
🔍	20/07/2020 13:21:29.289	DEBUG	got raw raw result commit [interfaces ethernet eth5] Traceback (most recent call last): File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 449, in <module> apply(c) File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 315, in apply e = EthernetIf(eth['intf']) File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 1054, in __init__ super().__init__(ifname) File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 944, in __init__ super().__init__(ifname, type) File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 88, in __init__ raise Exception('interface "{}" not found'.format(self._ifname)) Exception: interface "eth5" not found [[interfaces ethernet eth5]] failed Commit failed [edit] vyos@vyos
🔍	20/07/2020 13:21:29.290	ERROR	An error occurred during deployment of vyos::Config[vyos,nodeid=interfaces_ethernet_eth5],v=26 (exception: CommitError(' commit\r\n[interfaces ethernet eth5]\r\nTraceback (most recent call last):\r\n File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 449, in <module>\r\n apply(c)\r\n File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 315, in apply\r\n e = EthernetIf(eth[\'intf\'])\r\n File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 1054, in __init__\r\n super().__init__(ifname)\r\n File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 944, in __init__\r\n super().__init__(ifname, type)\r\n File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 88, in __init__\r\n raise Exception(\'interface "{}" not found\'.format(self._ifname))\r\nException: interface "eth5" not found\r\n\r\n[[interfaces ethernet eth5]] failed\r\nCommit failed\r\n[edit]\r\r\nvyos@vyos',))
🔍	20/07/2020 13:21:29.797	DEBUG	End run for resource vyos::Config[vyos,nodeid=interfaces_ethernet_eth5],v=26 in deploy baee78f0-1ce9-4fb2-aa91-ac0c997e3aaf

The log entry with the log level ERROR gives information about what went wrong during the deployment of the resource. Click on the magnifying glass in front of the log entry to get a full stack trace of the error.



Log message details

Level: ERROR
Timestamp: 20/07/2020 11:51:28.897
Message:

```
An error occurred during deployment of vyos::Config[vyos,nodeid=interfaces_ethernet_eth5],v=26 (exception: Com
```

Message kwargs

- **exc_info:**

```
true
```

- **exception:**

```
CommitError(' commit\r\n[ interfaces ethernet eth5 ]\r\nTraceback (most recent call last):\r\n  File "/us
```

- **traceback:**

```
Traceback (most recent call last):
  File "/opt/inmanta/lib64/python3.6/site-packages/inmanta/agent/handler.py", line 881, in execute
    self.create_resource(ctx, desired)
  File "/var/lib/inmanta/1d3c26c8-4917-4e56-9001-e584e8e6c1a2/agent/code/modules/inmanta_plugins.vyos.py"
    self._execute(ctx, resource, delete=False)
  File "/var/lib/inmanta/1d3c26c8-4917-4e56-9001-e584e8e6c1a2/agent/code/modules/inmanta_plugins.vyos.py"
    vyos.commit()
  File "/var/lib/inmanta/1d3c26c8-4917-4e56-9001-e584e8e6c1a2/agent/env/lib/python3.6/site-packages/vymgm
    raise CommitError(output)
vymgmt.router.CommitError:  commit
[ interfaces ethernet eth5 ]
Traceback (most recent call last):
  File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 449, in <module>
    apply(c)
  File "/usr/libexec/vyos/conf_mode/interfaces-ethernet.py", line 315, in apply
    e = EthernetIf(eth['intf'])
  File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 1054, in __init__
    super().__init__(ifname)
  File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 944, in __init__
    super().__init__(ifname, type)
  File "/usr/lib/python3/dist-packages/vyos/ifconfig.py", line 88, in __init__
    raise Exception('interface "{}" not found'.format(self._ifname))
Exception: interface "eth5" not found
```

Inmanta client

The *lsm_services_diagnose* API method returns a diagnosis for a given service instance. It contains among others an overview of errors during the validation compile that caused the rejection.

Rest API

The *http://<host>:<port>/lsm/v1/service_inventory/<service_entity_name>/<instance_id>/diagnose* endpoint returns a diagnosis similar to the one described in the previous section.

7.8.2 Validation failure

If a service instance does not move through the expected lifecycle states but instead enters a rejected state, this means a validation failure occurred. This section describes how to detect and find the cause for validation failures.

Detect validation failure

When a validation failure occurs for a service instance, it will enter a rejected state. Each lifecycle transfer may have an associated rejected state, the convention is to include "rejected" in the name of the state. All lifecycles defined in this module follow this convention. Example states are *rejected*, *update_rejected* and *update_rejected_failed*.

Determine the cause of the validation failure

This section documents two ways to find the exception that caused the validation compile to fail. The first uses the Inmanta client from Python code, the second uses the REST API directly. The general procedure is the following: first, find the latest event (according to the *timestamp* field) for the current version with a compile id. Then use that id to pull in the compile report. Then, for the error message look at the report's compile data. For a trace look in the report for the error stream for the compile stage that failed (return code other than zero).

Inmanta client

The *lsm_services_diagnose* API method returns a diagnosis for a given service instance. It contains among others an overview of relevant resource failures.

Rest API

The *http://<host>:<port>/lsm/v1/service_inventory/<service_entity_name>/<instance_id>/diagnose* endpoint returns a diagnosis similar to the one described in the previous section.

7.9 Limitations

This section describes some limitations of the `lsm` module.

7.9.1 A ServiceEntity cannot contain resource in the undefined state

The `resources` attribute of a `ServiceEntity` cannot contain resources which are in the deployment state `undefined`. If the `resources` attribute does contain such a resource, the lifecycle state machine will get stuck in its current state and the deployment of the service will hang.

7.10 Lifecycle

7.10.1 Lifecycle State Labels

Each state in the lifecycle has a `label` attribute, which is used by the UI to mark service instances in these states visually, according to the labels.

The possible values are:

- **info**: This is the default label, it represents states the instance goes through normally, which don't require special attention.
- **success**: This label should be used for states when there are no problems with the instance, and it's stable in this state.
- **warning**: The warning label should be applied for states where the instance requires attention, because it might have run into some problems.
- **danger**: The danger label represents the situation when there are serious problems with the instance.

7.10.2 Lifecycle construction

When creating a new lifecycle, it is important to know where validation states should be added in order to avoid invalid lifecycles.

Creating/deleting and exporting/not-exporting states

If services are guaranteed to be independent, the validation states can be ommited. If one service depends on another, they could conflict when creating/exporting. If an Entity `Child` has an *inter_service_relation* to another Entity `Parent`, than the `Parent` instance should be created first. Otherwise `Child` will have a dangling reference. Another way they could come into conflict is if they use any kind of identifier: Multiple instances could be created with the same identifier. Conflicts could also arise when deleting/not-exporting: If the `Parent` instance used in a `Child` is deleted, `Child` will have a reference to something that no longer exist, which will result in an invalid desired state (see *delete-validating state*).

7.11 Attribute and entity metadata

This section describes the metadata fields that can be associated with service entities, embedded entities and its attributes and how these metadata fields can be set in the model.

7.11.1 Attribute description

Definition

The attribute description metadata is useful to provide textual information about attributes. This text will be displayed in the service catalog view of the web console.

Usage

To add a description to an attribute, create a metadata attribute with type string and whose name is the attribute's name extended with the suffix "__description".

Example

```
entity Interface :
    string interface_name
    string interface_name__description="The name of the interface"
end
```

A detailed example can be found *here*.

7.11.2 Attribute modifier

Definition

Adding the attribute modifier metadata lets the compiler know if:

- This attribute should be provided by an end-user or set by the orchestrator.
- This attribute's value is allowed to change after creation.

Usage

The modifier itself is defined like a regular attribute, with a few caveats:

- it should be of type lsm::attribute_modifier.
- its name should extend the decorated attribute's name with the suffix "__modifier".
- its value should be one of the *supported values*.

Example

```
entity Interface :
    string interface_name
    lsm::attribute_modifier interface_name__modifier="rw+"
end
```

A detailed example can be found *here*.

Supported values

- **r**: This attribute can only be set by an allocator.
- **rw**: This attribute can be set on service instantiation. It cannot be altered anymore afterwards.
- **rw+**: This attribute can be set freely during any phase of the lifecycle.

Attributes modifiers can also be specified on *relational attributes*.

7.11.3 Annotations

Definition

Annotations are key-value pairs that can be associated with an entity (service entity or embedded entity) or an attribute (simple attribute or relational attribute). These annotations don't influence the behavior of LSM or the Inmanta Service Orchestrator itself, but are intended to pass meta data to other components. For example, they can be used to pass on visualization meta-data to the the web-console to improve the user-experience.

Annotations on entities

Annotations can be attached to an entity using the `__annotations` attribute. This attribute has the type `dict` and requires a default value that defines the annotations. Each key-value pair in the dictionary contains respectively the name and the value of the annotation. The value of an annotation can be any of the simple types (string, float, int, bool), lists and dicts. Note: These values are the default values of an attribute, therefore they must be constants and cannot include varables, attribute access or plugins.

Example

The example below illustrates how the annotation `annotation=value` can be set on on a service entity. Annotations can be set on embedded entities in the same way.

```
entity Interface extends lsm::ServiceEntity:
    string interface_name
    dict __annotations = {"annotation": "value"}
end
```

Annotations on simple attributes

Annotations can be attached to simple (non-relational) attributes by defining an attribute of type dict, with a name `<attribute>__annotations`, where `<attribute>` is the name of the attribute the annotations belong to. This attribute needs a default value containing the attributes. The values of the elements in the dictionary must be strings.

Example

The example below shows how the annotation `annotation=value` is set on the attribute `interface_name`. Annotations can be set on simple attributes of embedded entities in the same way.

```
entity Interface extends lsm::ServiceEntity:
    string interface_name
    dict interface_name__annotations = {"annotation": "value"}
end
```

Annotations on relational attributes

Annotations can be attached to a relational attribute by replacing the `--` part of the relationship definition with an instance of the `lsm::RelationAnnotations` entity. This entity has a dict attribute `annotations` that represents the annotations that should be set on the relational attribute. The values of this dictionary must be strings. By convention the name of the `lsm::RelationAnnotations` instance should be prefixed and suffixed with two underscores. This improves the readability of the relationship definition.

Example

The example below illustrates how the annotation `annotation=value` can be attached to the relational attribute `ports`.

```
entity Router extends lsm::ServiceEntity:
    string name
end

entity Port extends lsm::EmbeddedEntity:
    number id
end

__annotations__ = lsm::RelationAnnotations(
    annotations={"annotation": "value"}
)
Router.ports [0:] __annotations__ Port._router [1]
```

7.12 Validation types

By default, the compiler validates the types of the attributes of a service instance. For certain types, type checking is done at the API level, prior to compilation. This provides a more interactive feedback to the operator since validation is performed before firing up the compile process. Type validation is done at the API level for the following types:

- Primitive types: int, number, string, bool, dict and list.
- Certain types defined via the *typedef statement*.

The std module already defines many commonly used types for which validation is performed at the API level (e.g., ip, url, date, . . .). The section below defines for which typedef statement validation is done at the API level.

7.12.1 Supported forms

Enumeration

Syntax

```
typedef <attr> as <type> matching self in <enumeration>
```

Examples

```
typedef colour as string matching self in ["red", "green", "blue"]
typedef bundle_size as number matching self in [1, 10, 100]
```

Regular expressions

Syntax

```
typedef <attr> as <type> matching <pattern>
```

Example

```
typedef lowercase as string matching /^[a-z]+$/
```

Number constraints

Syntax

```
typedef <attr> as <type> matching self <cmp> <value> [(or | and) <comparison_2> ...]
typedef <attr> as <type> matching <value> <cmp> self [(or | and) <comparison_2> ...]
```

Where:

- <cmp> is one of <, <=, >, >=
- <value> is any number

Example

```
typedef port_number as int matching self > 1023 and self <= 65535
```

---

std::validate_type()

The std::validate_type() function allows for finer grained type definition.

These three forms are supported:

```
typedef <attr> as <type> matching std::validate_type(<parameters>)
typedef <attr> as <type> matching std::validate_type(<parameters>) == true
typedef <attr> as <type> matching true == std::validate_type(<parameters>)
```

Example

```
typedef my_type as int matching true == std::validate_type("pydantic.conint", self, {
↪"gt": 0, "lt": 10})
```

7.13 Service Identity

For each Service Entity, it's possible to define a `Service Identity`. This is an attribute of the Service, and it can be used to identify and query the instances belonging to this service, in case using the default UUIDs is not desirable.

7.13.1 Specifying an identity

In order to use a Service Identity, the `service_identity` field of a Service Entity should be set, and point to an attribute of said Service. It's also possible to define a display name for a service entity (using the `service_identity_display_name` field), which can be used by the frontend to show these values.

There are certain rules concerning Service Identities. The attribute, that is used as an identity:

- should have an `rw` modifier
- should not be optional
- its type should be either string or int (or their constrained variants)
- its values should be unique (with regards to the service entity and environment)

An example of how the identity can be defined in the model:

```
entity TestService extends lsm::ServiceEntity:
    string service_id
end

implement TestService using std::none

binding = lsm::ServiceEntityBinding(
    service_entity="__config__::TestService",
    lifecycle=lsm::fsm::simple,
    service_entity_name="{service_entity}",
    service_identity="service_id",
    service_identity_display_name="Service ID"
)
```

7.13.2 Adding service identity to an existing entity

Adding a Service Identity to an existing Service is possible, with certain constraints:

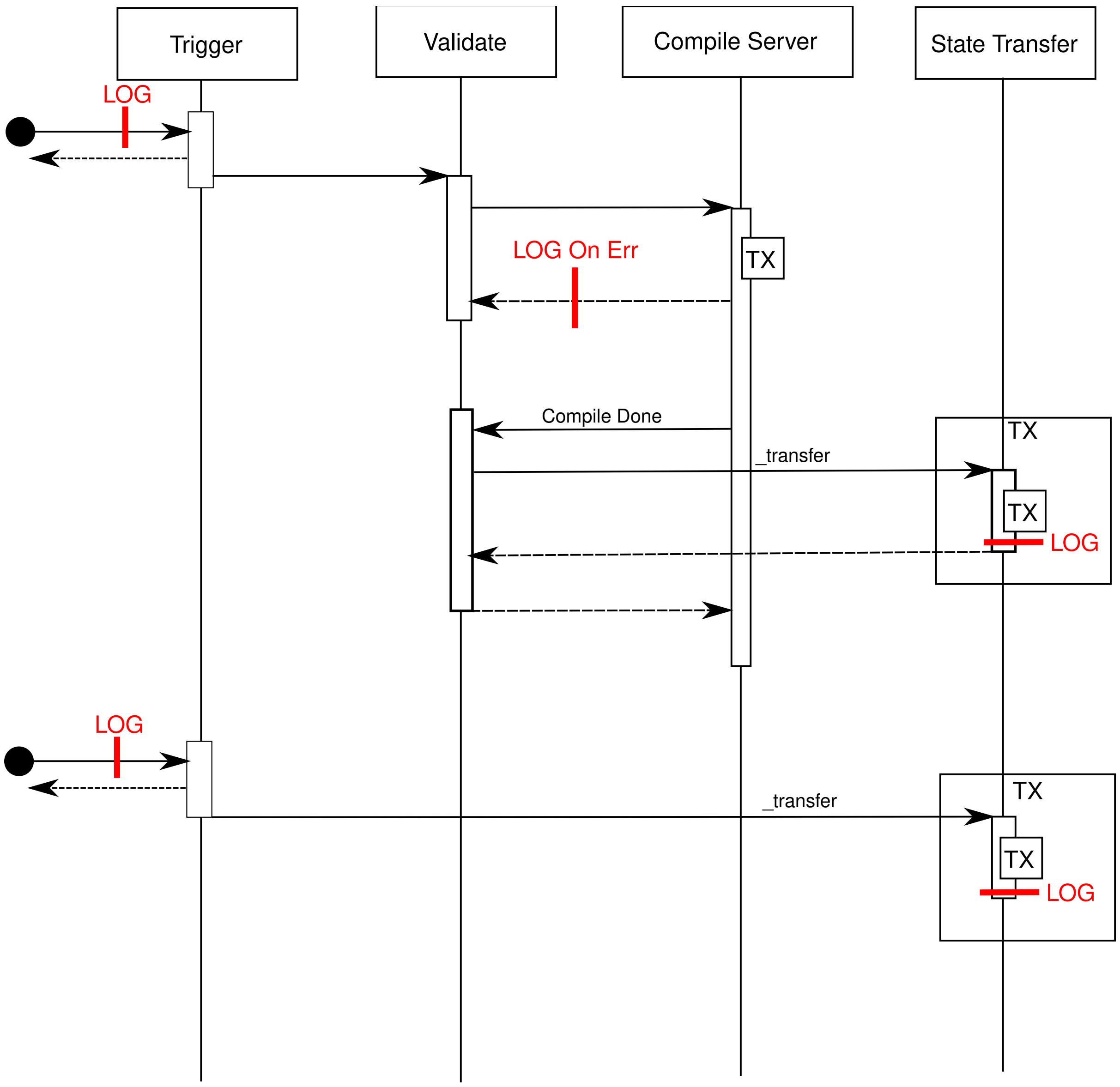
- It's not allowed to change or delete an existing identity
- If the values of a proposed identity are not unique with regards to the existing instances, the update will be rejected

7.13.3 Querying service instances using their service identity attribute

To use the service identity for querying, it can be specified as the `service_id` parameter to for the `GET` instance endpoint according to the pattern `<service_identity>=<identity_value>`, instead of a `UUID`. For example: `/service_inventory/test_entity/order_id=1234`

7.14 State Transfer Transactional Behavior

This document provides the highlights of how the transactional and logging behavior is intended and implemented.



Three important aspects are taken into account

1. *API behavior*: Async style, a valid request is accepted with a 200 return but this does not imply a successful transfer. To verify if a transfer is successful, study the event log.
2. *Event Log*: The event log is the primary mechanism to provide feedback to operators/users. It logs all triggers and all state transfers. Errors while starting a validation are also logged, as they abort the flow.
3. *Transactions*: the state transfer itself is a transaction, the associated event logging is performed in a parent transaction.
   - In case of success, both the log line and state transfer are present.
   - In case of transition failure, the sub transaction is aborted and failure is logged in the parent transaction.
   - In case of orchestrator failure, either log and state transfer are present or completely absent.

In the scenario, where the server crashes after an api_set_state has returned success, but before it was executed, the api_set_state operation is lost. **The log needs to be checked to ensure transfer**

7.14.1 Self transitions

A self transition with no action is not executed, as the state is not changed in any way.

If we would make self transition increment the state version, we would get stuck in a 'recompile storm' by resource based transaction that are triggered by repair runs.

7.14.2 Auto transition

Auto transitions are triggered after the state transfer transaction is committed. If the server fails right after the transfer, it is lost.

At server startup, the auto transitions are recovered. (i.e. all instances are checked)

This recovery has to take into account that validations are ALSO recovered. To this effect, the validation correlation ID on auto transfers are uuid3(instanceid, intancestateversion)

7.14.3 Operations

promote is only valid if there is a candidate rollback is only valid if there is a rollback set

7.14.4 Logging

For each event

1- event arrival is logged if the event is accepted (i.e. returns 200)

7.15 Service catalog

The service catalog contains all defined service entities. A service entity models a service offered by the infrastructure. A service entity is the definition of the service: its attributes and the lifecycle of the service. A service entity is created by marking an inmanta entity in a service model as a service entity and associate it with a service lifecycle. The orchestration layer will take care of defining and maintaining the definition in the service catalog.

A service entity models the following properties for a service:

- The name of the service entity
- **A list of attributes that have a name, a type (number, string, . . .) and a modifier. This modifier determines whether the attribute is:**
  - `r` / readonly: readonly from the client perspective and allocated server side by the LSM

  - rw / read-write: Attributes can be set at creation time, but are readonly after the creation
  - rw+ / read-write always: Attribute can be set at creation time and modified when the service state allows it

- The *State machine* that defines the lifecycle of the service.

For each service entity the lifecycle service manager will create an REST API endpoint on the service inventory to perform create, read, update and delete (CRUD) service instances of service entities.

7.15.1 Creating service entities

Service entities are *entities* that extend `lsm::ServiceEntity` We define attributes for the service entity the same way as for entities. We can also define a modifier for the attribute. If no modifier is defined for an attribute, it will be `rw` by default. Here is an example of an entity definition where the modifier of the `address` attribute is set to `rw+`.

```
import lsm
import lsm::fsm
import ip

entity InterfaceIPAssignment extends lsm::ServiceEntity:
    """
        Interface details.

        :attr service_id: A unique ID for this service.

        :attr router_ip: The IP address of the SR linux router that should be␣
↪configured.
        :attr router_name: The name of the SR linux router that should be configured.
        :attr interface_name: The name of the interface of the router that should be␣
↪configured.
        :attr address: The IP-address to assign to the given interface.
    """
    string service_id

    string router_ip
    string router_name
    string interface_name

    string address
    lsm::attribute_modifier address__modifier="rw+"

end

index InterfaceIPAssignment(service_id)

implement InterfaceIPAssignment using parents
```

We also need to add a lifecycle and a name to the service. This is done by creating an instance of the `ServiceEntityBinding` entity:

```
binding = lsm::ServiceEntityBindingV2(
    service_entity="__config__::InterfaceIPAssignment",
    lifecycle=lsm::fsm::service,
    service_entity_name="service_simple",
)
```

It's also possible to define a service identity for a service. For more information, see *Service Identity*.

7.16 Service Inventory

The service inventory provides an inventory with all service instances per service entity defined in the catalog. The inventory provides operations to create and delete the service instance and to update the attributes of the service instance. Additionally, it governs the lifecycle of the instance as defined in the lifecycle registered in the service catalog.

7.16.1 CRUD operations

The service inventory exposes CRUD operations on service instances in the inventory through a RESTful API:

- `GET /lsm/v1/service_inventory/<service_entity>`: List all instances of a service entity
- `POST /lsm/v1/service_inventory/<service_entity>`: Create a new service entity
- `GET /lsm/v1/service_inventory/<service_entity>/<service_id>`: Get the current state of the service instance with id `service_id`
- `PATCH /lsm/v1/service_inventory/<service_entity>/<service_id>`: Update the attributes of the service instance with id `service_id`
- `DELETE /lsm/v1/service_inventory/<service_entity>/<service_id>`: Delete the service instance with id `service_id`
- `POST /lsm/v1/services/<service_type>/<service_id>/state`: Request a state transfer for the service instance with id `service_id`

The state machine attached to the lifecycle will determine whether the API call is successful or not.

7.17 Lifecycle Manager

7.17.1 State machine

The lifecycle of service instance is governed by a state machine. A state machine is represented as a directed graph: the nodes represent the different service states and the edges represent the possible transfers that are allowed. The states determine how the service instance is treated in the orchestration engine and the transfers determine what operations are possible on a service instance.

Each service instance is in a state defined in the lifecycle state machine attached to the service entity. The state defines how the orchestration engine handles that specific service instance in its refinement process. The orchestration engine can refine a full orchestration model in two modes:

- validation: It validates the orchestration model but does not generate a resource model for the resource controller.
- production: It validates the orchestration model and generates a resource model that the resource controller deploys and enforces on the managed infrastructure.

The value of the `model_state` attribute of the state determines how the service instance is handled by the orchestration engine:

- `inactive`: never include this instance in the orchestration model.
- `candidate`: include the instance in validation mode. In validation mode the candidate set of attributes are used. Only this candidate and all designed and active instances are included in the orchestration model.
- `designed`: this indicates a candidate is accepted and is queued to become active in the model. It is included in validation mode but its resources are not yet pushed to the resource controller. The orchestration engine uses the candidate attribute set.
- `active`: include in both modes (validate and production). The orchestration engine uses the active attribute set.

Once a service instance goes to active the lifecycle should also support updates. A service instance has three sets of attributes to support this: candidate, active and rollback. When the service instance is in candidate and designed mode but it has attributes in the active set, they are included in production mode with their active attribute set. The target_operation and error_operation action on state transfers control the contents of these attribute sets. These operations are discussed later on.

Transfers between states determine how the lifecycle of the service instance reacts on external events. Each transfer has a source state, a target state and an error state. The following events can trigger a state transfer:

- the creation of the service instance: The state of the new service instance is set to the start state defined in the state machine. Set attributes provided with the API call are stored in the candidate_attributes set of the instance.
- `auto`: This transfer is automatically performed when the lifecycle arrives in the source state. Auto transfers can be disabled by adding a configuration option.
- `api set state` call: When a set state API call is performed with matching source and target states
- `on_update`: Transfers marked as on_update are executed when a PATCH is performed on a service instance. The update attributes are stored based on the target_operation or error_operation attribute.
- `on_delete`: Transfers marked as on_delete are executed when a DELETE is performed on a service instance.
- resource based: This transfer is triggered when the orchestrator finishes deploying the resources that this service instance consists off.

The auto and api set state call can set the validation attribute to true. When this attribute is true, the orchestration engine refines the model in validation mode. When the validation succeeds the state transfers to the target state, if the validation fails the state transfers to the error state.

On each transfer the lifecycle manager can apply operations to the three attribute sets. These operations can be defined on a transfer to target with target_operation or to error with error_operation. On creation and update the attributes provided through the API are stored in candidate_attributes. For all other transitions the following operations are available:

- `clear <setname>`: Clear the given attribute set. Setname is one of the following: candidate, active or rollback
- `promote`: Promote the values in candidate to active and active to rollback.
- `rollback`: Do a roll back of the attributes by setting the values from rollback to active and active to candidate.

On every state transfer the version of the service instance is incremented.

7.18 Patterns

- Validating with intermediary state
- Batched with on error serial

7.19 Glossary

lifecycle
A formal description of all the states a service instance can be in, between creation and deletion and the possible transfers between the states.

service entity
In the Inmanta lifecycle service manager multiple service entities are registered from an orchestration model. A service entity defines the attributes of a *service instance* and the lifecycle state machine.

service instance
The lifecycle manager manages the lifeycle of service instance.

state
A service instance is always in a state defined in the lifecycle state machine. This state determines how the service instance behaves.

state machine
The lifecycle of a service is modelled as a state machine. This state machine consists of *states* the service can be in and *transfers* from a source state to a destination state.

transfer
A *state* transfer from one state to another. Transfers are used to connect events with a state transfer.

trigger
A trigger is an external event that causes a service instance to transfer to a next state in its lifecycle. A trigger can be an external API call or the orchestrator finishing a deploy of the resources the service consists of.

7.20 Dict Path Library

This extension also uses the *Dict Path library*. This library can be used to extract or modify specific elements from an arbitrary location in a nested dictionary-based data structure.

7.21 Partial Compiles

Partial compiles are an advanced feature that allow increased scaling in the number of services. Instead of triggering compiles for the full model whenever a service instance is created, updated or has a state transfer, only the part of the model relevant for that service instance is recompiled.

LSM expands on the normal *resources set based partial compiles* by automatically creating a single resource set for each service instance. To add resources to the instance's resource set, simply add them to its `owned_resources` relation and make sure to select the `parents` implementation for your service entities. LSM will then make sure to populate the resource set and to correctly trigger related compiles and exports.

For more advanced scenarios, refer to *the lsm partial compile section*.

For a more generic introduction to partial compiles (without lsm), including resource set semantics, modelling guidelines and how to approach testing, refer to the generic *partial compiles* section.

Finally, to enable lsm's partial compiles on the server, set the `lsm_partial_compile` environment setting to true.

CHAPTER

EIGHT

ADMINISTRATOR DOCUMENTATION

8.1 Operational Procedures

This document describes the best practices for various operational procedures.

Note: issue templates for all procedures are available at the bottom of this page

8.1.1 Project Release for Production

This process describes the steps to prepare an inmanta project for production release.

For small projects relying exclusively on public modules and Python dependencies, the default pip config, which pulls packages from `https://pypi.org/` can be used. If the project requires private packages, then, for security reasons, the default pip config, which pulls packages from `https://pypi.org/` should not be used and all packages should be hosted in an internal, curated python repository (like nexus or devpi). See PEP 708 for more information. See the *Configure pip index* section for more information on how to set the project's pip configuration.

Context

- The project development and testing is complete
- All modules have been properly released (See *Releasing and distributing modules* for the procedure).
- The project is in a git repo, with a specific branch dedicated to production releases
- The project is checked out on disk.
- All modules are checked out on the correct, tagged commit.

Procedure

1. **Check modules install mode:**
    - For v1 modules (if any): ensure that `install_mode` in `project.yml` is set to `release`.
    - For v2 modules (if any): ensure that `pip.pre` is not set in `project.yml`.
2. Freeze all modules with:

```
inmanta -vv -X project freeze --recursive --operator "=="
```

This will cause the `project.yml` file to be updated with constraints that only allow this project to work with this exact set of module versions. This ensures that no unwanted updates can 'leak' into the production environment.

3. Verify that all modules are frozen to the correct version.

    Open `project.yml` and verify that all module versions are frozen to the expected versions

4. Commit this change.

```
git commit -a
```

5. Push to the release branch.

```
git push
```

8.1.2 Upgrade of service model on the orchestrator

This process describes how to safely take an existing project from one version to the next.

Context

- The orchestrator has the project already deployed and running
- The project is released (as described above)

Pre-Upgrade steps

1. Verify that environment safety settings are on (this should always be the case)
   - *protected_environment = True*
2. Temporarily disable auto_deploy
   - *auto_deploy = False*
3. Click 'recompile' to verify that no new deploy would start.
   - A new version will appear but it will not start to deploy
4. Inspect the current state of the latest deployed version, verify no failures are happening and the deploy looks healthy
5. (Optional) Perform a dryrun. Wait for the dryrun to complete and take note of all changes detected by the dryrun. Ideally there should be none.

Upgrade procedure

1. Click *Update project & recompile*
   - A new version will appear but it will not start to deploy
2. Click *Perform dry run* on the new version
   - The dryrun report will open
   - Wait for the dryrun to finish
   - Inspect any changes found by the dryrun, determine if they are expected. If unexpected things are present, go to the abort procedure.
3. If all is OK, click deploy to make the changes effective

Post Upgrade procedure

1. Re-enable auto_deploy
    - *auto_deploy = True*

Upgrade abort/revert

1. Delete the bad (latest) version
2. Push a revert commit onto the release branch (*git revert HEAD; git push*)
3. Go through the Upgrade procedure again to make this revert effective

8.1.3 Deployment of a new service model to the orchestrator

This process describes how to safely deploy a new model to the orchestrator.

Context

- The orchestrator has an environment set up for the project, but it has not been deployed yet.
- The project is released (as described above)

Procedure

1. Cross check all settings in the environment settings tab with the development team.
2. Verify that environment safety settings are on (should always be the case)
    - *protected_environment = True*
3. Temporarily disable auto_deploy

- *auto_deploy = False*

4. Click 'recompile' to install the project.

- A new version will appear but it will not start to deploy
- This may take a while as the project has to be installed.
- In case of problems, consult the Compile Reports

5. Verify that the resources in this first version are as expected.
6. Click deploy to make the changes effective

- Keep a close eye on progress and problems that may arise.
- In case of trouble, hit the emergency stop. Resuming after a stop is very easy and stopping gives you the time to investigate.

7. Verify that automation setting are on

- *agent_trigger_method_on_auto_deploy = push_incremental_deploy*
- *auto_deploy = true*
- *push_on_auto_deploy = true*
- *server_compile = true*

8. If this model uses LSM, perform initial tests of all services via the API.

Extra careful deploy procedure

For models that are considered risky, it is possible to enable the model in a more gradual way. The general idea is to disengage all features on the orchestrator that make the agents perform unsupervised deployments. Then the agents can be activated by hand, one-by-one.

This procedure only works when all agents are autostarted by the server.

1. Take note of the following settings
   - *autostart_agent_deploy_interval*
   - *autostart_agent_repair_interval*
2. Disable spontaneous deployment
   - *autostart_agent_deploy_interval = 0*
   - *autostart_agent_repair_interval = 0*
   - *auto_deploy = True*
   - *push_on_auto_deploy = False*
3. Click 'recompile' to install the project.
   - A new version will appear
   - It will go to the deploying state
   - But no resources will be deployed
4. In the agent tab, click *deploy on agent* on the 'internal' agent. Press *force repair* in the dropdown menu.
   - All agents will come online
5. Perform a dryrun, to verify there are no undesirable effects.
6. Click *deploy on agent/force repair* on each agent. Verify results.
7. Ensure all environment setting are set correctly
   - *agent_trigger_method_on_auto_deploy = push_incremental_deploy*
   - *auto_deploy = true*
   - *push_on_auto_deploy = true*
   - *server_compile = true*
   - *autostart_agent_deploy_interval*
   - *autostart_agent_repair_interval*

8.1.4 Issue templates

For convenient inclusion in issue tickets, this section provides ready made markdown templates.

Project Release for Production

```
* [ ] Verify in `project.yml` that `install_mode` is set to `release`.
* [ ] Freeze all modules with `inmanta -vv -X project freeze --recursive --operator
↪"=="`
* [ ] Verify that all modules are frozen to the correct version
* [ ] Commit this change (`git commit -a`)
* [ ] Push to the release branch (`git push`)
```

Upgrade of service model on the orchestrator

```
* Pre-Upgrade steps:

1. Verify that environment safety settings are on (this should always be the case)

    * [ ] `protected_environment = True`

2. Temporarily disable auto_deploy

   * [ ] `auto_deploy = False`

3. [ ] Click ‘recompile’ to verify that no new deploy would start.

    * A new version will appear but it will not start to deploy

4. [ ] Inspect the current state of the latest active version, verify no failures are␣
↪happening and the deploy looks healthy
5. [ ] (Optional) Perform a dryrun. Wait for the dryrun to complete and take note of␣
↪all changes detected by the dryrun. Ideally there should be none.

* Upgrade procedure

1. [ ] Click `Update and recompile`

   * A new version will appear but it will not start to deploy

2. [ ] Click dryrun on the new version

   * The dryrun report will open
   * Wait for the dryrun to finish
   * [ ] Inspect any changes found by the dryrun, determine if they are expected. If␣
↪unexpected things are present, go to the abort procedure.
3. [ ] If all is OK, click deploy to make the changes effective

* Post Upgrade procedure

1. Re-enable auto_deploy

    * [ ] `auto_deploy = True`

* Upgrade abort/revert

1. [ ] Delete the bad (latest) version
2. [ ] Push a revert commit onto the release branch (`git commit revert HEAD; git␣
↪push`)
3. [ ] Click `Update and recompile`
```

(continues on next page)

(continued from previous page)

```
   * A new version will appear but it will not start to deploy

4. [ ] Click dryrun on the new version

   * The dryrun report will open
   * Wait for the dryrun to finish
   * [ ] Inspect any changes found by the dryrun, this should be identical to the␣
↪dryrun before the upgrade. If this is not the case, hit the emergency stop button␣
↪and and contact support.
```

8.2 Diagnosing problems

When an unexpected problem arises with an inmanta environment, you might want to work directly on the environment on the orchestrator host to diagnose it. The `inmanta-workon` command, installed by the RPM, provides that functionality. `inmanta-workon myenvironment` puts you in the environment's project directory and activates its Python venv. If you don't know the name of the environment by heart, `inmanta-workon --list` gives an overview of all environments on the server.

For more details, see `inmanta-workon --help`.

Note: If you didn't install inmanta from RPM, you can manually source the `inmanta-workon-register.sh` script to get access to the `inmanta-workon` command. You can find the script in the `misc` directory in the `inmanta-core` git repository.

8.3 Configuration

Note: The documentation of the configuration options themselves can be found in the *Inmanta configuration reference*.

8.3.1 Inmanta server and Inmanta agent

The Inmanta server and the Inmanta agent, started via systemd, will read their configuration from the following locations:

1. `/etc/inmanta/inmanta.cfg`
2. `/etc/inmanta/inmanta.d/*.cfg`
3. `environment variables`

The configuration options specified in the `/etc/inmanta/inmanta.d/` directory override the configuration options specified in `/etc/inmanta/inmanta.cfg`. If the directory `/etc/inmanta/inmanta.d/` contains two files with the same configuration option, the conflict is resolved using the alphabetical order of the filenames. Filenames which appear later in the alfabetical order override the configuration options from their predecessors in that order.

After having read the configuration files, inmanta will read environment variables. The environment variables overwrite any other types of configuration, if set. All settings can be set using environment variables with the following convention:

`INMANTA_{section.name}_{setting.name}`

Keep in mind that everything should be in ALL CAPS and that any dashes in the setting names must be replaced by underscores.

8.3.2 Inmanta CLI tool

The `inmanta` CLI tool reads its configuration at the following locations:

1. `/etc/inmanta/inmanta.cfg`
2. `/etc/inmanta/inmanta.d/*.cfg` (override using the `--config-dir` option)
3. `~/.inmanta.cfg`
4. `.inmanta`
5. `.inmanta.cfg`
6. The config file specified on the CLI using the `-c` options
7. Environment variables

The `inmanta` CLI tool searches for the `.inmanta` and `.inmanta.cfg` files in the directory where the CLI command is executed.

Configuration files which are ranked lower in the above-mentioned list override the configuration options specified by their predecessors. If the directory `/etc/inmanta/inmanta.d/` contains two files with the same configuration option, the conflict is resolved using the alphabetical order of the filenames. Filenames which appear later in the alphabetical order override the configuration options from their predecessors in that order.

The number 2 (`/etc/inmanta/inmanta.d/*.cfg`) in the above-mentioned list can be overridden using the `--config-dir` option of the `inmanta` command. More information about these options can be found in the *inmanta command reference*

8.4 HA setup

This page describes how to deploy an Inmanta server in a HA setup and how to perform a failover when required.

8.4.1 Setup a HA PostgreSQL cluster

The Inmanta server stores its state in a PostgreSQL database. As such, the PostgreSQL database should be deployed in a high available setup, to ensure the durability of the state of the Inmanta Orchestrator. This page describes how to setup a two node PosgreSQL cluster, consisting of a master node and a warm standby. The master node performs synchronous replication to the standby node. When the master node fails, the standby can be promoted to the new master node by performing a manual action.

This setup has a number of properties:

- It ensure durability by only returning operations like API calls when both database instances has confirmed that the changes have been stored on disk.
- It is possible to use a tool such as pgpool to loadbalance read-only database queries to the standby node. However, this is out of scope of this manual.
- It does not provide any additionaly availability, it even slighly reduces it: both database servers need to be up and responsive to process write queries. If the standby node is down, the master node will block on any write query. Read queries continue to be served until the database pool is exhausted.

For almost all types of deployments it provides a good trade-off between setup and operational complexity and the availability and durability guarantees. If both durability and higher availability are required, a setup with at least 3 databases is required. This is out of scope for this documentation. Please contact support for assistance on this topic.

Prerequisites

- **Master node:** The master node has been setup according to *step 2* and *step 3* of the Inmanta installation documentation.
- **Standby node:** The standby node should only have a PostgreSQL installation, so only *step 2* of the Inmanta installation documentation should be executed.

Configure the master node

Login on the master node and perform the following changes in the `/var/lib/pgsql/data/postgresql.conf` file:

```
# Adjust the listen address as such that the standby node
# can connect to the master node.
listen_addresses = '*'

# Increase the wal_level to the required level for data replication
wal_level = replica

# Only report success to the client when the transaction has been
# flushed to permanent storage
synchronous_commit = on

# Force synchronous replication to the standby node. The application_name
# uniquely identifies the standby instance and can be freely chosen as long
# as it only consists of printable ASCII characters.
synchronous_standby_names = 'inmanta'

# Make sure that no queries can be executed on the standby
# node while it is in recovery mode.
hot_standby = off
```

Execute the commands mentioned below on the master node. These commands do two thing:

- They create a replication user with replication and login privileges. The standby node will use this user to connect to the master node.
- They create a new replication slot, named *replication*. This replication slot will make sure that sufficient data is retained on the master node to synchronize the standby node with the master node.

```
$ sudo su - postgres -c 'psql'
$ CREATE USER replication WITH REPLICATION LOGIN PASSWORD '<password-replication-user>
↪';
$ SELECT * FROM pg_create_physical_replication_slot('replication');
$ \q
```

Add the lines mentioned below to the `/var/lib/pgsql/data/pg_hba.conf` file. This will make sure that the replication user can be used to setup a replication connection from the standby node to the master. Since, the standby node can become the master node, both hosts should be add to the file.

```
host    replication     replication      <ip-master-node>/32        md5
host    replication     replication      <ip-standby-node>/32       md5
```

Restart the `postgresql` service to activate the configuration changes.

```
$ sudo systemctl restart postgresql
```

Configure the standby node

The standby gets configured by creating a backup of the master node and restoring it on the standby node. The commands mentioned below create a backup in the /tmp/backup directory. This command will prompt for the password of the replication user. By setting the -R option, a standby.signal and a postgresql.auto.conf file will be added to the backup. The presence of the former will make the PostgreSQL server start as a standby. The latter contains replication-specific configuration settings. Those will be processed after the postgresql.conf file is processed.

```
$ sudo su - postgres -c 'pg_basebackup -h <ip-master-node> -U replication -X stream -
↪R -D /tmp/backup -S replication -W'
```

On the standby node, clear the content of the /var/lib/pgsql/data directory and replace it with the content of the backup created on the master node. The postgresql.auto.conf file needs to be adjusted as such that it has the application_name parameter in the primary_conninfo setting. This application_name should match the name configured in the synchronous_standby_names setting of the postgresql.conf file of the master node.

```
primary_conninfo = 'user=replication password=<password> channel_binding=prefer host=
↪<password> port=5432 sslmode=prefer sslcompression=0 ssl_min_protocol_version=TLSv1.
↪2 gssencmode=prefer krbsrvname=postgres target_session_attrs=any application_
↪name=inmanta'
primary_slot_name = 'replication'
```

Comment out, the synchronous_standby_names setting in the postgresql.conf file of the standby node. This will ensure that the standby node acts fully independently when it is promoted to a master node. Finally, start and enable the PostgreSQL service on the standby node.

```
$ sudo systemctl start postgresql
$ sudo systemctl enable postgresql
```

Monitoring

This setup requires both database to be up to be up and functional. It is highly recommended to monitor this the availability of the database and the replication status. For most monitoring systems (such as nagios/icinga or promotheus/alertmanager) there are plugins avilable to do this in an efficient manner.

8.4.2 Failover PostgreSQL

This section describes the action required to recover from a failed PostgreSQL master node.

Promote a standby node to the new master node

When the master node fails, the standby node can be promoted to become the new master node. After this failover, the new master will acts as a fully independent instance, i.e. no replication will happen to a standby instance.

Execute the following command on the standby instance to promote it to a new master node:

```
$ sudo su - postgres -c 'pg_ctl promote -D /var/lib/pgsql/data/'
```

This command will remove the standby.signal file. It's also recommended to cleanup the postgresql.auto.conf file by executing the following commands:

```
$ sudo rm -f /var/lib/pgsql/data/postgresql.auto.conf
$ sudo systemctl reload postgresql
```

The old master node can be reconfigured to become the new standby node, by executing the step described in the next section.

Add a standby node to a newly promoted master node

This section explains how a standby can be add to a master node, which was created from a promoted standby node.

First, add a replication slot on the new master node by executing following commands:

```
$ sudo su - postgres -c 'psql'
$ SELECT * FROM pg_create_physical_replication_slot('replication');
$ \q
```

Then, configure the new standby instance by following the step mentioned in *Configure the standby node*. When the standby is up, the master node perform asynchronous replication to the standby node. The master node needs to be reconfigured to perform synchronous replication. This is done by adding the line mentioned below the `postgresql.conf` file of the master node. The `application_name` has to match the `application_name` set in the `postgresql.auto.conf` file of the standby node.

```
synchronous_standby_names = 'inmanta'
```

Finally, reload the configuration of the master node using the following command:

```
$ sudo systemctl reload postgresql
```

8.4.3 Failover an Inmanta server

This section describes different ways to failover an Inmanta server.

Failover an Inmanta server to the warm standby PostgreSQL instance

This section describes how to failover an Inmanta server to a new PostgreSQL master node when the previous master node has failed.

First, stop the orchestrator by stopping the `inmanta-server` service.

```
$ sudo systemctl stop inmanta-server
```

Promote the standby node to a master node by following the procedure mentioned in Section *Promote a standby node to the new master node*. When the promotion is finished, the Inmanta server can be reconfigured to start using the new master node. Do this by adjusting `database.host` setting the `/etc/inmanta/inmanta.d/database.cfg` file:

```
[database]
host=<ip-address-new-master-node>
name=inmanta
username=inmanta
password=<password>
```

Now, start the Inmanta orchestrator again:

```
$ sudo systemctl start inmanta-server
```

Start a new orchestrator on warm standby PostgreSQL instance

This section describes what should be done to recover when the Inmanta server and the PostgreSQL master node fail simultaneously. It is also possible to failover the Inmanta server when the PostgreSQL master node has not failed.

Before starting the failover process, it's important to ensure that the original Inmanta server is fully disabled. This is required to prevent the situation where two orchestrators are performing configuration changes on the same infrastructure simultaneously. Disabling the Inmanta orchestrator can be done by stopping the machine running the Inmanta server or disabling the `inmanta-server` service using the following commands:

```
$ sudo systemctl stop inmanta-server
$ sudo systemctl disable inmanta-server
```

The following step should only be executed when the PostgreSQL master node has failed.

Next, promote the standby PostgreSQL node to the new master node using the procedure in Section *Promote a standby node to the new master node*. When the (new) master node is up, a new Inmanta server can be installed according the procedure mention in the *Install Inmanta* section. In the `/etc/inmanta/inmanta.d/database.cfg` configuration file, the `database.host` setting should contain the IP address of the new PostgreSQL master node.

When the Inmanta server is up and running, a recompile should be done for each existing configuration model.

8.5 Operational Procedures With LSM

This document describes the best practices for various operational procedures, when using Lifecycle and Service Management. These procedures are an extension to the ones described in *Operational Procedures*.

Note: issue templates for all procedures are available at the bottom of this page

8.5.1 Upgrade of service model on the orchestrator

This process describes how to safely take an existing project from one version to the next.

Context

- The orchestrator has the project already deployed and running
- The project is released (as described here: *Project Release for Production*)

Pre-Upgrade steps

1. Determine if this update in any way affects the service definition. If the update doesn't change the lifecycle or any aspect of the schema of the north bound api, flow the procedure here: *Upgrade of service model on the orchestrator*. Otherwise, go to the next step.
2. Determine if the update changes the structure of existing instances in the service inventory (i.e. add or remove fields). If this is the case, a database backup is required to revert the update.
3. Ensure you have shell access to the orchestrator.
4. Verify with the development team what the correct entry point is for exporting the API definition. Exporting the api definition requires the model to compile. Often, the model requires a correct API definition to compile. To break this cycle, developers have to provide an alternative entry point into the code (other than *main.cf*) that loads only the definitions. We will assume this file is called *loader.cf*

5. Verify that environment safety settings are on (this should always be the case)
   - *protected_environment = True*
6. Temporarily disable auto_deploy
   - *auto_deploy = False*
4. Click 'recompile' to verify that no new deploy would start.
   - A new version will appear but it will not start to deploy
5. Inspect the current state of the latest deployed version, verify no failures are happening and the deploy looks healthy
6. (Optional) Perform a dryrun. Wait for the dryrun to complete and take note of all changes detected by the dryrun. Ideally there should be none.
7. Block out all north-bound api calls (in the north-bound load balancer or firewall). This is to prevent instance changes during the update.
8. Pause all agents, to prevent state transitions during the update.
9. Backup the database (if required as described in step 2). A full backup is preferable (using eg *pgdump*). The tables *lsm_serviceentity* and *lsm_serviceinstance* are most crucial. A schema update may cause the instances in the database to be irreversible rewritten. This backup will ensure a way back.

Upgrade procedure

1. Instruct the orchestrator to pull in the latest version by clicking *Update project & recompile*

- The compiler pulls in the latest version
- This compile may fail or produce a new version, that will not start to deploy

2. Send the new service definition to the server: Log onto the orchestrator and navigate to the folder for this environment.

If no instance updates are expected use:

```
inmanta-workon $envid
inmanta -vvv -X export -j /dev/null -e $envid -f loader.cf --export-plugin service_
↪entities_exporter_strict
```

If instance update are expected (and you made a database backup), use:

```
inmanta-workon $envid
inmanta -vvv -X export -j /dev/null -e $envid -f loader.cf --export-plugin service_
↪entities_exporter
```

3. Click *Recompile*
   - The compiler will produce a new version, that will not start to deploy
4. Re-enable the agents.
5. Click *Perform dry run* on the new version
   - The dryrun report will open
   - Wait for the dryrun to finish
   - Inspect any changes found by the dryrun, determine if they are expected. If unexpected things are present, go to the abort procedure.
4. If all is OK, click deploy to make the changes effective

Post Upgrade procedure

1. Re-enable auto_deploy
    - *auto_deploy = True*
2. Allow requests to be sent to the north bound api again

Upgrade abort/revert

1. Delete the bad (latest) version produced during the update in the web-console
2. Push a revert commit onto the release branch (*git revert HEAD; git push*)
3. Go through the Upgrade procedure again to make this revert effective
4. If the API update is irreversible or the end-result after revert is different from the expected result, restore the database tables *lsm_serviceentity* and *lsm_serviceinstance*.

8.5.2 Deployment of a new service model to the orchestrator

This process describes how to safely deploy a new model to the orchestrator.

Context

- The orchestrator has an environment set up for the project, but it has not been deployed yet.
- The project is released (as described above)

Procedure

1. Cross check all settings in the environment settings tab with the development team.
2. Verify with the development team what the correct entry point is for exporting the API definition. Exporting the api definition requires the model to compile. Often, the model requires a correct API definition to compile. To break this cycle, developers have to provide an alternative entry point into the code (other than *main.cf*) that loads only the definitions. We will assume this file is called *loader.cf*
3. Verify that environment safety settings are on (should always be the case)
    - *protected_environment = True*
4. Temporarily disable auto_deploy
    - *auto_deploy = False*
5. Click 'recompile' to install the project.
    - To check if the compile is done, check the *Compile Reports*
    - A new version may appear but it will not start to deploy
    - This may take a while as the project has to be installed.

6. Send the new service definition to the server: Log onto the orchestrator and navigate to the folder for this environment.

If no instance updates are expected use:

```
inmanta-workon $envid
inmanta -vvv -X export -j /dev/null -e $envid -f loader.cf --export-plugin service_
↪entities_exporter_strict
```

1. Click *Recompile*

- The compiler will produce a new version, that will not start to deploy

2. Verify that the resources in this first version are as expected.
3. Click deploy to make the changes effective
   - Keep a close eye on progress and problems that may arise.
   - In case of trouble, hit the emergency stop. Resuming after a stop is very easy and stopping gives you time to investigate.
7. Verify that automation setting are on
   - *agent_trigger_method_on_auto_deploy = push_incremental_deploy*
   - *auto_deploy = true*
   - *push_on_auto_deploy = true*
   - *server_compile = true*
8. Perform initial tests of all services via the API.

8.5.3 Issue templates

For convenient inclusion in issue tickets, this section provides ready made markdown templates.

Upgrade of service model on the orchestrator

```
# Required Information:

  * [ ] The entry point for exporting the API definition is:
  * [ ] The environment id is:

# Pre-Upgrade steps:

1. Verify that environment safety settings are on (this should always be the case)

   * [ ] `protected_environment = True`

2. Temporarily disable auto_deploy

   * [ ] `auto_deploy = False`

3. [ ] Click 'recompile' to verify that no new deploy would start.

   * A new version will appear but it will not start to deploy

4. [ ] Inspect the current state of the latest active version, verify no failures are␣
↪happening and the deploy looks healthy
5. [ ] (Optional) Perform a dryrun. Wait for the dryrun to complete and take note of␣
↪all changes detected by the dryrun. Ideally there should be none.
6. [ ] Block out all north-bound api calls
7. [ ] Pause all agents
8. [ ] Backup the database `pgdump`

# Upgrade procedure

1. [ ] Click `Update and recompile`
```

(continues on next page)

(continued from previous page)

~~~
    * A new version will appear but it will not start to deploy

2. [ ] Send the new service definition to the server:
    Log onto the orchestrator and navigate to the folder for this environment.
    ```sh
    cd /var/lib/inmanta/$envid/
    inmanta -vvv -X export -j /tmp/dump.json -e $envid -f loader.cf --export-plugin␣
↪service_entities_exporter
    ~~~
3. [ ] Click `Recompile`
4. [ ] Re-enable the agents

5. [ ] Click dryrun on the new version

    * The dryrun report will open
    * Wait for the dryrun to finish
    * [ ] Inspect any changes found by the dryrun, determine if they are expected. If␣
↪unexpected things are present, go to the abort procedure.
6. [ ] If all is OK, click deploy to make the changes effective

# Post Upgrade procedure

1. Re-enable auto_deploy

    * [ ] `auto_deploy = True`

2. Allow requests to be sent to the north bound api again

# Upgrade abort/revert

1. [ ] Delete the bad (latest) version
2. [ ] Push a revert commit onto the release branch (`git commit revert HEAD; git␣
↪push`)
3. [ ] Click `Update and recompile`
4. [ ] Send the old service definition to the server:
    Log onto the orchestrator and navigate to the folder for this environment.
    ```sh
    cd /var/lib/inmanta/$envid/
    inmanta -vvv -X export -j /tmp/dump.json -e $envid -f loader.cf --export-plugin␣
↪service_entities_exporter
    ```
3. [ ] Click `Recompile`
4. [ ] Re-enable the agents
5. [ ] Click dryrun on the new version

    * The dryrun report will open
    * Wait for the dryrun to finish
    * [ ] Inspect any changes found by the dryrun, this should be identical to the␣
↪dryrun before the upgrade. If this is not the case, hit the emergency stop button␣
↪and and contact support.
6. If the API update is irreversible or the end-result after revert is different from␣
↪the expected result, restore the database tables `lsm_serviceentity` and `lsm_
↪serviceinstance`.
```
```

Install of service model on the orchestrator

~~~
# Required Information:

  * [ ] The entry point for exporting the API definition is:
  * [ ] The environment id is:

# Pre-Upgrade steps:

1. Verify that environment safety settings are on (this should always be the case)

   * [ ] `protected_environment = True`

2. Temporarily disable auto_deploy

   * [ ] `auto_deploy = False`

3. [ ] Click 'recompile' to install the project.
4. [ ] Send the new service definition to the server:
    Log onto the orchestrator and navigate to the folder for this environment.
    ```sh
    cd /var/lib/inmanta/$envid/
    inmanta -vvv -X export -j /tmp/dump.json -e $envid -f loader.cf --export-plugin␣
↪service_entities_exporter
    ```
5. [ ] Click `Recompile`
6. [ ] Verify that the resources in this first version are as expected.

7. [ ] Click deploy to make the changes effective
8. [ ] Monitor progress
9. [ ] Verify that automation setting are on

    * `agent_trigger_method_on_auto_deploy = push_incremental_deploy`
    * `auto_deploy = true`
    * `push_on_auto_deploy = true`
    * `server_compile = true`


10. Perform initial tests of all services via the API.
~~~

8.6 Setting up SSL and authentication

This guide explains how to enable ssl and setup authentication.

8.6.1 SSL

This section explain how to setup SSL. SSL is not strictly required for authentication but it is highly recommended. Inmanta uses bearer tokens to authorize users and services. These tokens should be kept private and are visible in plain-text in the request headers without SSL.

SSL: server side

Setting a private key and a public key in the server configuration enables SSL on the server. The two options to set are *server.ssl-cert-file* and *server.ssl-key-file*.

For the autostarted agents and compiler to work, either add the CA cert to the trusted certificates of the system or set *server.ssl-ca-cert-file* to the truststore.

```
[server]
# The ssl certificate used by the server
ssl_cert_file=/etc/inmanta/server.crt
# The private key used by the server, associated with the certificate
ssl_key_file=/etc/inmanta/server.key.open

# The certificate chain that the compiler and agents should use to validate the␣
↪server certificate
ssl_ca_cert_file=/etc/inmanta/server.chain
# The address at which the compiler and agent should connect
# Must correspond to hostname the ssl certificate is bound to
server_address=localhost
```

SSL: agents and compiler

When using SSL, all remote components connecting to the server need to have SSL enabled as well.

For each of the transport configurations (compiler, agent, rpc client, ...) `ssl` has to be enabled: *agent_rest_transport*, *cmdline_rest_transport* and *compiler_rest_transport*.

The client needs to trust the SSL certificate of the server. When a self-signed SSL cert is used on the server, either add the CA cert to the trusted certificates of the system running the agent or configure the `ssl-ca-cert-file` option in the transport configuration.

For example for an agent this is *agent_rest_transport.ssl* and *agent_rest_transport.ssl-ca-cert-file*

Autostarted agents and compiles on the server also use SSL to communicate with the server. This requires either for the server SSL certificate to be trusted by the OS or by setting *server.ssl-ca-cert-file*. The server will use this value to set *compiler_rest_transport.ssl-ca-cert-file* and *server.ssl-ca-cert-file* for the compiler and the agents.

8.6.2 Authentication

Inmanta authentication uses JSON Web Tokens for authentication (bearer token). Inmanta issues tokens for service to service interaction (agent to server, compiler to server, cli to server and 3rd party API interactions). For user interaction through the web-console Inmanta can rely on its built-in authentication provider or on a 3rd party auth broker. Currently the web-console only supports Keycloak as 3rd party auth broker.

Inmanta expects a token of which it can validate the signature. Inmanta can verify both symmetric signatures with HS256 and asymmetric signatures with RSA (RS256). Tokens it signs itself for other processes are always signed using HS256. There are no key distribution issues because the server is both the signing and the validating party.

The server also provides limited authorization by checking for inmanta specific claims inside the token. All inmanta claims are prefixed with `urn:inmanta:`. These claims are:

- `urn:inmanta:ct` A *required* comma delimited list of client types for which this client is authenticated. Each API call has one or more allowed client types. The list of valid client types (ct) are:
  - agent
  - compiler
  - api (cli, web-console, 3rd party service)

- `urn:inmanta:env` An *optional* claim. When this claim is present the token is scoped to this inmanta environment. All tokens that the server generates for agents and compilers have this claim present to limit their access to the environment they belong to.

Setup server auth

The server requests authentication for all API calls when `server.auth` is set to true. When authentication is enabled all other components require a valid token.

Warning: When multiple servers are used in a HA setup, each server requires the same configuration (SSL enabled and private keys).

In the server configuration multiple token providers (issuers) can be configured (See *JWT auth configuration*). Inmanta requires at least one issuer with the HS256 algorithm. The server uses this to sign tokens it issues itself. This provider is indicated with sign set to true. Inmanta issues tokens for compilers the servers runs itself and for autostarted agents.

Compilers, cli and agents that are not started by the server itself, require a token in their transport configuration. This token is configured with the `token` option in the groups `agent_rest_transport`, `cmdline_rest_transport` and `compiler_rest_transport`.

A token can be retrieved either with `inmanta-cli token create` or via the web-console using the `tokens` tab on the settings page.

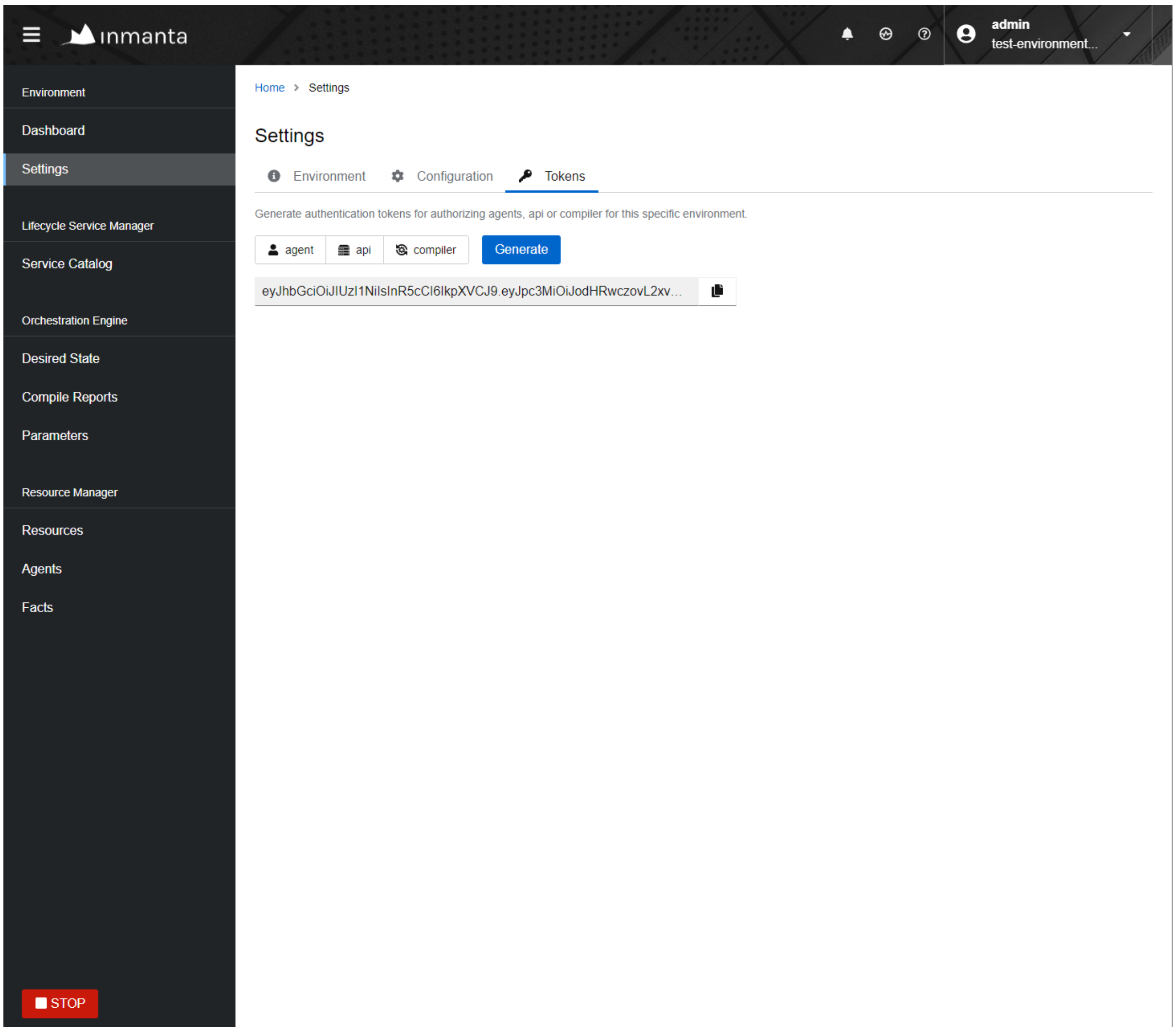
Setup the built-in authentication provider of the Inmanta server (See *Built-in authentication provider*) or configure an external issuer (See *External authentication providers*) for web-console access to bootstrap access to the create token api call. When no external issuer is available and web-console access is not required, the `inmanta-cli token bootstrap` command can be used to create a token that has access to everything. However, it expires after 3600s for security reasons.

For this command to function, it requires the issuers configuration with sign=true to be available for the cli command.

JWT auth configuration

The server searches for configuration sections that start with `auth_jwt_`, after the last _ an id has to be present. This section expects the following keys:

- algorithm: The algorithm used for this key. Only HS256 and RS256 are supported.
- sign: Whether the server can use this key to sign JWT it issues. Only one section may have this set to true.
- client_types: The client types from the `urn:inmanta:ct` claim that can be validated and/or signed with this key.
- key: The secret key used by symmetric algorithms such as HS256. Generate the key with a secure prng with minimal length equal to the length of the HMAC (For HS256 == 256). The key should be a urlsafe base64 encoded bytestring without padding. (see below of a command to generate such a key)
- expire: The default expire for tokens issued with this key (when sign = true). Use 0 for tokens that do not expire.
- issuer: The url of the issuer that should match for tokens to be valid (also used to sign this). The default value is https://localhost:8888/ This value is used to match auth_jwt_* sections configuration with JWT tokens. Make sure this is unique.
- audience: The audience for tokens, as per RFC this should match or the token is rejected.
- jwks_uri: The uri to the public key information. This is required for algorithm RS256. The keys are loaded the first time a token needs to be verified after a server restart. There is not key refresh mechanism.



Settings

Generate authentication tokens for authorizing agents, api or compiler for this specific environment.

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJodHRwczovL2xv...

Fig. 1: Generating a new token in the web-console.

- jwks_request_timeout: The timeout for the request to the 'jwks_uri', in seconds. If not provided, the default value of 30 seconds will be used.

An example configuration is:

```
[auth_jwt_default]
algorithm=HS256
sign=true
client_types=agent,compiler
key=rID3kG4OwGpajIsxnGDhat4UFcMkyFZQc1y3oKQTPRs
expire=0
issuer=https://localhost:8888/
audience=https://localhost:8888/
```

To generate a secure symmetric key and encode it correctly use the following command:

```
openssl rand 32 | python3 -c "import sys; import base64; print(base64.urlsafe_
↪b64encode(sys.stdin.buffer.read()).decode().rstrip('='));"
```

8.6.3 Built-in authentication provider

The Inmanta server has a built-in authentication provider. This provider stores the authentication and authorization information into the PostgreSQL database. As such, there is no need to rely on a 3rd party auth broker. The sections below describe how to enable the built-in authentication provider and how to create the initial admin user. Additional users can then be created via the API or through the web console.

Step 1: Enable authentication

Ensure that the `server.auth` configuration option is enabled and that the `server.auth-method` configuration option is set to `database`. This means that the `/etc/inmanta/inmanta.d/server.cfg` file should contains the following:

```
[server]
auth=true
auth-method=database
...
```

Step 2: Generate the JWT configuration

Run the `/opt/inmanta/bin/inmanta-initial-user-setup` command on the orchestrator server. This command will output a generated JWT configuration if no JWT configuration is already in-place on the server.

```
$ /opt/inmanta/bin/inmanta-initial-user-setup
This command should be execute locally on the orchestrator you want to configure. Are␣
↪you running this command locally? [y/N]: y
Server authentication:                           enabled
Server authentication method:                    database
Error: No signing config available in the configuration.
To use a new config, add the following to the configuration in /etc/inmanta/inmanta.d/
↪auth.cfg:

[auth_jwt_default]
algorithm=HS256
sign=true
client_types=agent,compiler,api
```

(continues on next page)

(continued from previous page)

```
key=NYR2LtAsKSs7TuY0D8ZIqmMaLcICC3lf_ur4FGlLUcQ
expire=0
issuer=https://localhost:8888/
audience=https://localhost:8888/

Error: Make sure signing configuration is added to the config. See the documentation␣
↪for details.
```

Verify whether the hostname, in the generated configuration section, is correct and put the configuration snippet in the location mentioned in the output of the command.

Step 3: Create the initial user

Re-run the same command again to create the initial user. The password for this new user must be at least 8 characters long.

```
$ /opt/inmanta/bin/inmanta-initial-user-setup
This command should be execute locally on the orchestrator you want to configure. Are␣
↪you running this command locally? [y/N]: y
Server authentication:                            enabled
Server authentication method:                     database
Authentication signing config:                    found
Trying to connect to DB:                          inmanta (localhost:5432)
Connection to database                            success
What username do you want to use? [admin]:
What password do you want to use?:
User admin:                                       created
Make sure to (re)start the orchestrator to activate all changes.
```

Step 4: Restart the orchestrator

Now, restart the orchestrator to activate the new configuration.

```
$ sudo systemctl restart inmanta-server
```

After the restart of the orchestrator, authentication is enabled on all API endpoints. This also means that the web-console will ask for your credentials.

8.6.4 External authentication providers

Inmanta supports all external authentication providers that support JWT tokens with RS256 or HS256. These providers need to add a claims that indicate the allowed client type (`urn:inmanta:ct`). Currently, the web-console only has support for keycloak. However, each provider that can insert custom (private) claims should work. The web-console now relies on the keycloak js library to implement the OAuth2 implicit flow, required to obtain a JWT.

Tip: All patches to support additional providers such as Auth0 are welcome. Alternatively contact Inmanta NV for custom integration services.

Keycloak configuration

The web-console has out of the box support for authentication with Keycloak. Install keycloak and create an initial login as described in the Keycloak documentation and login with admin credentials.

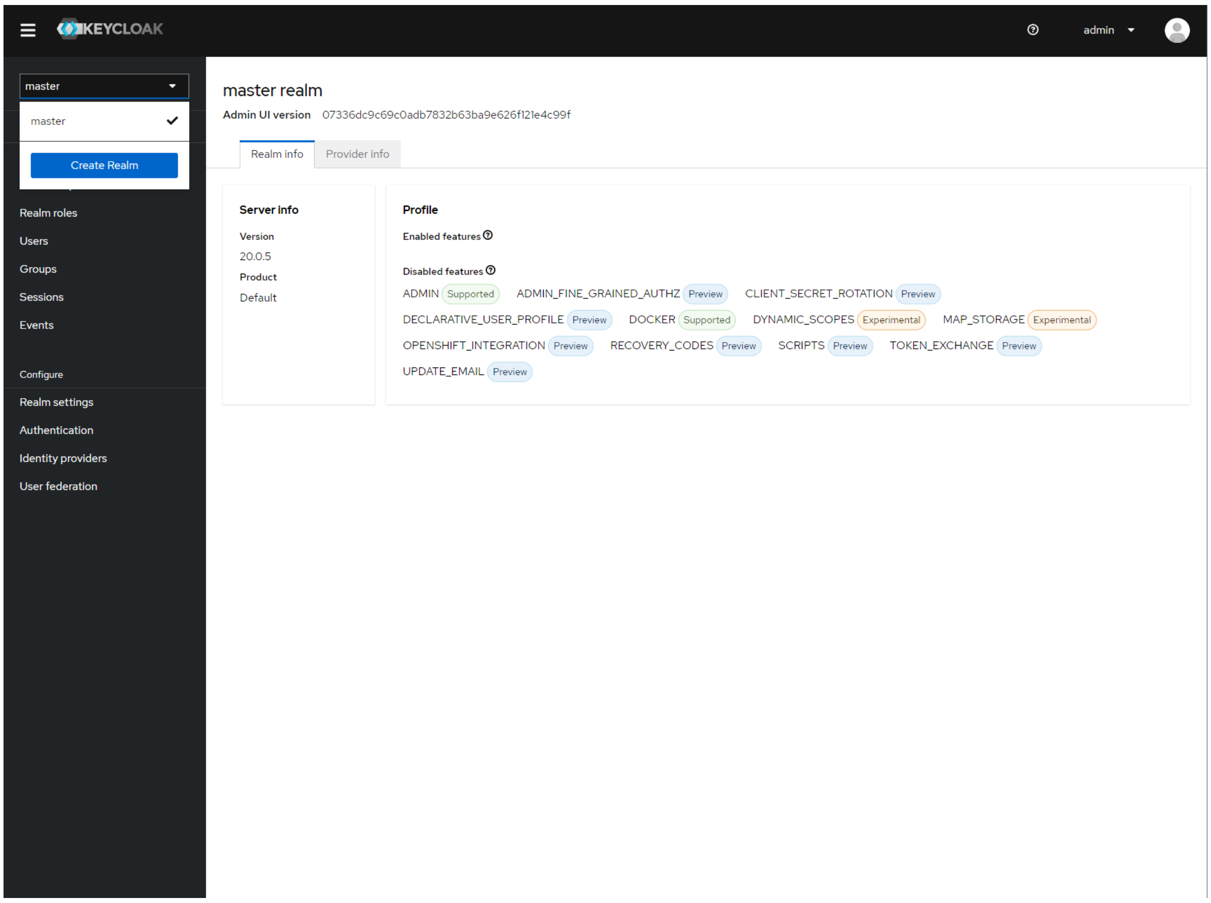
This guide was made based on Keycloak 20.0

If inmanta is configured to use SSL, the authentication provider should also use SSL. Otherwise, the web-console will not be able to fetch user information from the authentication provider.

Step 1: Optionally create a new realm

Create a new realm if you want to use keycloak for other purposes (it is an SSO solution) than Inmanta authentication. Another reason to create a new realm (or not) is that the master realm also provides the credentials to configure keycloak itself.

For example call the realm inmanta



master realm

Admin UI version 07336dc9c69c0adb7832b63ba9e626f121e4c99f

Server info

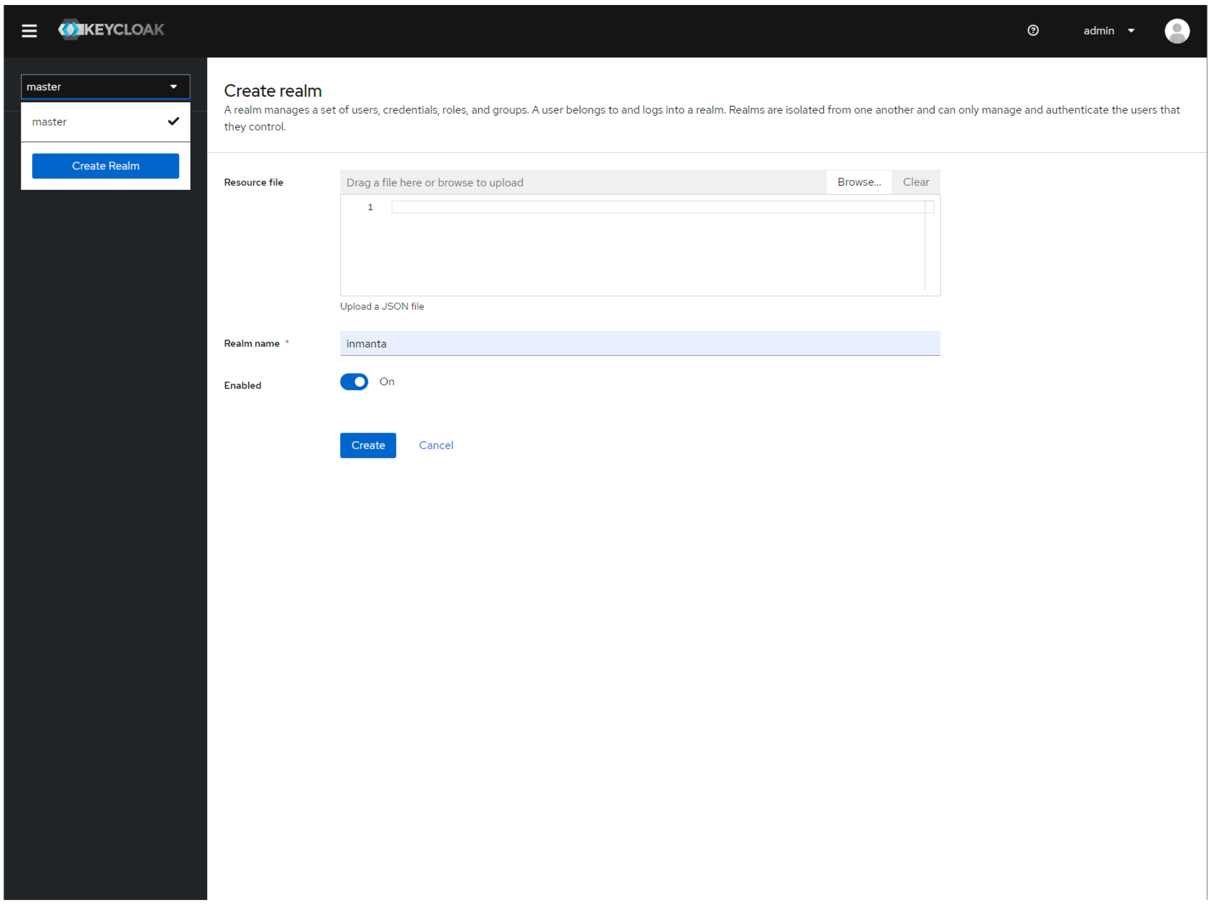
Version
20.0.5
Product
Default

Profile

Enabled features

Disabled features
ADMIN Supported ADMIN_FINE_GRAINED_AUTHZ Preview CLIENT_SECRET_ROTATION Preview
DECLARATIVE_USER_PROFILE Preview DOCKER Supported DYNAMIC_SCOPES Experimental MAP_STORAGE Experimental
OPENSHIFT_INTEGRATION Preview RECOVERY_CODES Preview SCRIPTS Preview TOKEN_EXCHANGE Preview
UPDATE_EMAIL Preview

Fig. 2: Create a new realm



Create realm

A realm manages a set of users, credentials, roles, and groups. A user belongs to and logs into a realm. Realms are isolated from one another and can only manage and authenticate the users that they control.

Resource file: Drag a file here or browse to upload

Upload a JSON file

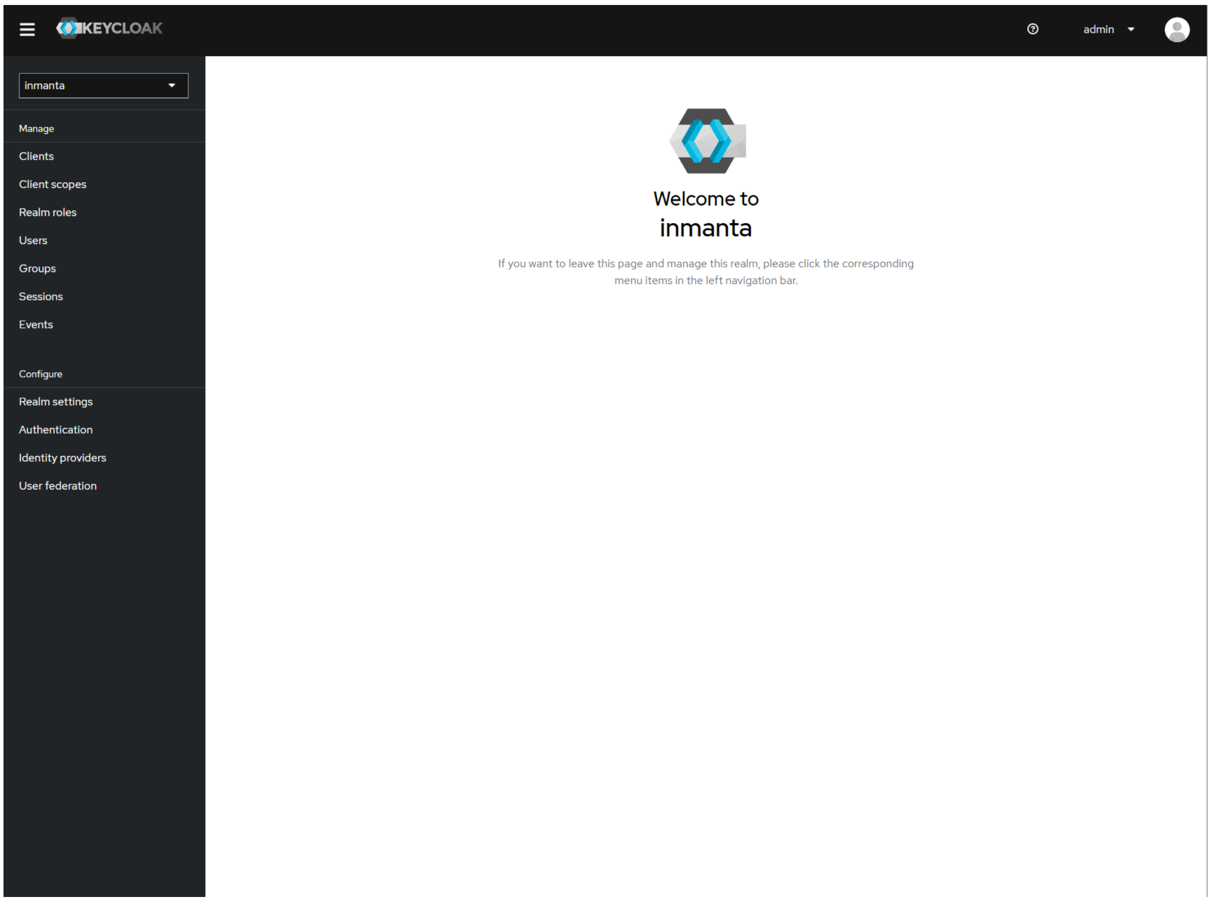
Realm name *: inmanta

Enabled: On

Fig. 3: Specify a name for the realm

Step 2: Add a new client to keycloak

Make sure the correct realm is active (the name is shown in the realm selection dropdown) to which you want to add a new client.



Welcome to
inmanta

If you want to leave this page and manage this realm, please click the corresponding menu items in the left navigation bar.

Fig. 4: The start page of your newly created realm.

Go to clients and click create on the right hand side of the screen.

Provide an id for the client and make sure that the client protocol is `openid-connect` and click save.

After clicking save, keycloak opens the configuration of the client. Modify the client to allow implicit flows and add valid redirect URIs and valid post logout redirect URIs. As a best practice, also add the allowed web origins. See the screenshot below as an example.

Go to the client scopes in your Client details.

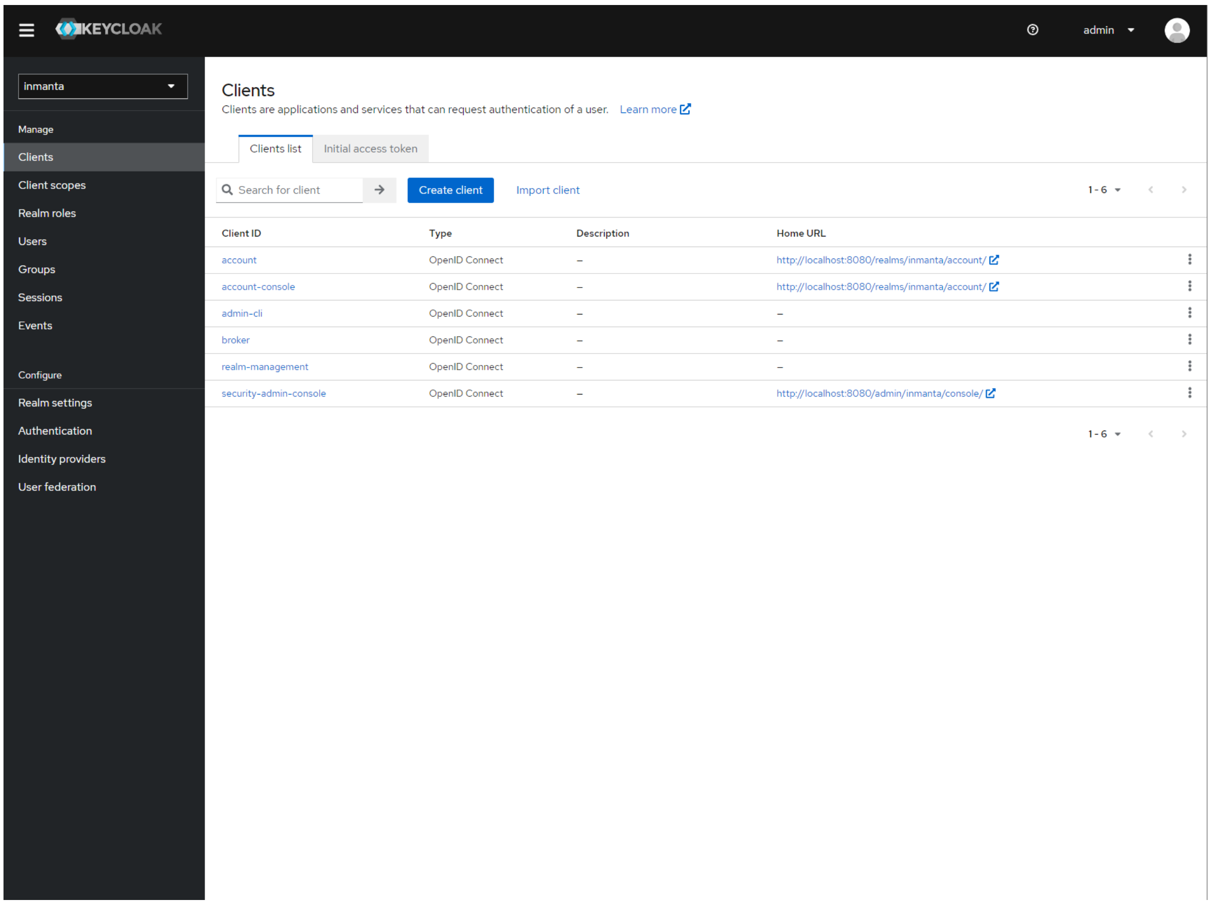
Add a mapper to add custom claims to the issued tokens for the API client type. Click on adding a new mapper and select By Configuration.

Select hardcoded claim, enter `:urn:inmanta:ct` as claim name and *api* as claim value and string as type. It should only be added to the access token.

Add a second mapper to add inmanta to the audience (only required for Keycloak 4.6 and higher). Click *add* again as in the previous step.

Fill in the following values:

- Name: inmanta-audience
- Mapper type: Audience
- Included Client Audience: inmanta

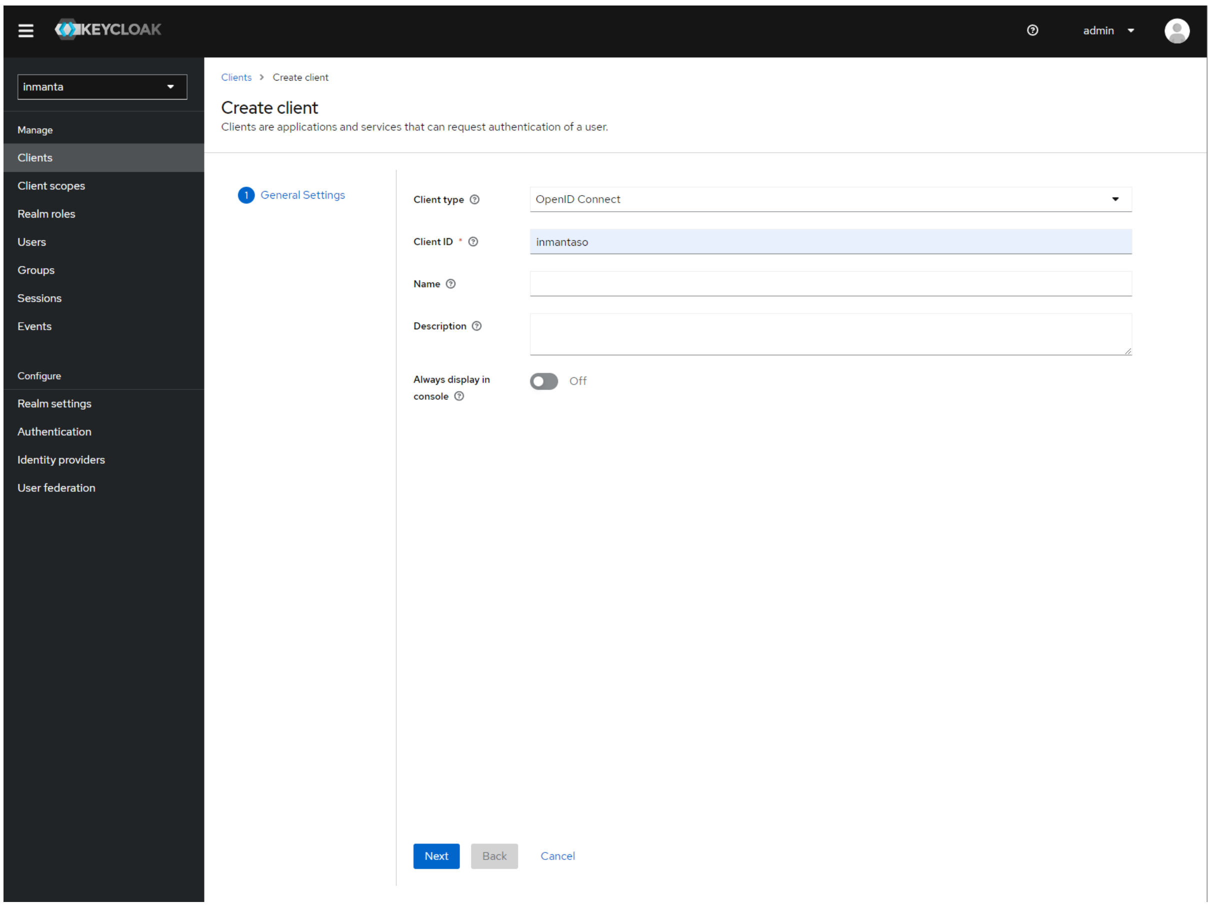


Clients

Clients are applications and services that can request authentication of a user. Learn more

Client ID	Type	Description	Home URL
account	OpenID Connect	–	http://localhost:8080/realms/inmanta/account/
account-console	OpenID Connect	–	http://localhost:8080/realms/inmanta/account/
admin-cli	OpenID Connect	–	–
broker	OpenID Connect	–	–
realm-management	OpenID Connect	–	–
security-admin-console	OpenID Connect	–	http://localhost:8080/admin/inmanta/console/

Fig. 5: Clients in the master realm. Click the create button to create an inmanta client.



Create client

Clients are applications and services that can request authentication of a user.

Client type: OpenID Connect

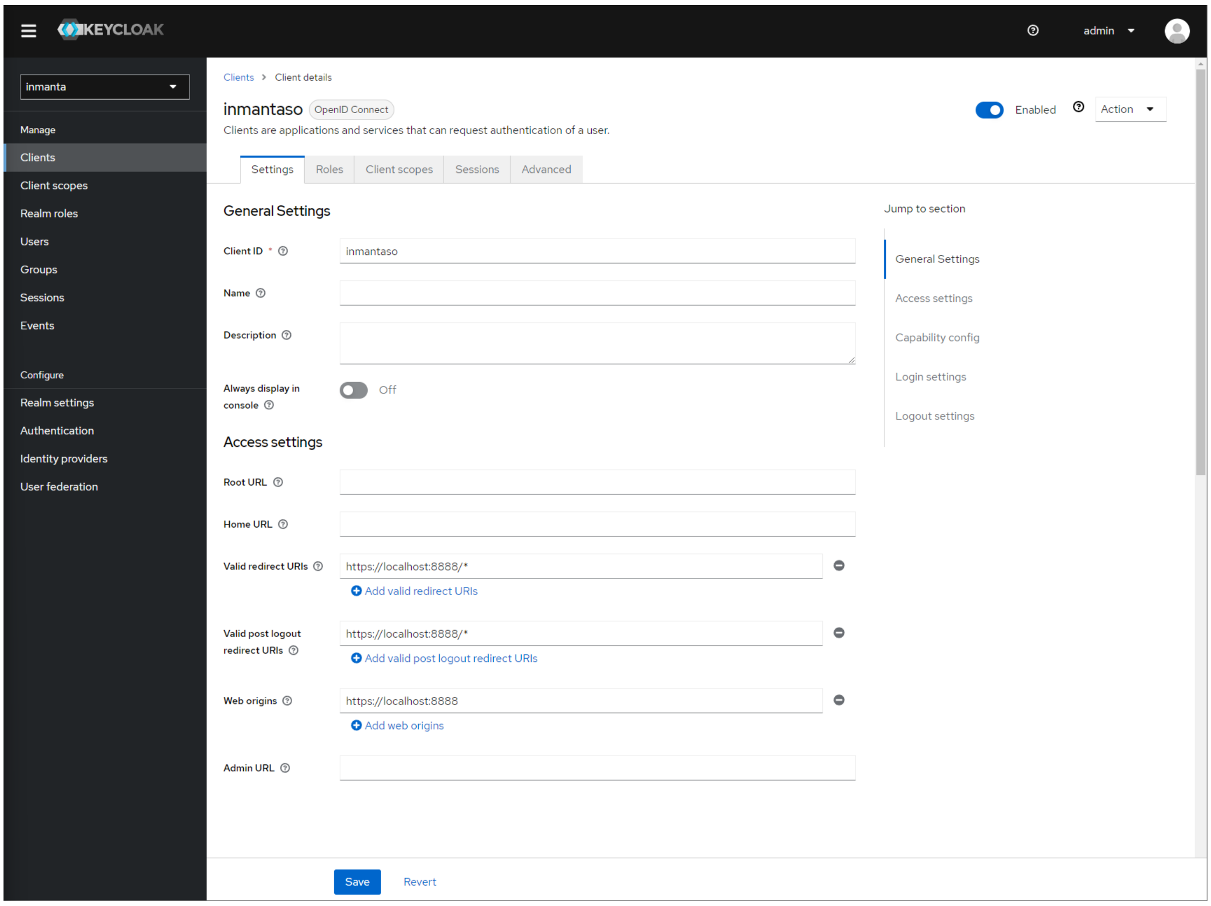
Client ID *: inmantaso

Name:

Description:

Always display in console: Off

Fig. 6: Create client screen



inmantaso OpenID Connect

Clients are applications and services that can request authentication of a user.

General Settings

Client ID *: inmantaso

Name:

Description:

Always display in console: Off

Access settings

Root URL:

Home URL:

Valid redirect URIs: https://localhost:8888/*

Valid post logout redirect URIs: https://localhost:8888/*

Web origins: https://localhost:8888

Admin URL:

- Add to access token: on

Click save.

Step 3: Configure inmanta server

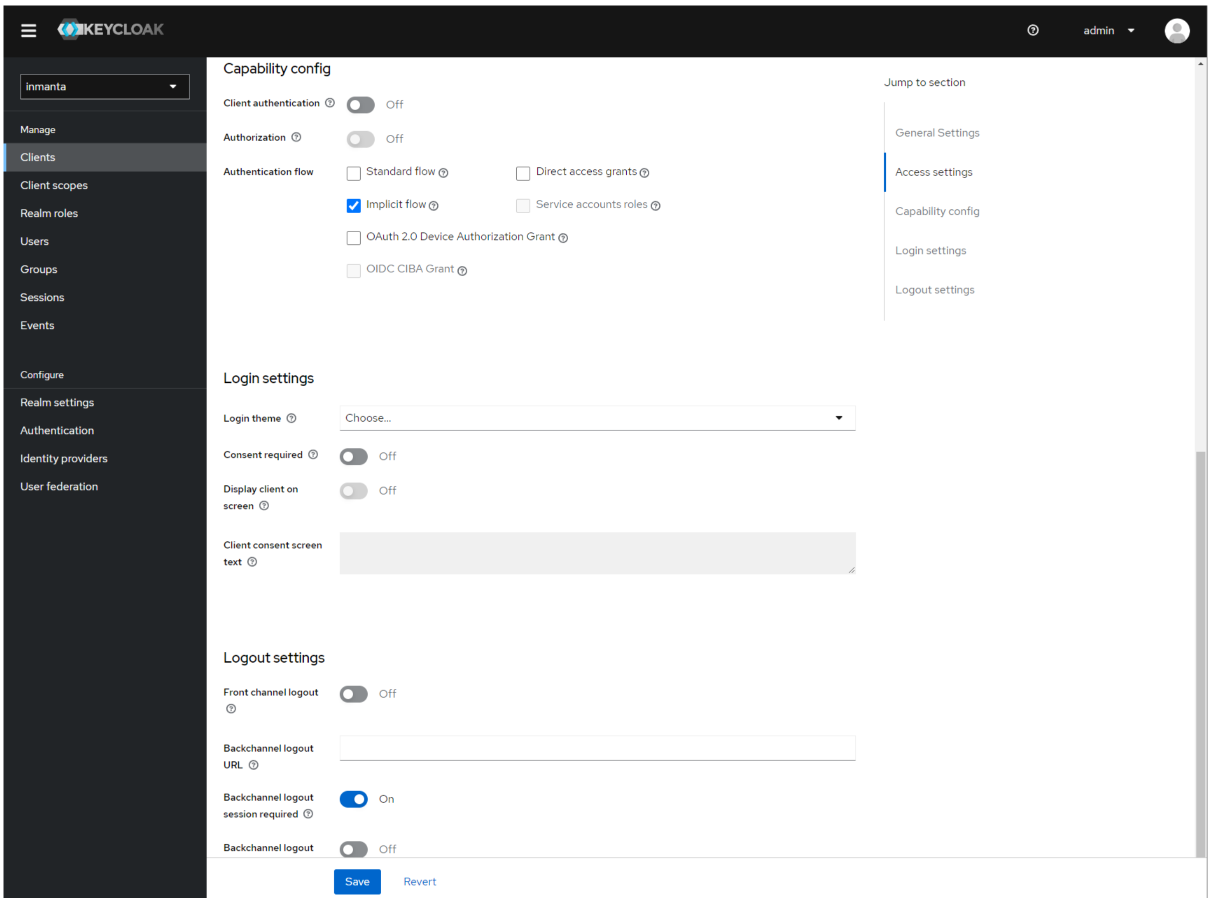
Select JSON format in the select box. This JSON string provides you with the details to configure the server correctly to redirect web-console users to this keycloak instance and to validate the tokens issued by keycloak.

Add the keycloak configuration parameters to the web-ui section of the server configuration file. Add a configuration file called */etc/inmanta/inmanta.d/keycloak.cfg*. Add the oidc_realm, oidc_auth_url and oidc_client_id to the web-ui section. Use the parameters from the installation json file created by keycloak.

```
[web-ui]
# generic OpenID connect configuration
oidc_realm=inmanta
oidc_auth_url=http://localhost:8080
oidc_client_id=inmantaso
```

> **Warning:** In a real setup, the url should contain public names instead of localhost, otherwise logins will only work on the machine that hosts inmanta server.

Configure a `auth_jwt_` block (for example `auth_jwt_keycloak`) and configure it to validate the tokens keycloak issues.



Capability config

Client authentication: Off

Authorization: Off

Authentication flow:
- [ ] Standard flow
- [ ] Direct access grants
- [x] Implicit flow
- [ ] Service accounts roles
- [ ] OAuth 2.0 Device Authorization Grant
- [ ] OIDC CIBA Grant

Login settings

Login theme: Choose...

Consent required: Off

Display client on screen: Off

Client consent screen text:

Logout settings

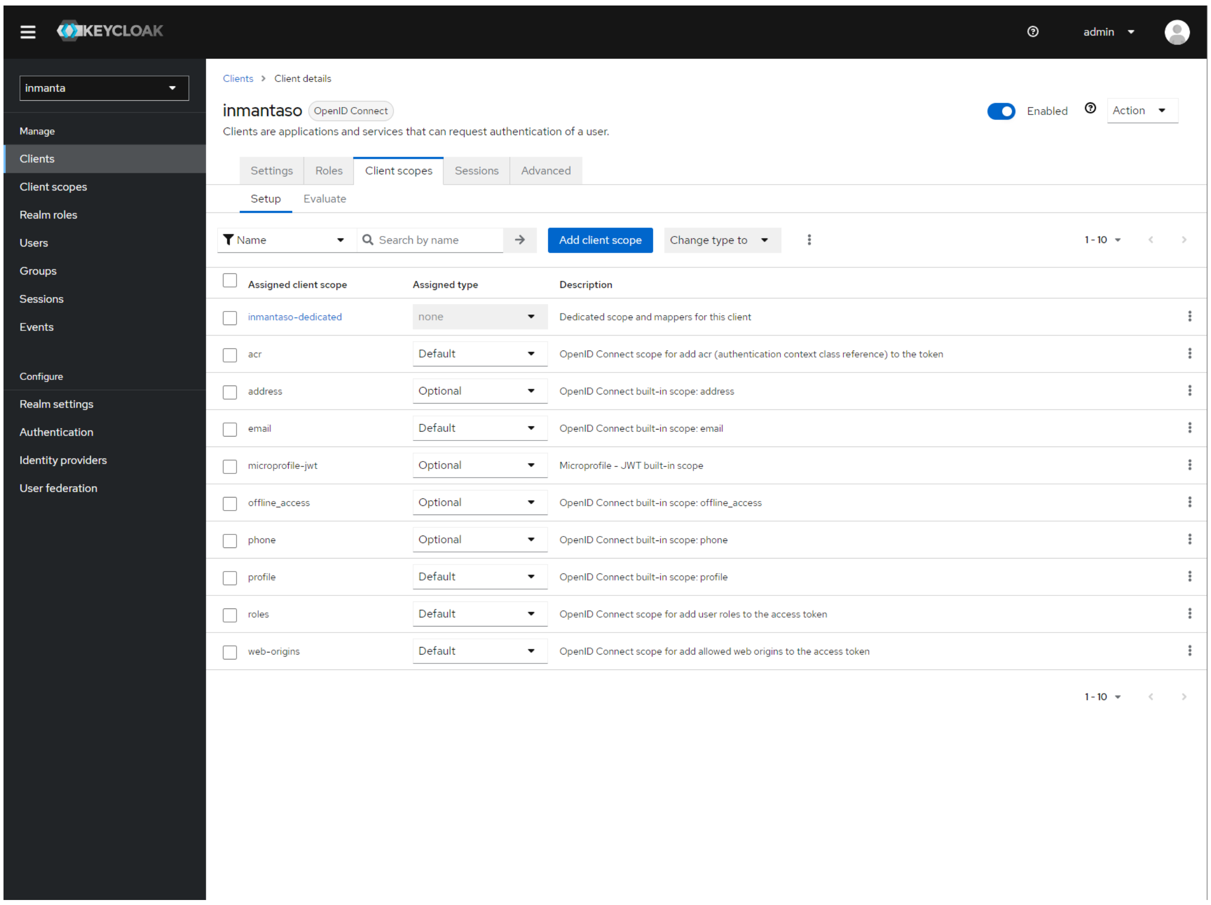
Front channel logout: Off

Backchannel logout URL:

Backchannel logout session required: On

Backchannel logout: Off

Fig. 7: Allow implicit flows (others may be disabled) and configure allowed callback urls of the web-console.

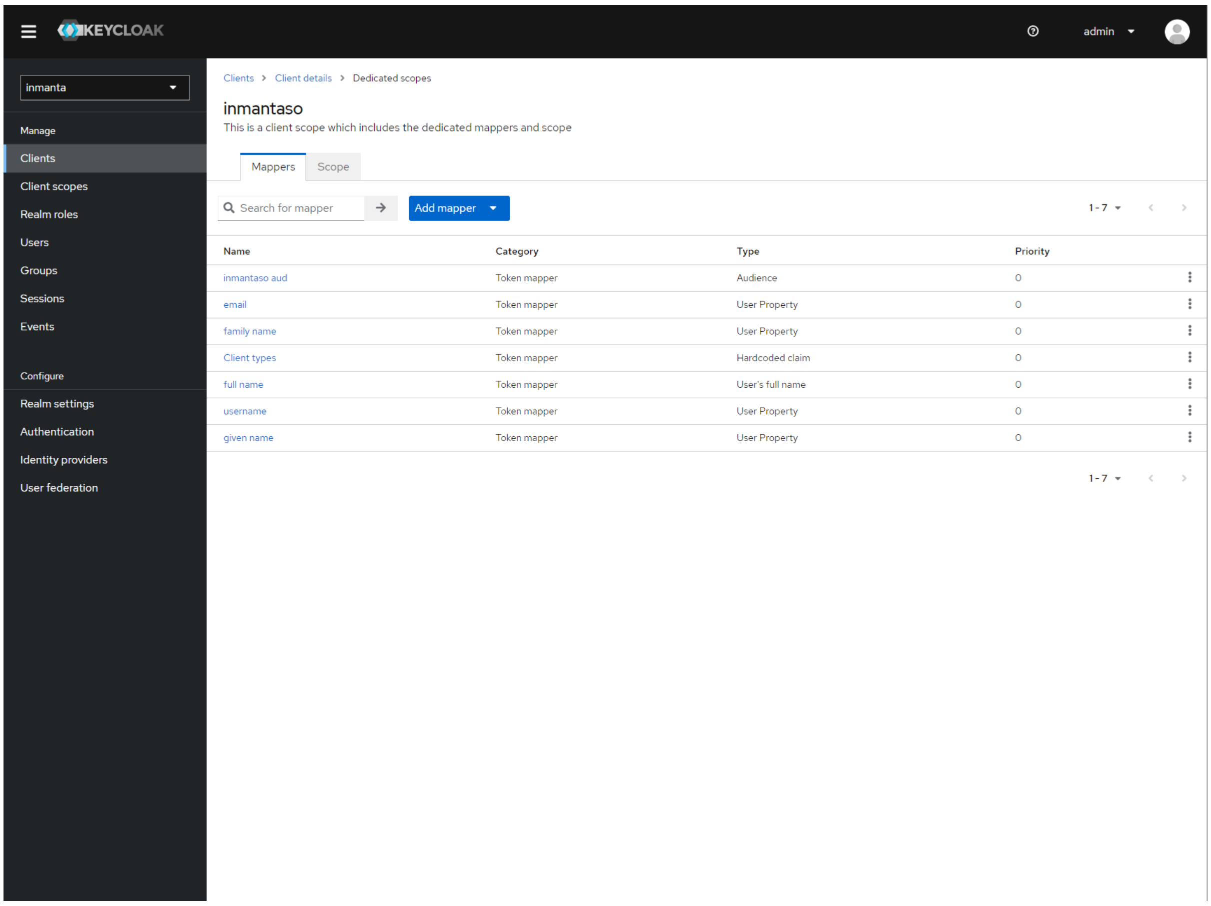


inmantaso OpenID Connect

Clients are applications and services that can request authentication of a user.

Assigned client scope	Assigned type	Description
inmantaso-dedicated	none	Dedicated scope and mappers for this client
acr	Default	OpenID Connect scope for add acr (authentication context class reference) to the token
address	Optional	OpenID Connect built-in scope: address
email	Default	OpenID Connect built-in scope: email
microprofile-jwt	Optional	Microprofile - JWT built-in scope
offline_access	Optional	OpenID Connect built-in scope: offline_access
phone	Optional	OpenID Connect built-in scope: phone
profile	Default	OpenID Connect built-in scope: profile
roles	Default	OpenID Connect scope for add user roles to the access token
web-origins	Default	OpenID Connect scope for add allowed web origins to the access token

Fig. 8: Click on inmantaso-dedicated to edit the dedicated scope and mappers.

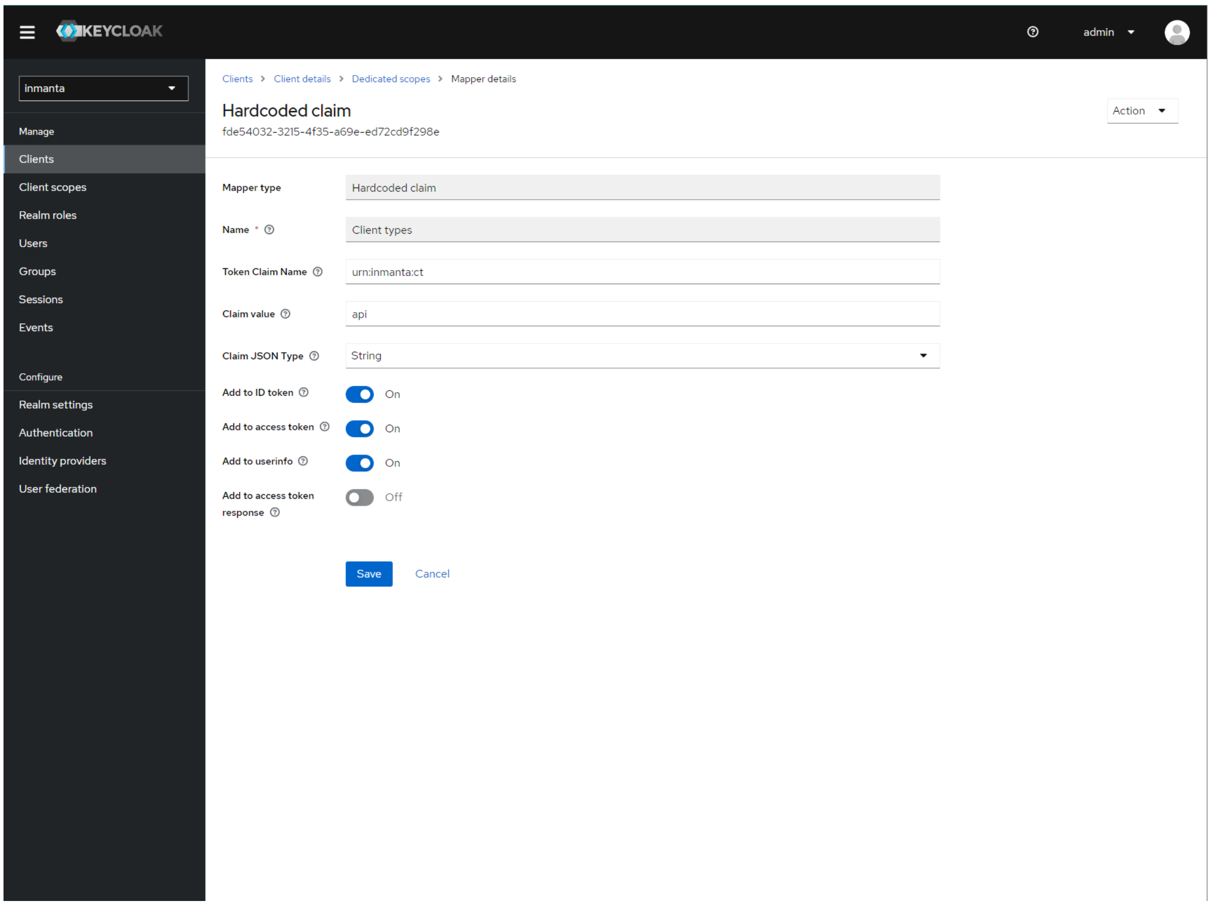


inmantaso

This is a client scope which includes the dedicated mappers and scope

Name	Category	Type	Priority
inmantaso aud	Token mapper	Audience	0
email	Token mapper	User Property	0
family name	Token mapper	User Property	0
Client types	Token mapper	Hardcoded claim	0
full name	Token mapper	User's full name	0
username	Token mapper	User Property	0
given name	Token mapper	User Property	0

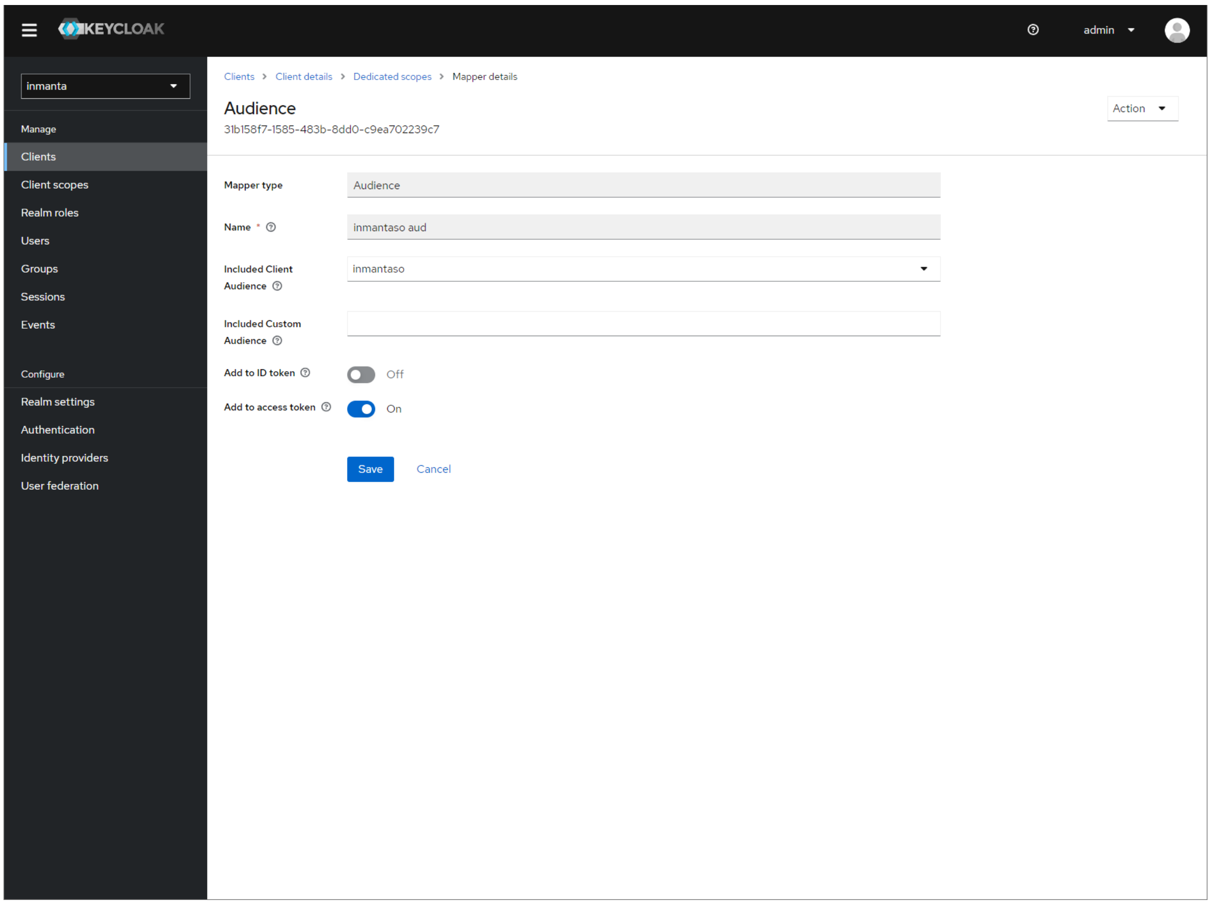
Fig. 9: Add a custom mapper to the client to include `:urn:inmanta:ct`



Hardcoded claim
fde54032-3215-4f35-a69e-ed72cd9f298e

Mapper type	Hardcoded claim
Name *	Client types
Token Claim Name	urn:inmanta:ct
Claim value	api
Claim JSON Type	String
Add to ID token	On
Add to access token	On
Add to userinfo	On
Add to access token response	Off

Fig. 10: Add the ct claim to all access tokens for this client.



Audience
31b158f7-1585-483b-8dd0-c9ea702239c7

Mapper type: Audience
Name *: inmantaso aud
Included Client Audience: inmantaso
Included Custom Audience:
Add to ID token: Off
Add to access token: On

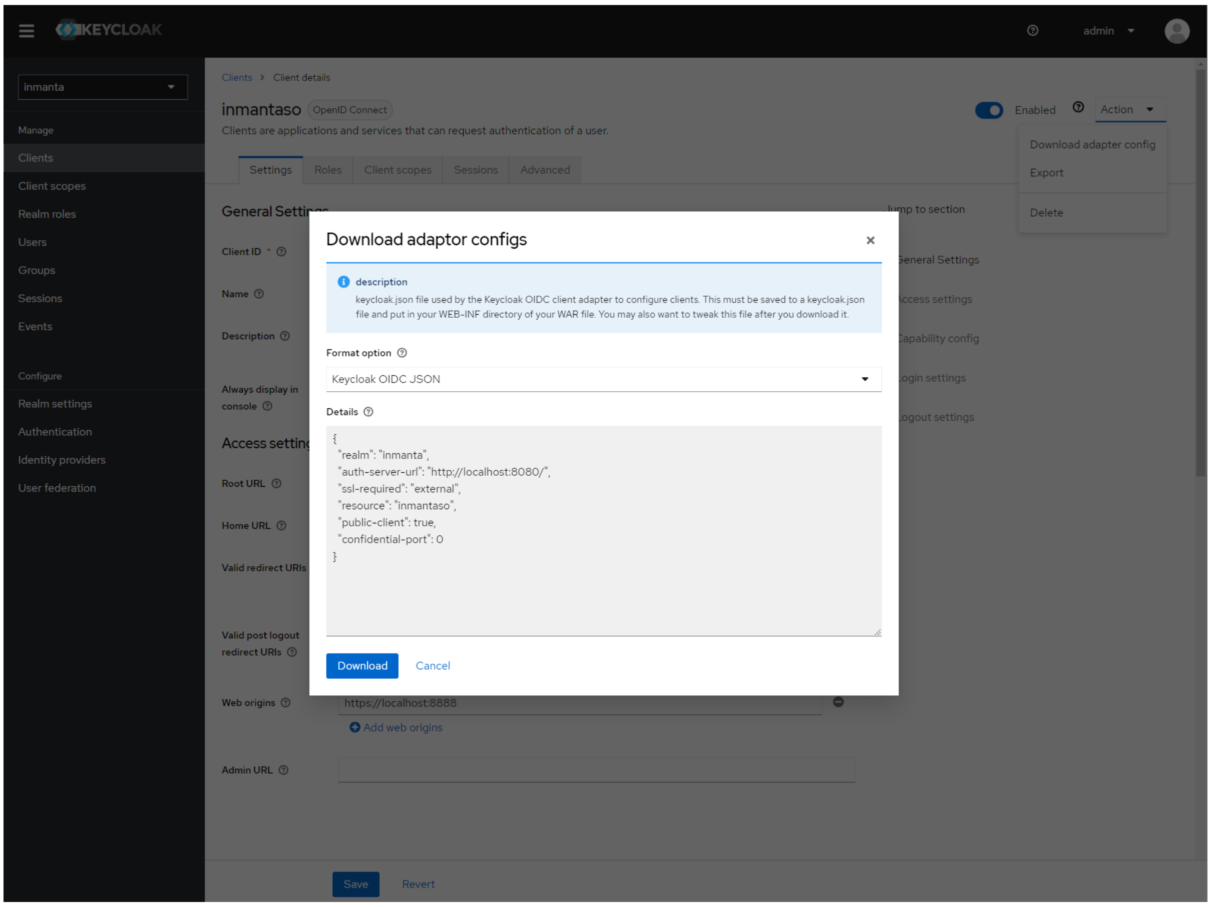
```
[server]
auth=true

[auth_jwt_keycloak]
algorithm=RS256
sign=false
client_types=api
issuer=http://localhost:8080/realms/inmanta
audience=inmantaso
jwks_uri=http://keycloak:8080/realms/inmanta/protocol/openid-connect/certs
validate_cert=false
```

Set the algorithm to RS256, sign should be false and client_types should be limited to api only. Next set the issuer to the correct value (watch out for the realm). Set the audience to the value of the resource key in the json file. Finally, set the jwks_uri so the server knows how to fetch the public keys to verify the signature on the tokens. (inmanta server needs to be able to access this url).

Both the correct url for the issuer and the jwks_uri is also defined in the openid-configuration endpoint of keycloack. For the examples above this url is http://localhost:8080/realms/inmanta/.well-known/openid-configuration (https://www.keycloak.org/docs/latest/securing_apps/index.html#endpoints)

Warning: When the certificate of keycloak is not trusted by the system on which inmanta is installed, set `validate_cert` to false in the `auth_jwt_keycloak` block for keycloak.



description
keycloak.json file used by the Keycloak OIDC client adapter to configure clients. This must be saved to a keycloak.json file and put in your WEB-INF directory of your WAR file. You may also want to tweak this file after you download it.

Format option: Keycloak OIDC JSON

Details

```
{
 "realm": "inmanta",
 "auth-server-url": "http://localhost:8080/",
 "ssl-required": "external",
 "resource": "inmantaso",
 "public-client": true,
 "confidential-port": 0
}
```

Fig. 11: Show the correct configuration parameters in JSON format. (Click on the top right dropdown 'Action' and pick 'Download adapter config'.)

8.6.5 Custom claims

Access to the orchestrator can be controlled using claim match expressions. In the section of the identity provider that you want to restrict you can configure the `claims` options. This is a multiline option where each line contains a match expression. There are two operators available:

- `in` for exact string match on a claim that contains a list of string values
- `is` for exact string match on a claim that is a string

You can use them as follows, for example each user gets two additional claims:

- `my:environments` which is a list of network environments the user is allowed to access. For example: lab and prod
- `my:scope` which indicates the scope of automation the orchestrator does. For example: network and dc

A user is allowed to have multiple environments but they can only have one scope. So that is why the environments is a list and scope is single string value.

On the lab orchestrator for the datacenter we can then configure it as follows:

```
[auth_jwt_keycloak]
algorithm=RS256
sign=false
client_types=api
issuer=http://localhost:8080/realms/inmanta
audience=inmantaso
jwks_uri=http://keycloak:8080/realms/inmanta/protocol/openid-connect/certs
validate_cert=false
claims=
  lab in my:environments
  my:scope is dc
```

This will only allow users with `lab` in the `my:environments` claim and `my:scope` equal to `dc`.

8.7 Environment variables

Environment variables can be supplied to the Inmanta server and its agents.

8.7.1 Supplying environment variables to the Inmanta server

The Inmanta server loads the environment variables specified in `/etc/sysconfig/inmanta-server` at startup. The example below defines three environment variables:

```
OS_AUTH_URL=http://openstack.domain
OS_USERNAME=admin
OS_PASSWORD=sYOUZdhcgwctSmA
```

These environment variables are accessible in a configurationmodel via the `std::get_env(name:  "string", default_value:  "string"=None)` plugin as shown in the following snippet:

```
import std
import openstack

provider = openstack::Provider(name="openstack",
                               connection_url=std::get_env("OS_AUTH_URL"),
                               username=std::get_env("OS_USERNAME"),
```

(continues on next page)

(continued from previous page)

```
                                    password=std::get_env("OS_PASSWORD"),
                                    tenant="dev")
```

8.7.2 Supplying environment variables to an agent

A manually started agent loads the environment variables specified in `/etc/sysconfig/inmanta-agent` at startup. This can be useful when a handler relies on the value of a certain environment variable.

8.8 Logging

This page describes the different logs files produced by the Inmanta server and its agents and explains what can be configured regarding to logging.

8.8.1 Overview different log files

By default log files are collected in the directory `/var/log/inmanta/`. Three different types of log files exist: the server log, the resource action logs and the agent logs. The server log and the resource action log files are produced by the Inmanta server. The agent log files are produced by the Inmanta agents.

Server log

The `server.log` file contains general debugging information regarding the Inmanta server. It shows information about actions performed by the Inmanta server (renewing parameters, purging resource action logs, etc.), API requests received by the Inmanta server, etc.

Resource action logs

The resource action log files contain information about actions performed on a specific resource. Each environment has one resource action log file. The filename of this log file looks as follows: `<server.resource-action-log-prefix>-<environment-id>.log`. The prefix can be configured with the configuration option *`server.resource-action-log-prefix`*.

The resource action log file contains information about the following resource action:

- **Store**: A new version of a configuration model and its resources has been pushed to the Inmanta server.
- **Pull**: An agent pulled its resources from the Inmanta server.
- **Deploy**: When an agent starts and ends the deployment of a certain resource.
- **Dryrun**: Execute a dryrun for a certain resource.

Agent logs

One agent produces the following three log files:

- `agent-<environment-id>.log`: This is the main log file of an agent. It contains information about when the agent started a deployment, which trigger caused that deployment, whether heartbeat messages are received from the server, whether the agent is a primary agent, etc.
- `agent-<environment-id>.out`: This log file contains all the messages written to the standard output stream of the resource handlers used by the agent.
- `agent-<environment-id>.err`: This log file contains all the messages written to the standard error stream of the resource handlers used by the agent.

8.8.2 Configure logging

Configuration options in Inmanta config file

The following log-related options can be set in an Inmanta config file:

- `log-dir`
- `purge-resource-action-logs-interval`
- `resource-action-log-prefix`

Documentation on these options can be found in the *Inmanta configuration reference*.

Change log levels server log

Edit the `--log-file-level` option in the ExecStart command of the inmanta-server service file. The inmanta-server service file can be found at `/usr/lib/systemd/system/inmanta-server.service`.

```
[Unit]
Description=The server of the Inmanta platform
After=network.target

[Service]
Type=simple
User=inmanta
Group=inmanta
ExecStart=/usr/bin/inmanta --log-file /var/log/inmanta/server.log --log-file-level 2 -
↪-timed-logs server
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

The `--log-file-level` takes the log-level as an integer, where `0=ERROR`, `1=WARNING`, `2=INFO` and `3=DEBUG`.

Apply the changes by reloading the service file and restarting the Inmanta server:

```
sudo systemctl daemon-reload
sudo systemctl restart inmanta-server
```

Log level manually started agent

The log level of a manually started agent can be changed in the same way as changing the log level of the Inmanta server. The service file for a Inmanta agent can be found at `/usr/lib/systemd/system/inmanta-agent.service`.

Log level auto-started agents

The default log level of an auto-started agent is INFO. Currently it's not possible to change this log level.

Resource action logs

The log level of the resource action log file is DEBUG. Currently it's not possible to change this log level.

Log level server-side compiles

The logs of a server side compile can be seen via the "Compile Reports" button in the web-console. The log level of these logs is DEBUG. Currently, it's not possible to change this log level.

Log level on CLI

By default logs are written to standard output when the `inmanta` or the `inmanta-cli` command is executed. The default log level is INFO. The log level of these commands can be changed by passing the correct number of v's with the option `-v`.

- `-v = warning`
- `-vv = info`
- `-vvv = debug`
- `-vvvv = traces`

By specifying the `-X` option, stacktraces are also shown written to standard output when an error occurs. When the `--log-file` option is specified on the commandline, logs are written to file instead of the standard output.

8.9 Performance Metering

This guide explains how to send performance metrics about the inmanta server to influxdb.

The inmanta server has a built-in pyformance instrumentation for all API endpoints and supports sending the results to influxdb.

8.9.1 Configuration summary

To enable performance reporting, set the options as found under *`influxdb`* in the server configuration file.

For example:

```
[influxdb]
# The hostname of the influxdb server
host = localhost
# The port of the influxdb server
port = 8086
# The name of the database on the influxdb server
name = inmanta
tags= environment=prod,az=a
```

8.9.2 Setup guide

1. To install influxdb, follow the instructions found at docs.influxdata.com.
2. Create a database to send the data to:

   ```
   influx
   CREATE DATABASE inmanta
   ```

3. Update the inmanta config file, add the following block

   ```
   [influxdb]
   # The hostname of the influxdb server
   host = localhost
   # The port of the influxdb server
   port = 8086
   # The name of the database on the influxdb server
   name = inmanta
   ```

4. Restart the inmanta server.
5. [optional] install grafana, follow the instructions found at https://grafana.com/grafana/download
6. [optional] load the inmanta dashboard found at https://grafana.com/grafana/dashboards/10089-inmanta-api-performance/

8.9.3 Reported Metrics

This section assumes familiarity with influxdb. See here.

All metrics are reported under the measurement *metrics*. Different measurements are distinguished by a tag called *key*.

Two main types of metrics are reported: 1. Metrics related to API performance 2. Others

API performance metrics

Each API method is reported with a *key=rpc.{endpoint_name}*. The *endpoint_name* is the server's internal name for the endpoint.

To know which url corresponds to which method, please consult either

- the *operationId* field of the OpenAPI spec or
- the method names in `inmanta.protocol.methods` and `inmanta.protocol.methods_v2`

The fields available for each API endpoint are (cfr metrics timer):

field	type	description
15m_rate	float	fifteen-minute exponentially-weighted moving average of the request rate
5m_rate	float	five-minute exponentially-weighted moving average of the request rate
1m_rate	float	one-minute exponentially-weighted moving average of the request rate
mean_rate	float	mean of the request rate
min	float	minimal observed request latency
50_percentile	float	median (50 percentile) observed request latency
75_percentile	float	75 percentile observed request latency
95_percentile	float	95 percentile observed request latency
99_percentile	float	99 percentile observed request latency
999_percentile	float	999 percentile observed request latency
max	float	maximal observed request latency
avg	float	average observed latency
std_dev	float	standard deviation of the observed latency
count	float	number of calls seen since server start
sum	float	total wall-time spent executing this call since server start

Other Metrics

Key	Type	Unit	Description
self.spec.cp	int	ns	The result of a small CPU benchmark, executed every second. Provides a baseline for machine performance.

8.10 Reverse proxy and Web Application Firewall

Communication between inmanta components and to the northbound API uses REST over HTTP(S). This section describes how to move the API behind a reverse proxy and optionally enable a web Application firewall. This is meant for all external traffic towards the orchestrator. It is not supported to proxy traffic from the compiler and agents to the server.

This guide focuses on access to the web-console, and to the northbound API. This guide works for both the OSS and the full version of the product.

8.10.1 Setup a reverse proxy

A reverse proxy receives the calls and proxies them to the inmanta service orchestrator API. This guide gives examples to set this up with Apache HTTPD, but similar rules could also be applied to NGINX or other reverse proxies. This guide assumes that the reverse proxy is installed on the same machine as the orchestrator.

1. Make sure you do not bind the orchestrator to the IP used by the proxy server so it cannot be bypassed. If only auto started agents are used, it is recommended to set the bind-address to localhost. See `server.bind-address` and `server.bind-port`. If you have remote agents, make sure that either by having multiple IPs or using firewall rules that the agents can connect directly to the orchestrator.

2. Install Apache HTTPD and make sure it is configured correctly (listen to the correct interfaces, ports, SSL, access control, . . .)

3a. The easiest setup is to proxy all traffic directly to the orchestrator:

```
# Proxy all requests to the orchestrator
<Location /console>
    ProxyPass http://localhost:8888/
```

(continues on next page)

(continued from previous page)

```
    Allow from all
    # Or limit access to certain users or prefixes
    # Allow from 10.x.x.x/24
</Location>
```

3b. Only proxy the calls that the orchestrator has endpoints for. Everything else will be handled by the reverse proxy:

```
# Web Console is a static single page application (SPA)
<Location /console>
    ProxyPass http://localhost:8888/console

    # Limit the possible methods to only get the content
    AllowMethods GET HEAD OPTIONS

    Allow from all
    # Or limit access
    # Allow from 10.x.x.x/24
</Location>

# Generic API: used by agents, web-console, integrations, ...
# Unless detailed error reports are requested, this API should not␣
↪be made available to
# any portals or tools
<Location /api>
    ProxyPass http://localhost:8888/api

    Allow from all
    # Or limit access
    # Allow from 10.x.x.x/24
</Location>

# LSM API: the northbound API called by tools such as customer␣
↪portals
<Location /lsm>
    ProxyPass http://localhost:8888/lsm

    Allow from all
    # Or limit access
    # Allow from 10.x.x.x/24
</Location>
```

When only exposing the LSM API even more specific proxy rules can be used. In the next section we provide example rules to restrict this with mod_security.

8.10.2 Web Application Firewall

This section provides configuration guidelines to enable additional filtering using mod security. These rules can of course be ported to other types of web application firewalls.

1. Install mod_security and enable it in Apache HTTPD according to their setup instructions.
2. Optional: Enable JSON body decoding to make sure only valid JSON reaches the orchestrator. This is available since version 2.8, however it is not enabled in the RPMS included with RHEL and Centos. Third party repos provide versions with JSON decoding enabled or distribution such as NGINX WAF.

JSON decoding is enabled when a similar config stanza is in the configuration:

```
# Make sure mod security is on and it inspects the body
SecRuleEngine On
SecRequestBodyAccess On

# Enable json body decoding when the content type is set to `application/
↪json`
SecRule REQUEST_HEADERS:Content-Type "application/json" \
    "id:'200001',phase:1,t:none,t:lowercase,pass,nolog,
↪ctl:requestBodyProcessor=JSON"
```

3. Add the generic inmanta rules. These will make sure that if the requests goes to an API it will only accept valid JSON. If the JSON processor is not enabled, these rules will still work, but the protection is reduced because invalid JSON can still reach the inmanta service orchestrator API. The rules are defined so that they will only trigger on calls to inmanta service orchestrator endpoints.

```
# Classify the call based on the request uri.
SecRule REQUEST_URI "@beginsWith /api/" \
    "id:'200501',phase:1,setvar:'tx.inmanta_context=api'"
SecRule REQUEST_URI "@beginsWith /api/v2/docs" \
    "id:'200502',phase:1,setvar:'tx.inmanta_context=docs'"
SecRule REQUEST_URI "@beginsWith /console" \
    "id:'200504',phase:1,setvar:'tx.inmanta_context=static'"
SecRule REQUEST_URI "@beginsWith /lsm/" \
    "id:'200510',phase:1,setvar:'tx.inmanta_context=lsm'"
SecRule REQUEST_URI "@beginsWith /lsm/v1/service_catalog_docs" \
    "id:'200511',phase:1,setvar:'tx.inmanta_context=docs'"

# All api and lsm calls should be json content so that the body will be␣
↪parsed by modsec
# If JSON decoding is not enabled, it will force the content type however␣
↪mod_security does not validate
# if the body is JSON
SecRule TX:INMANTA_CONTEXT "@rx api|lsm" \
    "id:'200600',phase:1,deny,status:400,msg:'API and LSM only accept json␣
↪content',chain"
    SecRule REQUEST_HEADERS:Content-Type "!@rx application/json" \
        "t:lowercase"

# Inmanta supports unicode, however this is often used in templates that␣
↪generate
# input for other systems. This rule will validate all utf8 encodings. It␣
↪is only enabled
# when sending data to inmanta backends
SecRule TX:INMANTA_CONTEXT "!@streq ''" \
```

(continues on next page)

(continued from previous page)

```
    "id:'200601',phase:1,deny,status:400,msg:'Invalid UTF provided',chain"
    SecRule ARGS "@validateUtf8Encoding" \
        "t:none"
```

This ruleset has been tested to be compatible with the OWASP core rule set. However, it does not do scoring. If an anomaly is detected a 400 request is returned. It does not return the default 403 because this tricks our web-console into warning the user to authenticate.

When the northbound API is only used for calls to LSM to manage service instances, mod_security can be used to restrict access even more. The following rules ensure that only calls for service "network" are allowed and callback management. The rules are set up in such a way that additional urls can be easily added to the ruleset:

```
# Only allow certain paths required for the "customer portal" to function:
SecAction \
"id:300001,\
    phase:1,\
    nolog,\
    pass,\
    t:none,\
    setvar:'tx.allowed_urls=|/lsm/v1/service_inventory/network| |/lsm/v1/
↪callbacks'"

SecRule REQUEST_URI "!@withIN %{tx.allowed_urls}" \
    "id:300002,phase:1,t:lowercase,deny,status:404"
```

When the OWASP core ruleset is enabled and particularly when JSON decoding is enabled, mod_security will also scan for SQL and XSS attacks. Especially the latter can be useful if a customer portal uses the API directly and the service model has free form attributes that can hold any content. In that case it may be useful to also use mod_security to protect against for example stored XSS attacks.

8.11 Support Procedure

1. **Create a support archive:**

    1. Using the CLI tool:

        1. Log on to the orchestrator machine
        2. Run one of the following commands:

        - if the orchestrator is still running:

        ```
        inmanta-support-tool collect-from-server
        ```

        - if the orchestrator is not running:

        ```
        inmanta-support-tool --config-dir /etc/inmanta/inmanta.dir␣
        ↪collect-full
        ```

    2. Using the web-console:

    Use the `Download support archive` button at the right top of the `Home > Status` page. By clicking this button the support archive will be downloaded.

2. Classify the severity of the incident

Severity Level	Service Window	File by phone	Description
Urgent	24/7	yes	Severe negative impact on operations. Unable to use orchestrator
High	24/7	yes	Degraded ability to use the orchestrator.
Normal	5/7		Work around is available.
Low	5/7		Information request, not related to any error.

3. Create a support ticket on support.inmanta.com
    1. Log in with your personal support count
    2. Click 'Submit a request'
    3. Describe the problem as best a possible
    4. Attach the archive created by the support tool to the support request
4. If the severity is high or urgent also contact the support phone number you have received and reference the issue you just created.

8.12 Upgrading the orchestrator

Upgrading the orchestrator can be done either in-place or by setting up a new orchestrator next to the old one and migrating the state from the old to the new instance. The sections below describe the upgrade procedure for each of both situations. These procedures can be used for major and non-major version upgrades.

Note: Make sure to read the new version's changelog for any version specific upgrade notes, before proceeding with any of the upgrade procedures mentioned below.

8.12.1 Upgrading the orchestrator in-place

This section describes how to upgrade an orchestrator in-place.

Note: Pre-requisite

- Before upgrading the orchestrator to a new major version, make sure the old orchestrator is at the latest version available within its major.
- Upgrades should be done one major version at a time. Upgrading from major version `X` to major version `X+2`, should be done by upgrading from `X` to `X+1` and then from `X+1` to `X+2`.

1. Halt all environments (by pressing the `STOP` button in the web-console for each environment).
2. Create a backup of the database:

```
pg_dump -U <db_user> -W -h <host> <db_name> > <db_dump_file>
```

3. Replace the content of the `/etc/yum.repos.d/inmanta.repo` file with the content for the new ISO version. This information can be obtained from the *installation documentation page* for the new ISO version.
4. Upgrade the Inmanta server. The orchestrator will automatically restart when the upgrade has finished. It might take some time before the orchestrator goes up, as some database migrations will be done.

```
dnf update inmanta-service-orchestrator-server
```

5. When accessing the web console, all the environments will be visible, and still halted.

6. One environment at a time:

   a. In the **Desired State** page of the environment, click `Update project & recompile`, accessible via the dropdown of the `Recompile` button. (`/console/desiredstate?env=<your-env-id>`).

   b. Resume the environment by pressing the green `Resume` button in the bottom left corner of the console.

> **Warning:** Make sure the compilation has finished and was successful before moving on to the next steps.

8.12.2 Upgrading by migrating from one orchestrator to another orchestrator

This document describes how to upgrade to a new version of the orchestrator by setting up a new orchestrator next to the existing orchestrator and migrating all the state from the existing to the new orchestrator. This procedure should be followed when an in-place upgrade of the orchestrator is not possible e.g. when the operating system needs to be upgraded alongside the orchestrator.

Terminology

The procedure below describes how to migrate from one running orchestrator denoted as the 'old orchestrator' to another one denoted as the 'new orchestrator'.

Procedure

Note: Pre-requisite

- Before upgrading the orchestrator to a new major version, make sure the old orchestrator is at the latest version available within its major.
- Upgrades should be done one major version at a time. Upgrading from major version `X` to major version `X+2`, should be done by upgrading from `X` to `X+1` and then from `X+1` to `X+2`.

---

1. **[New Orchestrator]**: Make sure the desired version of the orchestrator is installed, by following the installation instructions (see *Install Inmanta*) and set up a project to validate that the orchestrator is configured correctly (config, credentials, access to packages, etc.).

---

2. **[Old Orchestrator]** Halt all environments (by pressing the `STOP` button in the web-console for each environment).
3. **[Old Orchestrator]** Stop and disable the server:

```
sudo systemctl disable --now inmanta-server.service
```

4. **[Old Orchestrator]** Make a dump of the server database using `pg_dump`.

```
pg_dump -U <db_user> -W -h <host> <db_name> > <db_dump_file>
```

---

5. **[New Orchestrator]** Make sure the server is stopped:

```
sudo systemctl stop inmanta-server.service
```

6. **[New Orchestrator]** Drop the inmanta database and recreate it:

```
# drop the database
$ psql -h <host> -U <db_user> -W
drop database <db_name>;
exit

# re-create it
$ sudo -u postgres -i bash -c "createdb -O <db_user> <db_name>"
```

7. **[New Orchestrator]** Load the dump of the server database using `psql`.

```
psql -U <db_user> -W -h <host> -f <db_dump_file> <db_name>
```

8. **[New Orchestrator]** Start the orchestrator service, it might take some time before the orchestrator goes up, as some database migration will be done:

```
sudo systemctl enable --now inmanta-server.service
```

9. **[New Orchestrator]** When accessing the web console, all the environments will be visible, and still halted.
10. **[New Orchestrator]** One environment at a time:

    a. In the **Desired State** page of the environment, click `Update project & recompile`, accessible via the dropdown of the `Recompile` button. (`/console/desiredstate?env=<your-env-id>`).

    b. Resume the environment by pressing the green `Resume` button in the bottom left corner of the console.

    Warning: Make sure the compilation has finished and was successful before moving on to the next steps.

8.13 Inmanta Web Console

The Inmanta Web Console is a web GUI for the Inmanta Service Orchestrator.

8.13.1 Browser support

For using the web console, the last 2 versions of the Chrome, Firefox, Edge and Safari browsers are supported. For security reasons it's always recommended to use the latest version of these browsers.

8.13.2 Proxy

When configuring a proxy for the web-console, the url should always end in `/console`. The web-console uses the `/console` part as an `anchor`. This anchor is something recognizable in the url that is always present. It is also considered to be the root of the app. So a potential proxy would come before the anchor. And the app pages come after the anchor. If no anchor is present in the url, we know the url is faulty. So from an app perspective, the url has the following structure: (`proxy`) + (`anchor`) + (`application defined urls`)

Examples

Given the input url, the application will use the following `proxy + anchor`.

Scenario	input url	`proxy + anchor`
Empty proxy respected	/console/resources?env=abcd	/console
Proxy respected	/someproxy/console	/someproxy/console
Faulty url ignored	/someproxy	/console

CHAPTER

NINE

FREQUENTLY ASKED QUESTIONS

9.1 How do I use Inmanta with a http/https proxy?

Use the http_proxy and https_proxy environment variables to specify the proxy server to use. For the server installed from our RPMs, add the environment variable to the systemd unit file. Copy inmanta-server.service from /lib/systemd/systemd/system to /etc/systemd/system and add the following lines to the [Service] section with the correct proxy server details:

```
Environment=http_proxy=1.2.3.4:5678
Environment=https_proxy=1.2.3.4:5678
```

Afterwards run systemctl daemon-reload and restart the inmanta server.

9.2 I get a click related error/exception when I run inmanta-cli.

The following error is shown:

```
Traceback (most recent call last):
    File "/usr/bin/inmanta-cli", line 11, in <module>
        sys.exit(main())
    File "/opt/inmanta/lib64/python3.4/site-packages/inmanta/main.py", line 871, in␣
↪main
        cmd()
    File "/opt/inmanta/lib64/python3.4/site-packages/click/core.py", line 722, in __
↪call__
        return self.main(*args, **kwargs)
    File "/opt/inmanta/lib64/python3.4/site-packages/click/core.py", line 676, in main
        _verify_python3_env()
    File "/opt/inmanta/lib64/python3.4/site-packages/click/_unicodefun.py", line 118,␣
↪in _verify_python3_env
        'for mitigation steps.' + extra)
RuntimeError: Click will abort further execution because Python 3 was configured to␣
↪use ASCII as encoding for the environment.  Consult http://click.pocoo.org/python3/
↪for mitigation steps.
```

This error occurs when the locale are not set correctly. Make sure that LANG and LC_ALL are set. For example:

```
export LC_ALL=en_US.utf8
export LANG=en_US.utf8
```

9.3 The model does not compile and exits with "could not complete model".

There is an upperbound on the number of iterations used in the model transformation algorithm. For large models this might not be enough. This limit is controlled with the environment variable INMANTA_MAX_ITERATIONS The default value is set to 10000 iterations.

CHAPTER

TEN

GLOSSARY

agent
: The process that enforces the desired state described by *resources* by executing *handlers*. Each agent is responsible for all resources that go to a single device or API endpoint.

configuration model
: The *desired state* of the an *environment* is expressed in the configuration model. This model defines the desired state of all resources that need to be managed by Inmanta.

desired state
: The desired state expresses the state of all resources that Inmanta manages. Expressing a configuration in function of desired state makes the orchestrator more robust to failures compared to imperative based orchestration. An agent uses a *handler* to read the current state of the a resource and derive from the difference between current and desired state the actions required to change the state of the resource. Desired state has the additional benefit that Inmanta can show a dry run or execution plan of what would change if a new configuration is deployed.

  Imperative solutions require scripts that execute low level commands and handle all possible failure conditions. This is similar to how a 3D printer functions: a designer send the desired object (desired state) to the 3D printer software and this printer converts this to layers that need to be printed. An imperative 3D model, would require the designer to define all layers and printer head movements.

DSL
: Domain specific language. An Inmanta configuration model is written in a the Inmanta modelling DSL.

entity
: Concepts in the infrastructure are modelled in the configuration with entities. An entity defines a new type in the configuration model. See *Entities*.

environment
: Each environment represents a target infrastructure that inmanta manages. At least one environment is required, but often multiple environments of the same infrastructure are available such as development, integration and testing.

expert feature
: A feature that is stable, but requires great care and/or knowledge to use properly.

facts
: A resource in an infrastructure may have multiple properties that are not managed by Inmanta but their value is required as input in the configuration or for reporting purposes. *handlers* take care of extracting these facts and reporting them back to the server. More information in the *using facts* section.

handler
: A handler provides the interface between a resource in the model and the resource in the infrastructure. The agent loads the handler and uses it to read the current state, discover *facts* and make changes to the real resource.

infrastructure
: This is what Inmanta manages. This could be virtual machines with resources in these virtual machines. Physical servers and their os. Containers or resources at a cloud provider without any servers (e.g. "serverless")

infrastructure-as-code
Wikepedia defines "Infrastructure as code" as *the process of managing and provisioning computer data centers through machine-readable definition files, rather than physical hardware configuration or interactive configuration tools.* Inmanta achieves this by using a desired state configuration model that is entirely expressed in code.

instance
An *instance* of an *entity*. See also *Instantiation*.

main.cf
The file that defines the starting point of a configuration model. This file often only instantiates some high level entities and imports specific module.

module
A *configuration model* consists of multiple configuration modules. A module provides a partial and reusable configuration model and its related resources such as files, templates, . . . The *module developer guide* provides more details.

orchestration
Orchestration is the process of provisioning resources in the correct order and when they are available configuring them. Inmanta support both provisioning and configuring resources but can also delegate tasks to other (existing) tools.

plugin
A plugin is a python function that can be used in the *DSL*. This function recieves arguments from the configuration model and navigate relations and read attributes in the runtime model. Each function can also return a value to the model. Plugins are used for complex transformation based on data in the configuration model or to query external systems such as CMDBs or IPAM tools.

project
The management server of the Inmanta orchestrator can manage distinctive infrastructures. Each distinct infrastructure is defined in the server as a project. Each project consists of one or more *environment* such as development, integration and production.

relation
An attribute of an entity that references an other entity. Plugins, such as templates, can navigate relations. See also *Relations*.

resource
Inmanta orchestrates and manages resources, of any abstraction level, in an infrastructure. Examples of resources are: files and packages on a server, a virtual machine on a hypervisor, a managed database as a PaaS provider, a switch port on a switch, . . .

A resource has attributes that express the desired value of a property of the resource it represents in the infrastructure. For example the *`mode`* attribute of the the *`std::File`* resource. This attribute indicates the desired permissions of a UNIX file.

A resource needs to have a unique identifier in an environment. This identifier needs to be derived from attributes of the resource. This ensures that the orchestrator can (co-)manage existing resources and allows quick recovery of the orchestrator in failure conditions. This unique identifier consists of multiple fields. For example, `std::File[vm1,path="/etc/motd"]` This id contains the type of the resource, the name of the *agent* and the unique id with its value for this resource. The resource designer determines how this id is derived.

The fields in the id are:

- The first field is the type of the resource. For example: `std::File`
- The second field is the name of the agent that manages/groups the resource. For example: the name of the machine on which the file is defined `vm1`
- The third field is the identifying attribute and the value of this attribute. For example: the `path` of the file uniquely identifies a file on a machine.

resource handler
See *handler*

unknown

A user always provides a complete configuration model to the orchestrator. Depending on what is already deployed, Inmanta will determine the correct order of provisioning and configuration. Many configuration parameters, such a the IP address of a virtual machine at a cloud provider will not be known upfront. Inmanta marks this parameters as **unknown**. The state of any resource that uses such an unknown parameter becomes undefined.

CHAPTER

ELEVEN

INMANTA REFERENCE

Welcome to the Inmanta reference guide!

Here we explain all the features and options of Inmanta. If you're just looking to get started with Inmanta, please check out the *Quickstart* guide.

11.1 Command Reference

All inmanta commands and services are started by the `inmanta` command. This page provides an overview of all subcommands available:

11.1.1 inmanta

```
usage: inmanta [-h] [-p] [-c CONFIG_FILE] [--config-dir CONFIG_DIR]
               [--log-file LOG_FILE] [--logging-config LOGGING_CONFIG]
               [--log-file-level {0,1,2,3,4,ERROR,WARNING,INFO,DEBUG,TRACE}]
               [--timed-logs] [-v] [--warnings {warn,ignore,error}] [-X]
               [--version] [--keep-logger-names]
               {server,agent,compile,list-commands,help,modules,module,project,deploy,
↪export}
               ...
```

Named Arguments

-p	Profile this run of the program Default: False
-c, --config	Use this config file
--config-dir	The directory containing the Inmanta configuration files Default: "/etc/inmanta/inmanta.d"
--log-file	Path to the logfile
--logging-config	The path to the configuration file for the logging framework. This is a YAML file that follows the dictionary-schema accepted by logging.config.dictConfig(). All other log-related configuration arguments will be ignored when this argument is provided.
--log-file-level	Possible choices: 0, 1, 2, 3, 4, ERROR, WARNING, INFO, DEBUG, TRACE Log level for messages going to the logfile: 0=ERROR, 1=WARNING, 2=INFO, 3=DEBUG Default: "INFO"

--timed-logs	Add timestamps to logs Default: False
-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace Default: 0
--warnings	Possible choices: warn, ignore, error The warning behaviour. Must be one of 'warn', 'ignore', 'error' Default: "warn"
-X, --extended-errors	Show stack traces for errors Default: False
--version	Show the version of the installed Inmanta product and the version of its sub-components Default: False
--keep-logger-names	Display the log messages using the name of the logger that created the log messages. Default: False

Sub-commands

server

Start the inmanta server

```
inmanta server [-h] [-v] [--db-wait-time DB_WAIT_TIME]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--db-wait-time	Maximum time in seconds the server will wait for the database to be up before starting. A value of 0 means the server will not wait. If set to a negative value, the server will wait indefinitely.

agent

Start the inmanta agent

```
inmanta agent [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

compile

Compile the project to a configuration model

```
inmanta compile [-h] [-v] [-e ENVIRONMENT] [-X] [--server_address SERVER]
                [--server_port PORT] [--username USER] [--password PASSWORD]
                [--ssl] [--ssl-ca-cert CA_CERT] [--export-compile-data]
                [--export-compile-data-file EXPORT_COMPILE_DATA_FILE]
                [--no-cache] [--experimental-data-trace]
                [--experimental-dataflow-graphic] [-f MAIN_FILE]
                [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-e	The environment to compile this model for
-X, --extended-errors	Show stack traces for compile errors
--server_address	The address of the server hosting the environment
--server_port	The port of the server hosting the environment
--username	The username of the server
--password	The password of the server
--ssl	Enable SSL Default: False
--ssl-ca-cert	Certificate authority for SSL
--export-compile-data	Export structured json containing compile data such as occurred errors. Default: False
--export-compile-data-file	File to export compile data to. If omitted compile_data.json is used.
--no-cache	Disable caching of compiled CF files Default: True
--experimental-data-trace	Experimental data trace tool useful for debugging Default: False
--experimental-dataflow-graphic	Experimental graphic data flow visualization Default: False
-f	Main file Default: "main.cf"
--no-strict-deps-check	When this option is enabled, only version conflicts in the direct dependencies will result in an error. All other version conflicts will result in a warning. This option is mutually exclusive with the --strict-deps-check option. Default: False

--strict-deps-check	When this option is enabled, a version conflict in any (transitive) dependency will results in an error. This option is mutually exclusive with the --no-strict-deps-check option. Default: False

list-commands

Print out an overview of all commands

```
inmanta list-commands [-h]
```

help

show a help message and exit

```
inmanta help [-h] [subcommand]
```

Positional Arguments

subcommand	Output help for a particular subcommand

modules (module)

Subcommand to manage modules

```
inmanta modules [-h] [-v] [-m [MODULE]]
                {add,list,do,install,status,push,verify,commit,create,freeze,build,
↪v1tov2,release}
                ...
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-m, --module	Module to apply this command to

subcommand

cmd	Possible choices: add, list, do, install, status, push, verify, commit, create, freeze, build, v1tov2, release

Sub-commands

add

Add a module dependency to an Inmanta module or project. When executed on a project, the module is installed as well. Either --v1 or --v2 has to be set.

```
inmanta modules add [-h] [-v] [--v1] [--v2] [--override] module_req
```

Positional Arguments

module_req	The name of the module, optionally with a version constraint.

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--v1	Add the given module as a v1 module Default: False
--v2	Add the given module as a V2 module Default: False
--override	Override the version constraint when the given module dependency already exists. Default: False

list

List all modules used in this project in a table

```
inmanta modules list [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

do

Execute a command on all loaded modules

```
inmanta modules do [-h] [-v] command
```

Positional Arguments

command	the command to execute

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

install

The 'inmanta module install' command is no longer supported. Instead, use one of the following approaches:

1. To install a module in editable mode, use 'pip install -e .'.
2. For a non-editable installation, first run 'inmanta module build' followed by 'pip install ./dist/<dist-package>'.

```
inmanta modules install [-h] [-v] [-e] [path]
```

Positional Arguments

path	The path to the module.

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-e, --editable	Install in editable mode. Default: False

status

Run a git status on all modules and report

```
inmanta modules status [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

push

Run a git push on all modules and report

```
inmanta modules push [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

verify

Verify dependencies and frozen module versions

```
inmanta modules verify [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

commit

Commit all changes in the current module.

```
inmanta modules commit [-h] -m MESSAGE [-r] [--major] [--minor] [--patch]
                       [-v VERSION] [-a] [-t] [-n]
```

Named Arguments

-m, --message	Commit message
-r, --release	make a release Default: True
--major	make a major release Default: False
--minor	make a major release Default: False
--patch	make a major release Default: False
-v, --version	Version to use on tag
-a, --all	Use commit -a Default: False
-t, --tag	Create a tag for the commit.Tags are not created for dev releases by default, if you want to tag it, specify this flag explicitly Default: False

-n, --no-tag	Don't create a tag for the commit Default: False

create

Create a new module

```
inmanta modules create [-h] [-v] [--v1] name
```

Positional Arguments

name	The name of the module

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--v1	Create a v1 module. By default a v2 module is created. Default: False

freeze

Freeze all version numbers in module.yml. This command is only supported on v1 modules. On v2 modules use the pip freeze command instead.

```
inmanta modules freeze [-h] [-v] [-o OUTFILE] [-r] [--operator {==,~=,>=}]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-o, --outfile	File in which to put the new module.yml, default is the existing module.yml. Use - to write to stdout.
-r, --recursive	Freeze dependencies recursively. If not set, freeze_recursive option in module.yml is used, which defaults to False
--operator	Possible choices: ==, ~=, >= Comparison operator used to freeze versions, If not set, the freeze_operator option in module.yml is used which defaults to ~=

build

Build a Python package from a V2 module.

```
inmanta modules build [-h] [-v] [-o OUTPUT_DIR] [--dev] [-b] [path]
```

Positional Arguments

path	The path to the module that should be built. By default, the current working directory is used.

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-o, --output-dir	The directory where the Python package will be stored. Default: <module_root>/dist
--dev	Perform a development build of the module. This adds the build tag *.dev<timestamp>* to the package name. The timestamp has the form %Y%m%d%H%M%S. Default: False
-b, --byte-code	Produce a module wheel that contains only python bytecode for the plugins. Default: False

v1tov2

Convert a V1 module to a V2 module in place

```
inmanta modules v1tov2 [-h] [-v]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

release

When a stable release is done, this command:

- Does a commit that changes the current version to a stable version.
- Adds Git release tag.
- Does a commit that changes the current version to a development version that is one patch increment ahead of the released version.

When a development release is done using the --dev option, this command:

- Does a commit that updates the current version of the module to a development version that is a patch, minor or major version ahead of the previous stable release. The size of the increment is determined by the --revision, --patch, --minor or --major argument (--patch is the default). When a CHANGELOG.md file is present in the root of the module directory then the version number in the changelog is also updated accordingly. The changelog file is always populated with the associated stable version and not a development version.

```
inmanta modules release [-h] [-v] [--dev] [--major] [--minor] [--patch]
                        [--revision] [-m MESSAGE] [-c CHANGELOG_MESSAGE] [-a]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--dev	Create a development version. The new version number will have the .dev0 build tag. Default: False
--major	Do a major version bump compared to the previous stable release. Default: False
--minor	Do a minor version bump compared to the previous stable release. Default: False
--patch	Do a patch version bump compared to the previous stable release. Default: False
--revision	Do a revision version bump compared to the previous stable release (only with 4 digits version). Default: False
-m, --message	Commit message
-c, --changelog-message	This changelog message will be written to the changelog file. If the -m option is not provided, this message will also be used as the commit message.
-a, --all	Use commit -a Default: False

project

Subcommand to manage the project

```
inmanta project [-h] [-v] {freeze,init,install,update} ...
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace

subcommand

cmd	Possible choices: freeze, init, install, update

Sub-commands

freeze

Set all version numbers in project.yml

```
inmanta project freeze [-h] [-v] [-o OUTFILE] [-r] [--operator {==,~=,>=}]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-o, --outfile	File in which to put the new project.yml, default is the existing project.yml. Use - to write to stdout.
-r, --recursive	Freeze dependencies recursively. If not set, freeze_recursive option in project.yml is used,which defaults to False
--operator	Possible choices: ==, ~=, >= Comparison operator used to freeze versions, If not set, the freeze_operator option in project.yml is used which defaults to ~=

init

Initialize directory structure for a project

```
inmanta project init [-h] [-v] --name NAME [--output-dir OUTPUT_DIR]
                     [--default]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--name, -n	The name of the new project
--output-dir, -o	Output directory path Default: "./"
--default	Use default parameters for the project generation Default: False

install

Install all modules required for this project.

This command installs missing modules in the development venv, but doesn't update already installed modules if that's not required to satisfy the module version constraints. Use *inmanta project update* instead if the already installed modules need to be updated to the latest compatible version.

This command might reinstall Python packages in the development venv if the currently installed versions are not compatible with the dependencies specified by the different Inmanta modules.

```
inmanta project install [-h] [-v] [--no-strict-deps-check]
                        [--strict-deps-check]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--no-strict-deps-check	When this option is enabled, only version conflicts in the direct dependencies will result in an error. All other version conflicts will result in a warning. This option is mutually exclusive with the --strict-deps-check option. Default: False
--strict-deps-check	When this option is enabled, a version conflict in any (transitive) dependency will results in an error. This option is mutually exclusive with the --no-strict-deps-check option. Default: False

update

Update all modules to the latest version compatible with the module version constraints and install missing modules.

This command might reinstall Python packages in the development venv if the currently installed versions are not the latest compatible with the dependencies specified by the updated modules.

```
inmanta project update [-h] [-v] [--no-strict-deps-check]
                       [--strict-deps-check]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--no-strict-deps-check	When this option is enabled, only version conflicts in the direct dependencies will result in an error. All other version conflicts will result in a warning. This option is mutually exclusive with the --strict-deps-check option. Default: False
--strict-deps-check	When this option is enabled, a version conflict in any (transitive) dependency will results in an error. This option is mutually exclusive with the --no-strict-deps-check option. Default: False

deploy

Deploy with a inmanta all-in-one setup

```
inmanta deploy [-h] [-v] [--dry-run] [-f MAIN_FILE]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
--dry-run	Only report changes Default: False
-f	Main file Default: "main.cf"

export

Export the configuration

```
inmanta export [-h] [-v] [-g] [-j JSON] [-e ENVIRONMENT] [-d] [--full] [-m]
               [--server_address SERVER] [--server_port PORT] [--token TOKEN]
               [--ssl | --no-ssl] [--ssl-ca-cert CA_CERT] [-X] [-f MAIN_FILE]
               [--metadata METADATA] [--model-export]
               [--export-plugin EXPORT_PLUGIN] [--export-compile-data]
               [--export-compile-data-file EXPORT_COMPILE_DATA_FILE]
               [--no-cache] [--partial]
               [--delete-resource-set DELETE_RESOURCE_SET] [--soft-delete]
               [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

-v, --verbose	Log level for messages going to the console. Default is warnings,-v warning, -vv info, -vvv debug and -vvvv trace
-g	Dump the dependency graph Default: False
-j	Do not submit to the server but only store the json that would have been submitted in the supplied file
-e	The environment to compile this model for
-d	Trigger a deploy for the exported version Default: False
--full	Make the agents execute a full deploy instead of an incremental deploy. Should be used together with the -d option Default: False
-m	Also export the complete model Default: False
--server_address	The address of the server to submit the model to

--server_port	The port of the server to submit the model to
--token	The token to auth to the server
--ssl, --no-ssl	Enable SSL
--ssl-ca-cert	Certificate authority for SSL
-X, --extended-errors	Show stack traces for compile errors
-f	Main file Default: "main.cf"
--metadata	JSON metadata why this compile happened. If a non-json string is passed it is used as the 'message' attribute in the metadata.
--model-export	Export the configuration model to the server as metadata. Default: False
--export-plugin	Only use this export plugin. This option also disables the execution of the plugins listed in the configuration file in the export setting.
--export-compile-data	Export structured json containing compile data such as occurred errors. Default: False
--export-compile-data-file	File to export compile data to. If omitted compile_data.json is used.
--no-cache	Disable caching of compiled CF files Default: True
--partial	Execute a partial export. Does not upload new Python code to the server: it is assumed to be unchanged since the last full export. Multiple partial exports for disjunct resource sets may be performed concurrently but not concurrent with a full export. When used in combination with the *–json* option, 0 is used as a placeholder for the model version. Default: False
--delete-resource-set	Remove a resource set as part of a partial compile. This option can be provided multiple times and should always be used together with the –partial option.
--soft-delete	Use in combination with –delete-resource-set to delete these resource sets only if they are not being exported Default: False
--no-strict-deps-check	When this option is enabled, only version conflicts in the direct dependencies will result in an error. All other version conflicts will result in a warning. This option is mutually exclusive with the --strict-deps-check option. Default: False
--strict-deps-check	When this option is enabled, a version conflict in any (transitive) dependency will results in an error. This option is mutually exclusive with the --no-strict-deps-check option. Default: False

11.1.2 inmanta-cli

The `inmanta-cli` command can be used to interact with the inmanta server and agents, including managing projects, environments, parameters and more. The following reference explains the available subcommands.

inmanta-cli

Base command

```
inmanta-cli [OPTIONS] COMMAND [ARGS]...
```

Options

--host <host>

The server hostname to connect to

--port <port>

The server port to connect to

action-log

Subcommand to view the resource action log

```
inmanta-cli action-log [OPTIONS] COMMAND [ARGS]...
```

list

List the resource action log for a specific Resource.

```
inmanta-cli action-log list [OPTIONS]
```

Options

-e, --environment <environment>

Required The ID or name of the environment to use

--rvid <rvid>

Required The resource version ID of the resource

--action <action>

Only list this resource action

Options

store | push | pull | deploy | dryrun | getfact | other

show-messages

Show the log messages for a specific entry in the resource action log.

```
inmanta-cli action-log show-messages [OPTIONS]
```

Options

-e, --environment <environment>
  Required The ID or name of the environment to use

--rvid <rvid>
  Required The resource version ID of the resource

--action-id <action_id>
  Required The ID of the resource action record

agent

Subcommand to manage agents

```
inmanta-cli agent [OPTIONS] COMMAND [ARGS]...
```

list

List agents in an environment

```
inmanta-cli agent list [OPTIONS]
```

Options

-e, --environment <environment>
  Required The environment to use

pause

Pause a specific agent or all agents in a given environment. A paused agent cannot execute deploy operations.

```
inmanta-cli agent pause [OPTIONS]
```

Options

-e, --environment <environment>
  Required The environment to use

--agent <agent>
  The name of the agent to pause.

--all
  Pause all agents in the given environment

unpause

Unpause a specific agent or all agents in a given environment. A unpaused agent will be able to execute deploy operations.

```
inmanta-cli agent unpause [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

--agent <agent>
    The name of the agent to unpause.

--all
    Unpause all agents in the given environment

environment

Subcommand to manage environments

```
inmanta-cli environment [OPTIONS] COMMAND [ARGS]...
```

create

Create a new environment

```
inmanta-cli environment create [OPTIONS]
```

Options

-n, --name <name>
    Required The name of the new environment. The name should be unique for each project.

-p, --project <project>
    Required The id of the project this environment belongs to

-r, --repo-url <repo_url>
    The url of the repository that contains the configuration model

-b, --branch <branch>
    The branch in the repository that contains the configuration model

-s, --save
    Save the ID of the environment and the server to the .inmanta config file

delete

Delete an existing environment

ENVIRONMENT: ID or name of the environment to delete

```
inmanta-cli environment delete [OPTIONS] ENVIRONMENT
```

Arguments

ENVIRONMENT
Required argument

list

List all environments

```
inmanta-cli environment list [OPTIONS]
```

modify

Modify an existing environment

ENVIRONMENT: ID or name of the environment to modify

```
inmanta-cli environment modify [OPTIONS] ENVIRONMENT
```

Options

-n, --name <name>
Required The name of the new environment

-r, --repo-url <repo_url>
The url of the repository that contains the configuration model

-b, --branch <branch>
The branch in the repository that contains the configuration model

Arguments

ENVIRONMENT
Required argument

recompile

Request the server to recompile the model of this environment.

ENVIRONMENT: ID or name of the environment to trigger the recompile for

```
inmanta-cli environment recompile [OPTIONS] ENVIRONMENT
```

Options

-u, --update

Update the model and its dependencies before recompiling

Default
`False`

Arguments

ENVIRONMENT

Required argument

save

Save the ID of the environment and the server to the .inmanta config file

ENVIRONMENT: ID or name of the environment to write the config for

```
inmanta-cli environment save [OPTIONS] ENVIRONMENT
```

Arguments

ENVIRONMENT

Required argument

setting

Subcommand to manage environment settings

```
inmanta-cli environment setting [OPTIONS] COMMAND [ARGS]...
```

delete

Delete an environment setting

```
inmanta-cli environment setting delete [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

-k, --key <key>
    Required The key to delete

get

Get an environment setting

```
inmanta-cli environment setting get [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

-k, --key <key>
    Required The key to get

list

List settings of an environment

```
inmanta-cli environment setting list [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

set

Adjust an environment setting

```
inmanta-cli environment setting set [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

-k, --key <key>
    Required The key to set

-o, --value <value>
    Required The value to set

show

Show details of an environment

ENVIRONMENT: ID or name of the environment to show

```
inmanta-cli environment show [OPTIONS] ENVIRONMENT
```

Options

--format <format_string>

Instead of outputting a table, use the supplied format string. Accepts Python format syntax. Supported fields are 'id', 'name', 'project', 'repo_url', 'repo_branch'

Arguments

ENVIRONMENT

Required argument

lsm

Command to execute action on Inmanta lsm

```
inmanta-cli lsm [OPTIONS] COMMAND [ARGS]...
```

Options

--host <host>

The server hostname to connect to

--port <port>

The server port to connect to

resources

Subcommand to manage resources

```
inmanta-cli lsm resources [OPTIONS] COMMAND [ARGS]...
```

list

List resources associated with a service instance

```
inmanta-cli lsm resources list [OPTIONS]
```

Options

-e, --environment <environment>
  Required The environment to use

-s, --service-entity <service_entity>
  Required The name of the entity the instance belongs to

-i, --instance <instance>
  Required The instance id

-v, --version <version>
  Required Current version of the instance

service-entities

Subcommand to manage service entities

```
inmanta-cli lsm service-entities [OPTIONS] COMMAND [ARGS]...
```

list

List all entities

```
inmanta-cli lsm service-entities list [OPTIONS]
```

Options

-e, --environment <environment>
  Required The environment to use

service-instances

Subcommand to manage service instances

```
inmanta-cli lsm service-instances [OPTIONS] COMMAND [ARGS]...
```

list

List all instances of an entity

```
inmanta-cli lsm service-instances list [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

-s, --service-entity <service_entity>
    Required The name of the entity to list instances of

monitor

Monitor the deployment process of the configuration model in an environment, receiving continuous updates on the deployment status

```
inmanta-cli monitor [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

param

Subcommand to manage parameters

```
inmanta-cli param [OPTIONS] COMMAND [ARGS]...
```

get

Get a parameter from an environment

```
inmanta-cli param get [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

--name <name>
    Required The name of the parameter

--resource <resource>
    The resource id of the parameter

list

List parameters in an environment

```
inmanta-cli param list [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

set

Set a parameter in an environment

```
inmanta-cli param set [OPTIONS]
```

Options

-e, --environment <environment>
    Required The environment to use

--name <name>
    Required The name of the parameter

--value <value>
    Required The value of the parameter

project

Subcommand to manage projects

```
inmanta-cli project [OPTIONS] COMMAND [ARGS]...
```

create

Create a new project on the server

```
inmanta-cli project create [OPTIONS]
```

Options

-n, --name <name>
    Required The name of the new project

delete

Delete an existing project.

PROJECT: The id or name of the project to delete

```
inmanta-cli project delete [OPTIONS] PROJECT
```

Arguments

PROJECT
Required argument

list

List all projects

```
inmanta-cli project list [OPTIONS]
```

modify

Modify an existing project.

PROJECT: The id or name of the project to modify

```
inmanta-cli project modify [OPTIONS] PROJECT
```

Options

-n, --name <name>
Required The new name of the project

Arguments

PROJECT
Required argument

show

Show the details of a single project

PROJECT: The id or name of the project to show

```
inmanta-cli project show [OPTIONS] PROJECT
```

Arguments

PROJECT
Required argument

token

Subcommand to manage access tokens

```
inmanta-cli token [OPTIONS] COMMAND [ARGS]...
```

bootstrap

Generate a bootstrap token that provides access to everything. This token is only valid for 3600 seconds.

```
inmanta-cli token bootstrap [OPTIONS]
```

create

Create a new token for an environment for the specified client types

```
inmanta-cli token create [OPTIONS]
```

Options

-e, --environment <environment>
Required The environment to use.

--api
Add client_type api to the token.

--compiler
Add client_type compiler to the token.

--agent
Add client_type agent to the token.

version

Subcommand to manage versions

```
inmanta-cli version [OPTIONS] COMMAND [ARGS]...
```

list

List versions in an environment

```
inmanta-cli version list [OPTIONS]
```

Options

-e, --environment <environment>
: **Required** The environment to use

release

Release the specified version of the configuration model for deployment.

VERSION: Version of the model to release

```
inmanta-cli version release [OPTIONS] VERSION
```

Options

-e, --environment <environment>
: **Required** The environment to use

-p, --push
: Push the version to the deployment agents

--full
: Make the agents execute a full deploy instead of an incremental deploy. Should be used together with the –push option

Arguments

VERSION
: Required argument

report

Get a report about a version, describing the involved resources, agents and actions

```
inmanta-cli version report [OPTIONS]
```

Options

-e, --environment <environment>
: **Required** The environment to use

-i, --version <version>
: **Required** The version to create a report from

-l
: Show a detailed version of the report

11.2 Configuration Reference

This document lists all options for the inmanta server and inmanta agent.

The options are listed per config section.

11.2.1 agent

`executor-mode`

Type
threaded | forking

Default
`AgentExcutorMode.threaded`

EXPERIMENTAL: set the agent to use threads or fork subprocesses to create workers.

11.2.2 agent_rest_transport

`host`

Type
str

Default
`localhost`

IP address or hostname of the server

`max-clients`

Type
optional int

Default
`None`

The maximum number of simultaneous connections that can be open in parallel

`port`

Type
int

Default
`8888`

Server port

`request-timeout`

Type
int

Default
`120`

The time before a request times out in seconds

`ssl`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Connect using SSL?

`ssl-ca-cert-file`

Type
optional str

Default
`None`

CA cert file used to validate the server certificate against

`token`

Type
optional str

Default
`None`

The bearer token to use to connect to the API

11.2.3 client_rest_transport

`host`

Type
str

Default
`localhost`

IP address or hostname of the server

`max-clients`

Type
optional int

Default
`None`

The maximum number of simultaneous connections that can be open in parallel

`port`

Type
int

Default
`8888`

Server port

`request-timeout`

Type
int

Default
`120`

The time before a request times out in seconds `ssl`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Connect using SSL?

`ssl-ca-cert-file`

Type
optional str

Default
`None`

CA cert file used to validate the server certificate against

`token`

Type
optional str

Default
`None`

The bearer token to use to connect to the API

11.2.4 cmdline_rest_transport

`host`

Type
str

Default
`localhost`

IP address or hostname of the server

`max-clients`

Type
optional int

Default
`None`

The maximum number of simultaneous connections that can be open in parallel

`port`

Type
int

Default
`8888`

Server port

`request-timeout`

Type
int

Default
`120`

The time before a request times out in seconds

`ssl`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Connect using SSL?

`ssl-ca-cert-file`

Type
optional str

Default
`None`

CA cert file used to validate the server certificate against

`token`

Type
optional str

Default
`None`

The bearer token to use to connect to the API

11.2.5 compiler

`cache`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`True`

Enables the caching of compiled files.

`dataflow-graphic-enable`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Enables graphic visualization of the data flow in the model. Requires the datatrace_enable option. Requires graphviz.

`datatrace-enable`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Enables the experimental datatrace application on top of the compiler. The application should help in identifying the cause of compilation errors during the development process.

`export-compile-data`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Export structured json containing compile data such as occurred errors.

`export-compile-data-file`

Type
str

Default
`compile_data.json`

File to export compile data to. If omitted compile_data.json is used.

11.2.6 compiler_rest_transport

`host`

Type
str

Default
`localhost`

IP address or hostname of the server

`max-clients`

Type
optional int

Default
`None`

The maximum number of simultaneous connections that can be open in parallel

`port`

Type
int

Default
`8888`

Server port

`request-timeout`

Type
int

Default
`120`

The time before a request times out in seconds

`ssl`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Connect using SSL?

`ssl-ca-cert-file`

Type
optional str

Default
`None`

CA cert file used to validate the server certificate against

`token`

Type
optional str

Default
`None`

The bearer token to use to connect to the API

11.2.7 config

`agent-deploy-interval`

Type
Time, the number of seconds represented as an integer value or a cron-like expression

Default
`0`

Either the number of seconds between two (incremental) deployment runs of the agent or a cron-like expression. If a cron-like expression is specified, a deploy will be run following a cron-like time-to-run specification, interpreted in UTC. The expected format is *[sec] min hour dom month dow [year]* (If only 6 values are provided, they are interpreted as *min hour dom month dow year*). A deploy will be requested at the scheduled time. Note that if a cron expression is used the 'agent_deploy_splay_time' setting will be ignored. Set this to 0 to disable the scheduled deploy runs.

`agent-deploy-splay-time`

Type
Time, the number of seconds represented as an integer value

Default
`600`

The splaytime added to the agent-deploy-interval. Set this to 0 to disable the splaytime.

At startup the agent will choose a random number between 0 and agent-deploy-splay-time. It will wait this number of second before performing the first deployment run. Each subsequent repair deployment will start agent-deploy-interval seconds after the previous one.

`agent-get-resource-backoff`

Type
float

Default
`3`

This is a load management feature. It ensures that the agent will not pull resources from the inmanta server <agent-get-resource-backoff>*<duration-last-pull-in-seconds> seconds after the last time the agent pulled resources from the server. Setting this option too low may result in a high load on the Inmanta server. Setting it too high may result in long deployment times.

`agent-interval`

Type
Time, the number of seconds represented as an integer value

Default
`600`

[DEPRECATED] The run interval of the agent. Every run-interval seconds, the agent will check the current state of its resources against to desired state model

`agent-map`

Type
List of comma-separated key=value pairs

Default
dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = { } for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2)

By default the agent assumes that all agent names map to the host on which the process is executed. With the agent map it can be mapped to other hosts. This value consists of a list of key/value pairs. The key is the name of the agent and the format of the value is described in *`std::AgentConfig`*. When the configuration option config.use_autostart_agent_map is set to true, this option will be ignored.

example: iaas_openstack=localhost,vm1=192.16.13.2

`agent-names`

Type
List of comma-separated values

Default
None

Names of the agents this instance should deploy configuration for. When the configuration option config.use_autostart_agent_map is set to true, this option will be ignored.

`agent-reconnect-delay`

Type
int

Default
`5`

Time to wait after a failed heartbeat message. DO NOT SET TO 0

`agent-repair-interval`

Type
Time, the number of seconds represented as an integer value or a cron-like expression

Default
`600`

Either the number of seconds between two repair runs (full deploy) of the agent or a cron-like expression. If a cron-like expression is specified, a repair will be run following a cron-like time-to-run specification, interpreted in UTC. The expected format is *[sec] min hour dom month dow [year]* (If only 6 values are provided, they are interpreted as *min hour dom month dow year*). A repair will be requested at the scheduled time. Note that if a cron expression is used the 'agent_repair_splay_time' setting will be ignored. Setting this to 0 to disable the scheduled repair runs.

`agent-repair-splay-time`

Type
Time, the number of seconds represented as an integer value

Default
`600`

The splaytime added to the agent-repair-interval. Set this to 0 to disable the splaytime.

At startup the agent will choose a random number between 0 and agent-repair-splay-time. It will wait this number of second before performing the first repair run. Each subsequent repair deployment will start agent-repair-interval seconds after the previous one. This option is ignored and a splay of 0 is used if 'agent_repair_interval' is a cron expression

`agent-splay`

Type
Time, the number of seconds represented as an integer value

Default
`600`

[DEPRECATED] The splaytime added to the runinterval. Set this to 0 to disable splaytime. At startup the agent will choose a random number between 0 and "agent_splay. It will wait this number of second before performing the first deploy. Each subsequent deploy will start agent-interval seconds after the previous one.

`environment`

Type
optional uuid

Default
`None`

The environment this model is associated with

`export`

Type
List of comma-separated values

Default

The list of exporters to use. This option is ignored when the –export-plugin option is used.

`feature-file`

Type
optional str

Default
`None`

The loacation of the inmanta feature file.

`log-dir`

Type
str

Default
`/var/log/inmanta`

The directory where the resource action log is stored and the logs of auto-started agents.

`logging-config`

Type
optional str

Default
`None`

The path to the configuration file for the logging framework. This is a YAML file that follows the dictionary-schema accepted by logging.config.dictConfig(). All other log-related configuration options will be ignored when this option is set.

`node-name`

Type
str

Default
`socket.gethostname()`

Force the hostname of this machine to a specific value

`server-timeout`

Type
Time, the number of seconds represented as an integer value

Default
`125`

Amount of time to wait for a response from the server before we try to reconnect, must be larger than server.agent-hold

`state-dir`

Type
str

Default
`/var/lib/inmanta`

The directory where the server stores its state

`use-autostart-agent-map`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

If this option is set to true, the agent-map of this agent will be set the the autostart_agent_map configured on the server. The agent_map will be kept up-to-date automatically.

11.2.8 database

`connection-pool-max-size`

Type
int

Default
`70`

Max number of connections in the pool

`connection-pool-min-size`

Type
int

Default
`10`

Number of connections the pool will be initialized with

`connection-timeout`

Type
float

Default
`60`

Connection timeout in seconds

`host`

Type
str

Default
`localhost`

Hostname or IP of the postgresql server

`name`

Type
str

Default
`inmanta`

The name of the database on the postgresql server

`password`

Type
str

Default
`None`

The password that belong to the database user

`port`

Type
int

Default
`5432`

The port of the postgresql server

`username`

Type
str

Default
`postgres`

The username to access the database in the PostgreSQL server `wait-time`

Type
Time, the number of seconds represented as an integer value

Default
`0`

For how long the server should wait for the DB to be up before starting. If set to 0, the server won't wait for the DB. If set to a negative value, the server will wait forever.

11.2.9 deploy

`environment`

Type
optional str

Default
`deploy`

The environment name to use in the deploy

`project`

Type
optional str

Default
`deploy`

The project name to use in the deploy

11.2.10 influxdb

`host`

Type
str

Default

Hostname or IP of the influxdb server to send reports to

`interval`

Type
int

Default
`30`

Interval with which to report to influxdb

`name`

Type
str

Default
`inmanta`

The name of the database on the influxdb server `password`

Type
str

Default
`None`

The password that belong to the influxdb user

`port`

Type
int

Default
`8086`

The port of the influxdb server

`tags`

Type
List of comma-separated key=value pairs

Default

a dict of tags to attach to all influxdb records in the form tag=value,tag=value

`username`

Type
str

Default
`None`

The username to access the database in the influxdb server

11.2.11 license

`entitlement-file`

Type
str

Default
`/etc/inmanta/entitlement.jwe`

The entitlement file to enable features in orchestrator.

`license-key`

Type
str

Default
`/etc/inmanta/license.key`

The license file to activate the orchestrator.

11.2.12 lsm.callback

`logfile`

Type
str

Default
`callback.log`

Log file for callbacks

`timeout`

Type
float

Default
`30.0`

The request timeout for event notification callbacks in seconds

11.2.13 server

`access-control-allow-origin`

Type
optional str

Default
`None`

Configures the Access-Control-Allow-Origin setting of the http server.Defaults to not sending an Access-Control-Allow-Origin header.

`agent-hold`

Type
Time, the number of seconds represented as an integer value

Default
`server.agent-timeout` *3/4

Maximal time the server will hold an agent heartbeat call

`agent-process-purge-interval`

Type
Time, the number of seconds represented as an integer value

Default
`3600`

The number of seconds between two purges of old and expired agent processes. Set to zero to disable the cleanup. see *`server.agent-processes-to-keep`*

`agent-processes-to-keep`

Type
int

Default
`5`

Keep this amount of expired agent processes for a certain hostname `agent-timeout`

Type
Time, the number of seconds represented as an integer value

Default
`30`

Time before an agent is considered to be offline

`auth`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`False`

Enable authentication on the server API

`auth-method`

Type
str

Default
`oidc`

The authentication method to use: oidc or database

`auto-recompile-wait`

Type
Time, the number of seconds represented as an integer value

Default
`10`

DEPRECATED: The number of seconds to wait before the server may attempt to do a new recompile. Recompiles are triggered after facts updates for example.

`bind-address`

Type
List of comma-separated values

Default
`127.0.0.1`

A list of addresses on which the server will listen for connections. If this option is set, the *`server_rest_transport.port`* option is ignored.

`bind-port`

Type
int

Default
`8888`

The port on which the server will listen for connections. If this option is set, the *`server_rest_transport.port`* option is ignored.

`cleanup-compiler-reports-interval`

Type
Time, the number of seconds represented as an integer value

Default
`3600`

Number of seconds between old compile report cleanups. see *server.compiler-report-retention*

`compiler-report-retention`

Type
Time, the number of seconds represented as an integer value

Default
`604800`

The server regularly cleans up old compiler reports. This options specifies the number of seconds to keep old compiler reports for. The default is seven days.

`enabled-extensions`

Type
List of comma-separated values

Default
Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

A list of extensions the server must load. Core is always loaded.If an extension listed in this list is not available, the server will refuse to start.

`fact-expire`

Type
Time, the number of seconds represented as an integer value

Default
`3600`

After how many seconds will discovered facts/parameters expire.

`fact-renew`

Type
time; < *server.fact-expire*

Default
server.fact-expire /3

After how many seconds will discovered facts/parameters be renewed? This value needs to be lower than fact-expire

`fact-resource-block`

Type
Time, the number of seconds represented as an integer value

Default
`60`

Minimal time between subsequent requests for the same fact

`purge-resource-action-logs-interval`

Type
Time, the number of seconds represented as an integer value

Default
`3600`

The number of seconds between resource-action log purging `purge-versions-interval`

Type
Time, the number of seconds represented as an integer value

Default
`3600`

The number of seconds between version purging, see *`available_versions_to_keep`*.

`resource-action-log-prefix`

Type
str

Default
`resource-actions-`

File prefix in log-dir, containing the resource-action logs. The after the prefix the environment uuid and .log is added

`server-address`

Type
str

Default
`localhost`

The public ip address of the server. This is required for example to inject the inmanta agent in virtual machines at boot time.

`ssl-ca-cert-file`

Type
optional str

Default
`None`

The CA cert file required to validate the server ssl cert. This setting is used by the serverto correctly configure the compiler and agents that the server starts itself. If not set and SSL is enabled, the server cert should be verifiable with the CAs installed in the OS.

`ssl-cert-file`

Type
optional str

Default
`None`

SSL certificate file for the server key. Leave blank to disable SSL

`ssl-key-file`

Type
optional str

Default
`None`

Server private key to use for this server Leave blank to disable SSL

`tz-aware-timestamps`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`True`

Whether the server should return timezone aware timestamps. If False, the server will serialize timestamps in a time zone naive way (in implicit UTC). If True, timestamps are serialized as time zone aware objects.

`wait-after-param`

Type
Time, the number of seconds represented as an integer value

Default
`5`

Time to wait before recompile after new paramters have been received

11.2.14 server_rest_transport

`port`

Type
int

Default
`8888`

[DEPRECATED USE *`server.bind-port`*] The port on which the server listens for connections

11.2.15 unknown_handler

`default`

Type
str

Default
`prune-agent`

default method to handle unknown values

11.2.16 web-ui

`console-enabled`

Type
Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default
`True`

Whether the server should host the web-console or not

`console-path`

Type
str

Default
`/usr/share/inmanta/web-console`

The path on the local file system where the web-console can be found `features`

Type
List of comma-separated values

Default

A list of features that should be enabled in the web console.

`oidc-auth-url`

Type
str

Default
`None`

The auth url of the OpenID Connect server to use.

`oidc-client-id`

Type
str

Default
`None`

The OpenID Connect client id configured for this application.

`oidc-realm`

Type
str

Default
`inmanta`

The realm to use for OpenID Connect authentication.

11.3 Environment Settings Reference

This document lists all settings that can be set per environment. These changes are made through the API, the web-console or the CLI tool.

The supported settings are:

`agent_trigger_method_on_auto_deploy`

Type
enum: push_incremental_deploy, push_full_deploy

Default
push_incremental_deploy

The agent trigger method to use when push_on_auto_deploy is enabled

`auto_deploy`

Type
bool

Default
True

When this boolean is set to true, the orchestrator will automatically release a new version that was compiled by the orchestrator itself.

`auto_full_compile`

Type
str

Default
‘’

Periodically run a full compile following a cron-like time-to-run specification interpreted in UTC with format *[sec] min hour dom month dow [year]* (If only 6 values are provided, they are interpreted as *min hour dom month dow year*). A compile will be requested at the scheduled time. The actual compilation may have to wait in the compile queue for some time, depending on the size of the queue and the RECOMPILE_BACKOFF environment setting. This setting has no effect when server_compile is disabled.

`autostart_agent_deploy_interval`

Type
str

Default
600

The deployment interval of the autostarted agents. Can be specified as a number of seconds or as a cron-like expression. See also: *`config.agent-deploy-interval`*

`autostart_agent_deploy_splay_time`

Type
int

Default
10

The splay time on the deployment interval of the autostarted agents. See also: *`config.agent-deploy-splay-time`*

`autostart_agent_map`

Type
dict

Default
{‘internal’: ‘local:’}

A dict with key the name of agents that should be automatically started. The value is either an empty string or an agent map string. See also: *`config.agent-map`*

`autostart_agent_repair_interval`

Type
str

Default
86400

The repair interval of the autostarted agents. Can be specified as a number of seconds or as a cron-like expression. See also: *`config.agent-repair-interval`*

`autostart_agent_repair_splay_time`

Type
int

Default
600

The splay time on the repair interval of the autostarted agents. See also: *`config.agent-repair-splay-time`* `autostart_on_start`

Type
bool

Default
True

Automatically start agents when the server starts instead of only just in time.

`available_versions_to_keep`

Type
int

Default
100

The number of versions to keep stored in the database, excluding the latest released version.

`enable_batched_partial_compiles`

Type
bool

Default
False

Allow LSM to perform updates to service instances using a single partial compile, instead of doing a separate compile per update and per service instance.

`enable_lsm_expert_mode`

Type
bool

Default
False

This setting enables lsm expert mode. When enabled, it will be possible to use the LSM expert mode API endpoints. This bypass many of the safety checks done by the Inmanta server. Use with caution.

`environment_agent_trigger_method`

Type
enum: push_incremental_deploy, push_full_deploy

Default
push_incremental_deploy

The agent trigger method to use when no specific method is specified in the API call. This determines the behavior of the 'Promote' button. For auto deploy, agent_trigger_method_on_auto_deploy is used.

`environment_metrics_retention`

Type
int

Default
8760

The number of hours that environment metrics have to be retained before they are cleaned up. Default=8760 hours (1 year). Set to 0 to disable automatic cleanups.

`lsm_partial_compile`

Type
bool

Default
False

When this boolean is set to true, the compilation will be done using partial compiles, else full compiles will be done

`notification_retention`

Type
int

Default
365

The number of days to retain notifications for

`protected_environment`

Type
bool

Default
False

When set to true, this environment cannot be cleared or deleted.

`push_on_auto_deploy`

Type
bool

Default
True

Push a new version when it has been autodeployed.

`recompile_backoff`

Type
positive_float

Default
0.1

The number of seconds to wait before the server may attempt to do a new recompile. Recompiles are triggered after facts updates for example.

`resource_action_logs_retention`

Type
int

Default
7

The number of days to retain resource-action logs

`server_compile`

Type
bool

Default
True

Allow the server to compile the configuration model.

11.4 Compiler Configuration Reference

11.4.1 project.yml

Inside any project the compiler expects a `project.yml` file that defines metadata about the project, the location to store modules, repositories where to find modules and possibly specific versions of modules.

For basic usage information, see *Project creation guide*.

The `project.yml` file defines the following settings:

`class inmanta.module.ProjectMetadata`(**, requires: list[str] = [], name: str, description: str | None = None, freeze_recursive: bool = False, freeze_operator: str = '~=', author: str | None = None, author_email: NameEmail | None = None, license: str | None = None, copyright: str | None = None, modulepath: list[str] = [], repo: list[ModuleRepoInfo] = [], downloadpath: str | None = None, install_mode:* InstallMode *= InstallMode.release, relation_precedence_policy: list[str] = [], strict_deps_check: bool = True, agent_install_dependency_modules: bool = True, pip:* ProjectPipConfig *= ProjectPipConfig(index_url=None, extra_index_url=[], pre=None, use_system_config=False)*)

Parameters

- **`name`** – The name of the project.
- **`description`** – (Optional) An optional description of the project
- **`author`** – (Optional) The author of the project
- **`author_email`** – (Optional) The contact email address of author
- **`license`** – (Optional) License the project is released under
- **`copyright`** – (Optional) Copyright holder name and date.
- **`modulepath`** – (Optional) This value is a list of paths where Inmanta should search for modules.
- **`downloadpath`** – (Optional) This value determines the path where Inmanta should download modules from repositories. This path is not automatically included in the modulepath!
- **`install_mode`** – (Optional) [DEPRECATED] This key was used to determine what version of a module should be selected when a module is downloaded. For more information see *`InstallMode`*. This should now be set via the `pre` option of the `pip` section.
- **`repo`** – (Optional) A list (a yaml list) of repositories where Inmanta can find modules. Inmanta tries each repository in the order they appear in the list. Each element of this list requires a `type` and a `url` field. The type field can have the following values:
  - git: When the type is set to git, the url field should contain a template of the Git repo URL. Inmanta creates the git repo url by formatting {} or {0} with the name of the module. If no formatter is present it appends the name of the module to the URL.
  - package: [DEPRECATED] Setting up pip indexes should be done via the `index_urls` option of the `pip` section. Refer to the *migration guide* for more information.

  The old syntax, which only defines a Git URL per list entry is maintained for backward compatibility.

- **requires** – (Optional) This key can contain a list (a yaml list) of version constraints for modules used in this project. Similar to the module, version constraints are defined using PEP440 syntax.
- **freeze_recursive** – (Optional) This key determines if the freeze command will behave recursively or not. If freeze_recursive is set to false or not set, the current version of all modules imported directly in the main.cf file will be set in project.yml. If it is set to true, the versions of all modules used in this project will be set in project.yml.
- **freeze_operator** – (Optional) This key determines the comparison operator used by the freeze command. Valid values are [==, ~=, >=]. *Default is '~='*
- **relation_precedence_policy** – [EXPERIMENTAL FEATURE] A list of rules that indicate the order in which the compiler should freeze lists. The following syntax should be used to specify a rule *<first-type>.<relation-name> before <then-type>.<relation-name>*. With this rule in place, the compiler will first freeze *first-type.relation-name* and only then *then-type.relation-name*.
- **strict_deps_check** – Determines whether the compiler or inmanta tools that install/update module dependencies, should check the virtual environment for version conflicts in a strict way or not. A strict check means that all transitive dependencies will be checked for version conflicts and that any violation will result in an error. When a non-strict check is done, only version conflicts in a direct dependency will result in an error. All other violations will only result in a warning message.
- **agent_install_dependency_modules** – [DEPRECATED] If true, when a module declares Python dependencies on other (v2) modules, the agent will install these dependency modules with pip. This option should only be enabled if the agent is configured with the appropriate pip related environment variables. The option allows to an extent for inter-module dependencies within handler code, even if the dependency module doesn't have any handlers that would otherwise be considered relevant for this agent. Care should still be taken when you use inter-module imports. The current code loading mechanism does not explicitly order reloads. A general guideline is to use qualified imports where you can (import the module rather than objects from the module). When this is not feasible, you should be aware of Python's reload semantics and take this into account when making changes to handler code. Another caveat is that if the dependency module does contain code that is relevant for the agent, it will be loaded like any other handler code and it will be this code that is imported by any dependent modules (though depending on the load order the very first import may use the version installed by pip). If at some point this dependency module's handlers cease to be relevant for this agent, its code will remain stale. Therefore this feature should not be depended on in transient scenarios like this.
- **pip** – A configuration section that holds information about the pip configuration that should be taken into account when installing Python packages (See: `inmanta.module.ProjectPipConfig` for more details).

`class inmanta.module.ProjectPipConfig`(**, index_url: str | None = None, extra_index_url: Sequence[str] = [], pre: bool | None = None, use_system_config: bool = False*)

Parameters

- **index_url** – one pip index url for this project.
- **extra_index_url** – additional pip index urls for this project. This is generally only recommended if all configured indexes are under full control of the end user to protect against dependency confusion attacks. See the pip install documentation and PEP 708 (draft) for more information.
- **pre** – allow pre-releases when installing Python packages, i.e. pip –pre. If null, behaves like false unless pip.use-system-config=true, in which case system config is respected.

- `use_system_config` – defaults to false. When true, sets the pip's index url, extra index urls and pre according to the respective settings outlined above but otherwise respect any pip environment variables and/or config in the pip config file, including any extra-index-urls.

  If no indexes are configured in pip.index-url/pip.extra-index-url, this option falls back to pip's default behavior, meaning it uses the pip index url from the environment, the config file, or PyPi, in that order.

  For development, it is recommended to set this option to false, both for portability (and related compatibility with tools like pytest-inmanta-lsm) and for security (dependency confusion attacks could affect users that aren't aware that inmanta installs Python packages).

  See the *section* about setting up pip index for more information.

The code snippet below provides an example of a complete `project.yml` file:

```
name: quickstart
description: A quickstart project that installs a drupal website.
author: Inmanta
author_email: code@inmanta.com
license: Apache 2.0
copyright: Inmanta (2021)
modulepath: libs
downloadpath: libs
install_mode: release
repo:
  - url: https://github.com/inmanta/
    type: git
requires:
  - apache ~= 0.5.2
  - drupal ~= 0.7.3
  - exec ~= 1.1.4
  - ip ~= 1.2.1
  - mysql ~= 0.6.2
  - net ~= 1.0.5
  - php ~= 0.3.1
  - redhat ~= 0.9.2
  - std ~= 3.0.2
  - web ~= 0.3.3
  - yum ~= 0.6.2
freeze_recursive: true
freeze_operator: ~=
pip:
    index-url: https://pypi.org/simple/
    extra-index-url: []
    pre: false
    use-system-config: false
```

Configure pip index

This section explains how to configure a project-wide pip index. This index will be used to download v2 modules and v1 modules' dependencies. By default, a project created using the *Project creation guide* is configured to install packages from `https://pypi.org/simple/`. The *`ProjectPipConfig`* section of the project.yml file offers options to configure this behaviour. Some of these options are detailed below:

pip.use-system-config

This option determines the isolation level of the project's pip config. When false (the default), any pip config set on the system through pip config files is ignored, the `PIP_INDEX_URL`, `PIP_EXTRA_INDEX_URL` and `PIP_PRE` environment variables are ignored, and pip will only look for packages in the index(es) defined in the project.yml. When true, the orchestrator will use the system's pip configuration for the pip-related settings except when explicitly overriden in the `project.yml` (See below for more details).

Setting this to `false` is generally recommended, especially during development, both for portability (achieving consistent behavior regardless of the system it runs on, which is important for reproductive testing on developer machines, easy compatibility with Inmanta pytest extensions, and consistency between compiler and agents) and for security (the isolation reduces the risk of dependency confusion attacks).

Setting this to `true` will have the following consequences:

- If no index is set in the `project.yml` file i.e. both `index-url` and `extra-index-url` are unset, then Pip's default search behaviour will be used: environment variables, pip config files and then PyPi (in that order).
- If `index-url` is set, this value will be used over any index defined in the system's environment variables or pip config files.
- If `extra-index-url` is set, these indexes will be used in addition to any extra index defined in the system's environment variables or pip config files, and passed to pip as extra indexes.
- If `pre` is set, it will supersede pip's `pre` option set by the `PIP_PRE` environment variable or in pip

config file. When true, pre-release versions are allowed when installing v2 modules or v1 modules' dependencies.

- Auto-started agents live on the same host as the server, and so they will share the pip config at the system level.

Warning: `use-system-config = true` should only be used if the pip configuration is fully managed at the system level and secure for each component of the orchestrator.

Example scenario

1) During development

Using a single pip index isolated from any system config is the recommended approach. The `pre=true` option allows pip to use pre-release versions, e.g. when testing dev versions of modules published to the dev index. Here is an example of a dev config:

```yaml
pip:
    index-url: https://my_repo.secure.example.com/repository/dev
    extra-index-url: []
    pre: true
    use-system-config: false
```

2) In production

Using a single pip index is still the recommended approach, and the use of pre-release versions should be disabled.

For a portable project (recommended), disable `use-system-config` and set `index-url` to the secure internal repo e.g.:

```yaml
pip:
    index-url: https://my_repo.secure.example.com/repository/inmanta-production
    pre: false
    use-system-config: false
```

If you prefer to manage the pip configuration at the system level, use `use-system-config:  true` e.g.:

```yaml
pip:
    pre: false
    use-system-config: true
```

Note: Any pip config set explicitly in the project config will always take precedence over the system config. For more details see *pip.use-system-config*.

Pip-related settings that are not supported by the project config are not overridden.

To use a setting from the system's pip configuration without overriding it, leave the corresponding option unset in the `project.yml` file.

Note: Set up authentication towards the index using netrc. See this *section* for more information.

Migrate to project-wide pip config

This section is a migration guide for upgrading to `inmanta-service-orchestrator 7.0.0` or `inmanta 2024.0`. `inmanta-core 11.0.0` introduced new options to configure pip settings for the whole project in a centralized way. For detailed information, see *here*. The following code sample can be used as a baseline in the `project.yml` file:

```yaml
pip:
    index-url: https://my_repo.secure.example.com/repository/inmanta-production
    pre: false
    use-system-config: false
```

Alternatively, if you prefer to manage the pip config at the system level, refer to this *section*.

All the v2 module sources currently set in a `repo` section of the `project_yml` with type `package` should also be duplicated in the `pip.index-url` (and `pip.extra-index-url` if more than one index is being used).

If you want to allow pre-releases for v2 modules and other Python packages, set `pip.pre = true` in the project config file. This used to be controlled by the `InstallMode` set at the project level or at a module level.

Make sure the agents have access to the index(es) configured at the project level.

Run a full compile after upgrading in order to export the project pip config to the server, so that it is available for agents. This will ensure that the agents follow the pip config defined in the project. For reference, prior to `inmanta-core 11.0.0`, the agents were always using their respective system's pip config.

Breaking changes:

- Indexes defined through the `repo` option with type `package` will be ignored.
- Dependencies for v1 modules will now be installed according to the pip config in the project configuration file, while they previously always used the system's pip config.
- The agent will follow the pip configuration defined in the *project.yml*.
- `PIP_PRE` is now ignored unless `use-system-config` is set.
- Allowing the installation of pre-release versions for v2 modules through the `InstallMode` is no longer supported. Use the project.yml `pip.pre` section instead.

Changes relative to `inmanta-2023.4` (OSS):

- `pip.use_config_file` is refactored into `pip.use-system-config`.
- An error is now raised if `pip.use-system-config` is false and no "primary" index is set through `pip.index-url`.
- Pip environment variables are no longer ignored when `pip.use-system-config` is true and the corresponding option from the `project_yml` is unset.

11.4.2 Module metadata files

The metadata of a V1 module is present in the module.yml file. V2 modules keep their metadata in the setup.cfg file. Below sections describe each of these metadata files.

module.yml

Inside any V1 module the compiler expects a `module.yml` file that defines metadata about the module.

The `module.yml` file defines the following settings:

class inmanta.module.**ModuleMetadata**(**, name: str, description: str | None = None, freeze_recursive: bool = False, freeze_operator: str = '~=', version: str, license: str, deprecated: bool | None = None*)

The code snippet below provides an example of a complete `module.yml` file:

```
name: openstack
description: A module to manage networks, routers, virtual machine, etc. on an␣
↪Openstack cluster.
version: 3.7.1
license: Apache 2.0
compiler_version: 2020.2
requires:
  - ip
  - net
  - platform
  - ssh
  - std
freeze_recursive: false
freeze_operator: ~=
```

setup.cfg

Inside any V2 module the compiler expects a `setup.cfg` file that defines metadata about the module.

The code snippet below provides an example of a complete `setup.cfg` file:

```
[metadata]
name = inmanta-module-openstack
description = A module to manage networks, routers, virtual machine, etc. on an␣
↪Openstack cluster.
version = 3.7.1
license = Apache 2.0
compiler_version = 2020.2
freeze_recursive = false
freeze_operator = ~=

[options]
install_requires =
  inmanta-modules-ip
  inmanta-modules-net
  inmanta-modules-platform
  inmanta-modules-ssh
  inmanta-modules-std
```

11.5 Programmatic API reference

This page describes parts of inmanta code base that provide a stable API that could be used from modules or extensions.

Warning: Only those parts explicitly mentioned here are part of the API. They provide a stable interface. Other parts of the containing modules provide no such guarantees.

11.5.1 Constants

class inmanta.const.**LogLevel**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Log levels used for various parts of the inmanta orchestrator.

CRITICAL = 'CRITICAL'

DEBUG = 'DEBUG'

ERROR = 'ERROR'

INFO = 'INFO'

TRACE = 'TRACE'

WARNING = 'WARNING'

property to_int: int

class inmanta.const.**ResourceAction**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Enumeration of all resource actions.

deploy = 'deploy'

dryrun = 'dryrun'

getfact = 'getfact'

other = 'other'

pull = 'pull'

push = 'push'

store = 'store'

inmanta_lsm.const.**LSM_ENV_VARS : Sequence[str]**

This sequence contains all environment variables passed to the compiler by inmanta-lsm

11.5.2 Compiler exceptions

class inmanta.ast.**CompilerException**(*msg: str*)

Bases: `Exception`, `Exportable`

Base class for exceptions generated by the compiler

class inmanta.parser.**ParserException**(*location: Location*, *value: object*, *msg: str | None = None*)

Bases: *CompilerException*

Exception occurring during the parsing of the code

class inmanta.ast.**RuntimeException**(*stmt: Locatable | None*, *msg: str*)

Bases: *CompilerException*

Baseclass for exceptions raised by the compiler after parsing is complete.

class inmanta.ast.**ExternalException**(*stmt: Locatable | None*, *msg: str*, *cause: Exception*)

Bases: *RuntimeException*

When a plugin call produces an exception that is not a *RuntimeException*, it is wrapped in an ExternalException to make it conform to the expected interface

class inmanta.ast.**ExplicitPluginException**(*stmt: Locatable | None*, *msg: str*, *cause: PluginException*)

Bases: *ExternalException*

Base exception for wrapping an explicit *inmanta.plugins.PluginException* raised from a plugin call.

11.5.3 Plugins

class inmanta.plugins.**Context**(*resolver: Resolver*, *queue: QueueScheduler*, *owner: FunctionCall*, *plugin: Plugin*, *result: ResultVariable*)

An instance of this class is used to pass context to the plugin

get_client() → Client

get_compiler() → Compiler

get_data_dir() → str

Get the path to the data dir (and create if it does not exist yet

get_environment_id() → str

get_queue_scheduler() → QueueScheduler

get_resolver() → Resolver

get_sync_client() → SyncClient

get_type(*name: LocatableString*) → *Type*

Get a type from the configuration model.

run_sync(*function: Callable[[], Awaitable[T]]*, *timeout: int = 5*) → T

Execute the async function and return its result. This method uses this thread's current (not running) event loop if there is one, otherwise it creates a new one. The main use for this function is to use the inmanta internal rpc to communicate with the server.

Parameters

- **function** – The async function to execute. This function should return a yieldable object.
- **timeout** – A timeout for the async function.

Returns

The result of the async call.

Raises

ConnectionRefusedError – When the function timeouts this exception is raised.

inmanta.plugins.**plugin**(*function: Callable | None = None*, *commands: list[str] | None = None*, *emits_statements: bool = False*, *allow_unknown: bool = False*) → Callable

Python decorator to register functions with inmanta as plugin

Parameters

- **function** – The function to register with inmanta. This is the first argument when it is used as decorator.
- **commands** – A list of command paths that need to be available. Inmanta raises an exception when the command is not available.
- **emits_statements** – Set to true if this plugin emits new statements that the compiler should execute. This is only required for complex plugins such as integrating a template engine.
- **allow_unknown** – Set to true if this plugin accepts Unknown values as valid input.

class inmanta.plugins.**PluginException**(*message: str*)

Base class for custom exceptions raised from a plugin.

class inmanta.plugins.**PluginMeta**(*name: str*, *bases: tuple[type, ...]*, *dct: dict[str, object]*)

Bases: `type`

A metaclass that keeps track of concrete plugin subclasses. This class is responsible for all plugin registration.

classmethod **add_function**(*plugin_class: type[Plugin]*) → None

Add a function plugin class

classmethod **clear**(*inmanta_module: str | None = None*) → None

Clears registered plugin functions.

Parameters
`inmanta_module` – Clear plugin functions for a specific inmanta module. If omitted, clears all registered plugin functions.

`classmethod get_functions()` → dict[str, Type[Plugin]]

Get all functions that are registered

11.5.4 Resources

`inmanta.resources.resource`(*name: str*, *id_attribute: str*, *agent: str*)

A decorator that registers a new resource. The decorator must be applied to classes that inherit from `Resource`

Parameters

- `name` – The name of the entity in the configuration model it creates a resources from. For example `std::File`
- `id_attribute` – The attribute of *this* resource that uniquely identifies a resource on an agent. This attribute can be mapped.
- `agent` – This string indicates how the agent of this resource is determined. This string points to an attribute, but it can navigate relations (this value cannot be mapped). For example, the agent argument could be `host.name`

`class inmanta.resources.Resource`(*_id:* Id)

Plugins should inherit resource from this class so a resource from a model can be serialized and deserialized.

Such as class is registered when the `resource()` decorator is used. Each class needs to indicate the fields the resource will have with a class field named "fields". A metaclass merges all fields lists from the class itself and all superclasses. If a field it not available directly in the model object the serializer will look for static methods in the class with the name "get_$fieldname".

`clone`(***kwargs: Any*) → T

Create a clone of this resource. The given kwargs can be used to override attributes.

Returns
The cloned resource

`class inmanta.resources.PurgeableResource`(*_id:* Id)

See `std::PurgeableResource` for more information.

`class inmanta.resources.ManagedResource`(*_id:* Id)

See `std::ManagedResource` for more information.

`class inmanta.resources.IgnoreResourceException`

Throw this exception when a resource should not be included by the exported. Typically resources use this to indicate that they are not managed by the orchestrator.

`class inmanta.resources.Id`(*entity_type: str*, *agent_name: str*, *attribute: str*, *attribute_value: str*, *version: int = 0*)

A unique id that identifies a resource that is managed by an agent

`classmethod parse_id`(*resource_id: ResourceVersionIdStr | ResourceIdStr*, *version: int | None = None*) → *Id*

Parse the resource id and return the type, the hostname and the resource identifier.

Parameters
`version` – If provided, the version field of the returned Id will be set to this version.

resource_str() → ResourceIdStr

String representation for this resource id with the following format:
<type>[<agent>,<attribute>=<value>] - type: The resource type, as defined in the configuration model. For example *std::File*. - agent: The agent responsible for this resource. - attribute: The key attribute that uniquely identifies this resource on the agent - value: The corresponding value for this key attribute.

Returns
Returns a *inmanta.data.model.ResourceIdStr*

class inmanta.execute.util.**Unknown**(*source: object*)

An instance of this class is used to indicate that this value can not be determined yet.

Parameters
source – The source object that can determine the value

11.5.5 Handlers

inmanta.agent.handler.**cache**(*func: T_FUNC | None = None*, *ignore: list[str] = []*, *timeout: int = 5000*, *for_version: bool = True*, *cache_none: bool = True*, *cacheNone: bool | None = None*, *call_on_delete: Callable[[Any], None] | None = None*) → T_FUNC | Callable[[T_FUNC], T_FUNC]

decorator for methods in resource handlers to provide caching

this decorator works similar to memoization: when the decorate method is called, its return value is cached, for subsequent calls, the cached value is used instead of the actual value

The name of the method + the arguments of the method form the cache key

If an argument named version is present and for_version is True, the cache entry is flushed after this version has been deployed If an argument named resource is present, it is assumed to be a resource and its ID is used, without the version information

Parameters

- **timeout** – the number of second this cache entry should live
- **for_version** – if true, this value is evicted from the cache when this deploy is ready
- **ignore** – a list of argument names that should not be part of the cache key
- **cache_none** – cache returned none values
- **call_on_delete** – A callback function that is called when the value is removed from the cache, with the value as argument.

inmanta.agent.handler.**provider**(*resource_type: str*, *name: str*) → None

A decorator that registers a new handler.

Parameters

- **resource_type** – The type of the resource this handler is responsible for. For example, *std::File*
- **name** – A name to reference this provider.

class inmanta.agent.handler.**SkipResource**

Bases: Exception

A handler should raise this exception when a resource should be skipped. The resource will be marked as skipped instead of failed.

class inmanta.agent.handler.**ResourcePurged**

If the `read_resource()` method raises this exception, the agent will mark the current state of the resource as purged.

class inmanta.agent.handler.**HandlerContext**(*resource: Resource*, *dry_run: bool = False*, *action_id: UUID | None = None*, *logger: Logger | None = None*)

Context passed to handler methods for state related "things"

add_change(*name: str*, *desired: object*, *current: object = None*) → None

Report a change of a field. This field is added to the set of updated fields

Parameters

- **name** – The name of the field that was updated
- **desired** – The desired value to which the field was updated (or should be updated)
- **current** – The value of the field before it was updated

add_changes(***kwargs: BaseModel | UUID | bool | int | float | datetime | str*) → None

Report a list of changes at once as kwargs

Parameters

- **key** – The name of the field that was updated. This field is also added to the set of updated fields
- **value** – The desired value of the field.

To report the previous value of the field, use the add_change method

fields_updated(*fields: str*) → None

Report that fields have been updated

is_dry_run() → bool

Is this a dryrun?

set_fact(*fact_id: str*, *value: str*, *expires: bool = True*) → None

Send a fact to the Inmanta server.

Parameters

- **fact_id** – The name of the fact.
- **value** – The actual value of the fact.
- **expires** – Whether this fact expires or not.

set_status(*status: ResourceState*) → None

Set the status of the handler operation.

update_changes(*changes: dict[str, AttributeStateChange]*) → None

update_changes(*changes: dict[str, dict[str, BaseModel | UUID | bool | int | float | datetime | str | None]]*) → None

update_changes(*changes: dict[str, tuple[BaseModel | UUID | bool | int | float | datetime | str, BaseModel | UUID | bool | int | float | datetime | str]]*) → None

Update the changes list with changes

Parameters

changes – This should be a dict with a value a dict containing "current" and "desired" keys

class inmanta.agent.handler.**ResourceHandler**(*agent: inmanta.agent.executor.AgentInstance*, *io: IOBase | None = None*)

A class that handles resources.

`_abc_impl = <_abc._abc_data object>`

`_diff`(*current: TResource*, *desired: TResource*) → dict[str, dict[str, Any]]

Calculate the diff between the current and desired resource state.

Parameters

- **`current`** – The current state of the resource
- **`desired`** – The desired state of the resource

Returns

A dict with key the name of the field and value another dict with "current" and "desired" as keys for fields that require changes.

`check_facts`(*ctx:* HandlerContext, *resource: TResource*) → dict[str, object]

This method is called by the agent to query for facts. It runs `pre()` and `post()`. This method calls `facts()` to do the actual querying.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to query facts for.

Returns

A dict with fact names as keys and facts values.

`check_resource`(*ctx:* HandlerContext, *resource: TResource*) → TResource

Check the current state of a resource

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to check the current state of.

Returns

A resource to represents the current state. Use the *`clone()`* to create clone of the given resource that can be modified.

`do_changes`(*ctx:* HandlerContext, *resource: TResource*, *changes: Mapping[str, Mapping[str, object]]*) → None

Do the changes required to bring the resource on this system in the state of the given resource.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to check the current state of.
- **`changes`** – The changes that need to occur as reported by *`list_changes()`*

`execute`(*ctx:* HandlerContext, *resource: TResource*, *dry_run: bool = False*) → None

Enforce a resource's intent and inform the handler context of any relevant changes (e.g. set deployed status, report attribute changes). Called only when all of its dependencies have successfully deployed.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to deploy.
- **`dry_run`** – If set to true, the intent is not enforced, only the set of changes it would bring is computed.

`list_changes`(*ctx: HandlerContext*, *resource: TResource*) → dict[str, dict[str, Any]]

Returns the changes required to bring the resource on this system in the state described in the resource entry. This method calls *`check_resource()`*

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to check the current state of.

Returns

A dict with key the name of the field and value another dict with "current" and "desired" as keys for fields that require changes.

`class inmanta.agent.handler.CRUDHandler`(*agent: inmanta.agent.executor.AgentInstance*, *io: IOBase | None = None*)

This handler base class requires CRUD methods to be implemented: create, read, update and delete. Such a handler only works on purgeable resources.

`available`(*resource: TResource*) → bool

Kept for backwards compatibility, new handler implementations should never override this.

Parameters

`resource` – Resource for which to check whether this handler is available.

`calculate_diff`(*ctx: HandlerContext*, *current: TPurgeableResource*, *desired: TPurgeableResource*) → dict[str, dict[str, Any]]

Calculate the diff between the current and desired resource state.

Parameters

- **`ctx`** – Context can be used to get values discovered in the read method. For example, the id used in API calls. This context should also be used to let the handler know what changes were made to the resource.
- **`current`** – The current state of the resource
- **`desired`** – The desired state of the resource

Returns

A dict with key the name of the field and value another dict with "current" and "desired" as keys for fields that require changes.

`can_reload`() → bool

Can this handler reload?

Returns

Return true if this handler needs to reload on requires changes.

`check_facts`(*ctx: HandlerContext*, *resource: TResource*) → dict[str, object]

This method is called by the agent to query for facts. It runs `pre()` and `post()`. This method calls `facts()` to do the actual querying.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to query facts for.

Returns

A dict with fact names as keys and facts values.

`check_resource`(*ctx: HandlerContext*, *resource: TResource*) → TResource

Check the current state of a resource

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.

- **`resource`** – The resource to check the current state of.

Returns
A resource to represents the current state. Use the *`clone()`* to create clone of the given resource that can be modified.

`close``()` → None

Override this method to implement custom logic called by the agent on handler deactivation. i.e. when the instantiated handler will no longer be used by the agent.

`create_resource``(`*ctx:* HandlerContext, *resource: TPurgeableResource*`)` → None

This method is called by the handler when the resource should be created.

Parameters

- **`context`** – Context can be used to get values discovered in the read method. For example, the id used in API calls. This context should also be used to let the handler know what changes were made to the resource.
- **`resource`** – The desired resource state.

`delete_resource``(`*ctx:* HandlerContext, *resource: TPurgeableResource*`)` → None

This method is called by the handler when the resource should be deleted.

Parameters

- **`ctx`** – Context can be used to get values discovered in the read method. For example, the id used in API calls. This context should also be used to let the handler know what changes were made to the resource.
- **`resource`** – The desired resource state.

`deploy``(`*ctx:* HandlerContext, *resource: TResource*, *requires: Mapping[ResourceIdStr, ResourceState]*`)` → None

Main entrypoint of the handler that will be called by the agent to deploy a resource on the server. The agent calls this method for a given resource as soon as all its dependencies (*requires* relation) are ready. It is always called, even when one of the dependencies failed to deploy.

Takes appropriate action based on the state of its dependencies. Calls *execute* iff the handler should actually execute, i.e. enforce the intent represented by the resource. A handler may choose not to proceed to this execution stage, e.g. when one of the resource's dependencies failed.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to deploy
- **`requires`** – A dictionary mapping the resource id of each dependency of the given resource to its resource state.

`do_changes``(`*ctx:* HandlerContext, *resource: TResource*, *changes: Mapping[str, Mapping[str, object]]*`)` → None

Do the changes required to bring the resource on this system in the state of the given resource.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource to check the current state of.
- **`changes`** – The changes that need to occur as reported by *`list_changes()`*

`do_reload``(`*ctx:* HandlerContext, *resource: TResource*`)` → None

Perform a reload of this resource.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.

- **resource** – The resource to reload.

execute(*ctx: HandlerContext*, *resource: TPurgeableResource*, *dry_run: bool = False*) → None

Enforce a resource's intent and inform the handler context of any relevant changes (e.g. set deployed status, report attribute changes). Called only when all of its dependencies have successfully deployed.

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to deploy.
- **dry_run** – If set to true, the intent is not enforced, only the set of changes it would bring is computed.

facts(*ctx: HandlerContext*, *resource: TResource*) → dict[str, object]

Override this method to implement fact querying. A queried fact can be reported back in two different ways: either via the return value of this method or by adding the fact to the HandlerContext via the `set_fact()` method. `pre()` and `post()` are called before and after this method.

Parameters

- **ctx** – Context object to report changes, logs and facts to the agent and server.
- **resource** – The resource to query facts for.

Returns

A dict with fact names as keys and facts values.

get_client() → SessionClient

Get the client instance that identifies itself with the agent session.

Returns

A client that is associated with the session of the agent that executes this handler.

get_file(*hash_id: str*) → bytes | None

Retrieve a file from the fileserver identified with the given id.

Parameters

hash_id – The id of the content/file to retrieve from the server.

Returns

The content in the form of a bytestring or none is the content does not exist.

list_changes(*ctx: HandlerContext*, *resource: TResource*) → dict[str, dict[str, Any]]

Returns the changes required to bring the resource on this system in the state described in the resource entry. This method calls `check_resource()`

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to check the current state of.

Returns

A dict with key the name of the field and value another dict with "current" and "desired" as keys for fields that require changes.

post(*ctx: HandlerContext*, *resource: TResource*) → None

Method executed after a handler operation. Override this method to run after an operation.

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource being handled.

`pre`(*ctx:* HandlerContext, *resource: TResource*) → None

Method executed before a handler operation (Facts, dryrun, real deployment, . . .) is executed. Override this method to run before an operation.

Parameters

- **`ctx`** – Context object to report changes and logs to the agent and server.
- **`resource`** – The resource being handled.

`read_resource`(*ctx:* HandlerContext, *resource: TPurgeableResource*) → None

This method reads the current state of the resource. It provides a copy of the resource that should be deployed, the method implementation should modify the attributes of this resource to the current state.

Parameters

- **`ctx`** – Context can be used to pass value discovered in the read method to the CUD methods. For example, the id used in API calls
- **`resource`** – A clone of the desired resource state. The read method need to set values on this object.

Raises

- ***`SkipResource`*** – Raise this exception when the handler should skip this resource
- ***`ResourcePurged`*** – Raise this exception when the resource does not exist yet.

`run_sync`(*func: Callable[[], Awaitable[T]]*) → T

Run the given async function on the ioloop of the agent. It will block the current thread until the future resolves.

Parameters
`func` – A function that returns a yieldable future.

Returns
The result of the async function.

`set_cache`(*cache: AgentCache*) → None

The agent calls this method when it has deemed this handler suitable for a given resource. This cache will be used for methods decorated with @cache.

Parameters
`cache` – The AgentCache to use.

`stat_file`(*hash_id: str*) → bool

Check if a file exists on the server.

Parameters
`hash_id` – The id of the file on the server. The convention is the use the sha1sum of the content as id.

Returns
True if the file is available on the server.

`update_resource`(*ctx:* HandlerContext, *changes: dict[str, dict[str, Any]]*, *resource: TPurgeableResource*) → None

This method is called by the handler when the resource should be updated.

Parameters

- **`ctx`** – Context can be used to get values discovered in the read method. For example, the id used in API calls. This context should also be used to let the handler know what changes were made to the resource.
- **`changes`** – A map of resource attributes that should be changed. Each value is a tuple with the current and the desired value.

- **resource** – The desired resource state.

upload_file(*hash_id: str*, *content: bytes*) → None

Upload a file to the server

Parameters

- **hash_id** – The id to identify the content. The convention is to use the sha1sum of the content to identify it.
- **content** – A byte string with the content

class inmanta.agent.io.local.**LocalIO**(*uri: str*, *config: Dict[str, str | None]*)

This class provides handler IO methods

This class is part of the stable API.

chmod(*path: str*, *permissions: str*) → None

Change the permissions

Parameters

- **path** (*str*) – The path of the file or directory to change the permission of.
- **permissions** (*str*) – An octal string with the permission to set.

chown(*path: str*, *user: str | None = None*, *group: str | None = None*) → None

Change the ownership of a file.

Parameters

- **path** (*str*) – The path of the file or directory to change the ownership of.
- **user** (*str*) – The user to change to
- **group** (*str*) – The group to change to

close() → None

Close any resources

file_exists(*path: str*) → bool

Check if a given file exists

Parameters
path (*str*) – The path to check if it exists.

Returns
Returns true if the file exists

Return type
bool

file_stat(*path: str*) → *Dict*[str, int | str]

Do a stat call on a file

Parameters
path (*str*) – The file or direct to stat

Returns
A dict with the owner, group and permissions of the given path

Return type
dict[str, str]

hash_file(*path: str*) → str

Return the sha1sum of the file at path

Parameters
path (*str*) – The path of the file to hash the content of

Returns
The sha1sum in a hex string

Return type
str

`is_remote()` → bool
Are operation executed remote

Returns
Returns true if the io operations are remote.

Return type
bool

`is_symlink`(*path: str*) → bool
Is the given path a symlink

Parameters
`path` (*str*) – The path of the symlink

Returns
Returns true if the given path points to a symlink

Return type
str

`mkdir`(*path: str*) → None
Create a directory

Parameters
`path` (*str*) – Create this directory. The parent needs to exist.

`put`(*path: str*, *content: str*) → None
Put the given content at the given path

Parameters

- `path` (*str*) – The location where to write the file
- `content` (*bytes*) – The binarystring content to write to the file.

`read`(*path: str*) → str
Read in the file in path and return its content as string

Parameters
`path` (*str*) – The path of the file to read.

Returns
The string content of the file

Return type
string

`read_binary`(*path: str*) → bytes
Read in the file in path and return its content as a bytestring

Parameters
`path` (*str*) – The path of the file to read.

Returns
The byte content of the file

Return type
bytes

readlink(*path: str*) → str

Return the target of the path

Parameters
`path` (*str*) – The symlink to get the target for.

Returns
The target of the symlink

Return type
str

remove(*path: str*) → None

Remove a file

Parameters
`path` (*str*) – The path of the file to remove.

rmdir(*path: str*) → None

Remove a directory

Parameters
`path` (*str*) – The directory to remove

run(*command: str*, *arguments: List[str] = []*, *env: Dict[str, str] | None = None*, *cwd: str | None = None*, *timeout: int | None = None*) → Tuple[str, str, int]

Execute a command with the given argument and return the result

Parameters

- `command` (*str*) – The command to execute.
- `arguments` (*list*) – The arguments of the command
- `env` (*dict*) – A dictionary with environment variables.
- `cwd` (*str*) – The working dir to execute the command in.
- `timeout` (*int*) – The timeout for this command. This parameter is ignored if the command is executed remotely with a python 2 interpreter.

Returns
A tuple with (stdout, stderr, returncode)

Return type
tuple

symlink(*source: str*, *target: str*) → None

Symlink source to target

Parameters

- `source` (*str*) – Create a symlink of this path to target
- `target` (*str*) – The path of the symlink to create

11.5.6 Export

`@inmanta.export.dependency_manager`(*function: Callable[[dict[str, Entity], dict[*Id*, *Resource*]], None]*) → None

Register a function that manages dependencies in the configuration model that will be deployed.

11.5.7 Attributes

`class inmanta.ast.attribute.Attribute`(*entity:* Entity, *value_type:* Type, *name: str*, *location: Location*, *multi: bool = False*, *nullable: bool = False*)

The attribute base class for entity attributes.

Parameters
`entity` – The entity this attribute belongs to

`get_type()` → *Type*

Get the type of this attribute.

`property type: Type`

Get the type of this attribute.

`validate`(*value: object*) → None

Validate a value that is going to be assigned to this attribute. Raises a *`inmanta.ast.RuntimeException`* if validation fails.

`class inmanta.ast.attribute.RelationAttribute`(*entity:* Entity, *value_type:* Type, *name: str*, *location: Location*)

Bases: *`Attribute`*

An attribute that is a relation

11.5.8 Modules

`class inmanta.module.InstallMode`(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

The module install mode determines what version of a module should be selected when a module is downloaded.

`master = 'master'`

For V1 modules: Use the module's master branch. For V2 modules: Equivalent to *`InstallMode.prerelease`*

`prerelease = 'prerelease'`

Similar to *`InstallMode.release`* but prerelease versions are allowed as well.

`release = 'release'`

Only use a released version that is compatible with the current compiler and any version constraints defined in the `requires` lists for the project or any other modules (see *`ProjectMetadata`*, `ModuleV1Metadata` and `ModuleV2Metadata`).

A module is considered released in the following situations:

- **For V1 modules: There is a tag on a commit. This tag is a valid, pep440 compliant version identifier and it's not a**
  prelease version.
- **For V2 modules: The python package was published on a Python package repository, the version identifier is pep440**
  compliant and is not a prerelease version.

`inmanta.module.INSTALL_OPTS: list[str] = ['release', 'prerelease', 'master']`

List of possible module install modes, kept for backwards compatibility. New code should use *InstallMode* instead.

class `inmanta.module.InvalidModuleException`(*msg: str*)

This exception is raised if a module is invalid.

class `inmanta.module.InvalidMetadata`(*msg: str*, *validation_error: ValidationError* | *None* = *None*)

This exception is raised if the metadata file of a project or module is invalid.

class `inmanta.module.ModuleLike`(*path: str*)

Bases: `ABC`, `Generic[TMetadata]`

Commons superclass for projects and modules, which are both versioned by git

Variables

name – The name for this module like instance, in the context of the Inmanta DSL.

abstract classmethod from_path(*path: str*) → *ModuleLike* | None

Get a concrete module like instance from a path. Returns None when no project or module is present at the given path.

property metadata: TMetadata

class `inmanta.module.Module`(*project:* Project | *None*, *path: str*)

Bases: *ModuleLike*[`TModuleMetadata`], `ABC`

This class models an inmanta configuration module

abstract classmethod from_path(*path: str*) → *Module* | None

Get a concrete module like instance from a path. Returns None when no project or module is present at the given path.

get_plugin_files() → Iterator[tuple[Path, ModuleName]]

Returns a tuple (absolute_path, fq_mod_name) of all python files in this module.

unload() → None

Unloads this module instance from the project, the registered plugins and the loaded Python modules.

`inmanta.module.ModuleName = inmanta.module.ModuleName`

NewType creates simple unique types with almost zero runtime overhead.

NewType(name, tp) is considered a subtype of tp by static type checkers. At runtime, NewType(name, tp) returns a dummy callable that simply returns its argument.

Usage:

```
UserId = NewType('UserId', int)

def name_by_id(user_id: UserId) -> str:
    ...

UserId('user')          # Fails type check

name_by_id(42)          # Fails type check
name_by_id(UserId(42))  # OK

num = UserId(5) + 1     # type: int
```

class `inmanta.module.ModuleV1`(*project:* Project | *None*, *path: str*)

Bases: *Module*[`ModuleV1Metadata`], `ModuleLikeWithYmlMetadataFile`

classmethod from_path(*path: str*) → TModule | None

Get a concrete module like instance from a path. Returns None when no project or module is present at the given path.

class inmanta.module.**ModuleV2**(*project: Project | None, path: str, is_editable_install: bool = False, installed_version: Version | None = None*)

Bases: *Module*[ModuleV2Metadata]

classmethod from_path(*path: str*) → TModule | None

Get a concrete module like instance from a path. Returns None when no project or module is present at the given path.

is_editable() → bool

Returns True iff this module has been installed in editable mode.

class inmanta.module.**ModuleSource**

Bases: Generic[TModule]

get_installed_module(*project: Project | None, module_name: str*) → TModule | None

Returns a module object for a module if it is installed.

Parameters

- **project** – The project associated with the module.
- **module_name** – The name of the module.

class inmanta.module.**ModuleV2Source**(*urls: Sequence[str] = []*)

Bases: *ModuleSource*[*ModuleV2*]

inmanta.module.**Path** = **inmanta.module.Path**

NewType creates simple unique types with almost zero runtime overhead.

NewType(name, tp) is considered a subtype of tp by static type checkers. At runtime, NewType(name, tp) returns a dummy callable that simply returns its argument.

Usage:

```
UserId = NewType('UserId', int)

def name_by_id(user_id: UserId) -> str:
    ...

UserId('user')          # Fails type check

name_by_id(42)          # Fails type check
name_by_id(UserId(42))  # OK

num = UserId(5) + 1     # type: int
```

class inmanta.loader.**PluginModuleFinder**(*modulepaths: list[str]*)

Bases: MetaPathFinder

Custom module finder which handles V1 Inmanta modules. V2 modules are handled using the standard Python finder. This finder is stored as the last entry in *meta_path*, as such that the default Python Finders detect V2 modules first.

classmethod reset() → None

Remove the PluginModuleFinder from sys.meta_path.

inmanta.loader.**unload_inmanta_plugins**(*inmanta_module: str | None = None*) → None

Unloads Python modules associated with inmanta modules (*inmanta_plugins* submodules).

Parameters
`inmanta_module` – Unload the Python modules for a specific inmanta module. If omitted, unloads the Python modules for all inmanta modules.

11.5.9 Project

`class inmanta.module.Project`(*path: str*, *autostd: bool = True*, *main_file: str = 'main.cf'*, *venv_path: str | VirtualEnv | None = None*, *attach_cf_cache: bool = True*, *strict_deps_check: bool | None = None*)

Bases: `ModuleLike[ProjectMetadata]`, `ModuleLikeWithYmlMetadataFile`

An inmanta project

Variables

- `modules` – The collection of loaded modules for this project.
- `module_source` – The v2 module source for this project.

`classmethod get`(*main_file: str = 'main.cf'*, *strict_deps_check: bool | None = None*) → *Project*

Get the instance of the project

`install_modules`(*, *bypass_module_cache: bool = False*, *update_dependencies: bool = False*) → None

Installs all modules, both v1 and v2.

Parameters

- `bypass_module_cache` – Fetch the module data from disk even if a cache entry exists.
- `update_dependencies` – Update all Python dependencies (recursive) to their latest versions.

`load`(*install: bool = False*) → None

Load this project's AST and plugins.

Parameters
`install` – Whether to install the project's modules before attempting to load it.

`classmethod set`(*project: Project*, *, *clean: bool = True*) → None

Set the instance of the project.

Parameters
`clean` – Clean up all side effects of any previously loaded projects. Clears the registered plugins and loaded Python plugins packages.

`class inmanta.module.ProjectNotFoundException`(*msg: str*)

Bases: `CompilerException`

This exception is raised when inmanta is unable to find a valid project

11.5.10 Python Environment

`inmanta.env.mock_process_env`(*, *python_path: str | None = None*, *env_path: str | None = None*) → None

Overrides the process environment information. This forcefully sets the environment that is recognized as the outer Python environment. This function should only be called when a Python environment has been set up dynamically and this environment should be treated as if this process was spawned from it, and even then with great care.

Parameters

- `python_path` – The path to the python binary. Only one of *python_path* and *env_path* should be set.

- **env_path** – The path to the python environment directory. Only one of *python_path* and *env_path* should be set.

class inmanta.env.**VirtualEnv**(*env_path: str*)

Creates and uses a virtual environment for this process. This virtualenv inherits from the previously active one.

init_env() → None

Initialize the virtual environment.

use_virtual_env() → None

Activate the virtual environment.

11.5.11 Variables

class inmanta.ast.variables.**Reference**(*name: LocatableString*)

This class represents a reference to a value

Variables

name – The name of the Reference as a string.

name

11.5.12 Typing

The *inmanta.ast.type* module contains a representation of inmanta types, as well as validation logic for those types.

class inmanta.ast.type.**Type**

This class is the abstract base class for all types in the Inmanta *DSL* that represent basic data. These are types that are not relations. Instances of subclasses represent a type in the Inmanta language.

get_base_type() → *Type*

Returns the base type for this type, i.e. the plain type without modifiers such as expressed by *[]* and *?* in the *DSL*.

is_primitive() → bool

Returns true iff this type is a primitive type, i.e. number, string, bool.

type_string() → str | None

Returns the type string as expressed in the Inmanta *DSL*, if this type can be expressed in the *DSL*. Otherwise returns None.

validate(*value: object | None*) → bool

Validate the given value to check if it satisfies the constraints associated with this type. Returns true iff validation succeeds, otherwise raises a `inmanta.ast.RuntimeException`.

with_base_type(*base_type:* Type) → *Type*

Returns the type formed by replacing this type's base type with the supplied type.

class inmanta.ast.type.**NullableType**(*element_type:* Type)

Bases: `Type`

Represents a nullable type in the Inmanta *DSL*. For example *NullableType(Number())* represents *number?*.

class inmanta.ast.type.**Primitive**

Bases: `Type`

Abstract base class representing primitive types.

cast(*value: object | None*) → object

Cast a value to this type. If the value can not be cast, raises a *inmanta.ast.RuntimeException*.

`class inmanta.ast.type.`**Number**

Bases: *Primitive*

This class represents an integer or a float in the configuration model.

`class inmanta.ast.type.`**Integer**

Bases: *Number*

An instance of this class represents the int type in the configuration model.

`class inmanta.ast.type.`**Bool**

Bases: *Primitive*

This class represents a simple boolean that can hold true or false.

`class inmanta.ast.type.`**String**

Bases: *Primitive*

This class represents a string type in the configuration model.

`class inmanta.ast.type.`**Union**(*types: List[*Type*]*)

Bases: *Type*

Instances of this class represent a union of multiple types.

`class inmanta.ast.type.`**Literal**

Bases: *Union*

Instances of this class represent a literal in the configuration model. A literal is a primitive or a list or dict where all values are literals themselves.

`class inmanta.ast.type.`**List**

Bases: *Type*

Instances of this class represent a list type containing any types of values. This class refers to the list type used in plugin annotations. For the list type in the Inmanta DSL, see *LiteralList*.

`class inmanta.ast.type.`**TypedList**(*element_type:* Type)

Bases: *List*

Instances of this class represent a list type containing any values of type element_type. For example *TypedList(Number())* represents *number[]*.

`class inmanta.ast.type.`**LiteralList**

Bases: *TypedList*

Instances of this class represent a list type containing only *Literal* values. This is the *list* type in the *DSL*

`class inmanta.ast.type.`**Dict**

Bases: *Type*

Instances of this class represent a dict type with any types of values.

`class inmanta.ast.type.`**TypedDict**(*element_type:* Type)

Bases: *Dict*

Instances of this class represent a dict type containing only values of type element_type.

`class inmanta.ast.type.`**LiteralDict**

Bases: *TypedDict*

Instances of this class represent a dict type containing only *Literal* values. This is the *dict* type in the *DSL*

class inmanta.ast.type.**ConstraintType**(*namespace: Namespace*, *name: str*)

Bases: `NamedType`

A type that is based on a primitive type but defines additional constraints on this type. These constraints only apply on the value of the type.

inmanta.ast.type.**TYPES**

Maps Inmanta *DSL* types to their internal representation. For each key, value pair, *value.type_string()* is guaranteed to return key.

Note: The type classes themselves do not represent inmanta types, their instances do. For example, the type representation for the inmanta type *number* is *Number()*, not *Number*.

11.5.13 Protocol

class inmanta.protocol.common.**Result**(*code: int = 0*, *result: dict[str, Any] | None = None*)

A result of a method call

code

The result code of the method call.

property result: dict[str, Any] | None

11.5.14 Data

Warning: In contrast to the rest of this section, the data API interface is subject to change. It is documented here because it is currently the only available API to interact with the data framework. A restructure of the data framework is expected at some point. Until then, this API should be considered unstable.

inmanta.data.**TBaseDocument** : **typing.TypeVar**

TypeVar with BaseDocument bound.

class inmanta.data.**BaseDocument**(*from_postgres: bool = False*, ***kwargs: object*)

A base document in the database. Subclasses of this document determine collections names. This type is mainly used to bundle query methods and generate validate and query methods for optimized DB access. This is not a full ODM.

Fields are modelled using type annotations similar to protocol and pydantic. The following is supported:

- Attributes are defined at class level with type annotations
- Attributes do not need a default value. When no default is provided, they are marked as required.
- When a value does not have to be set: either a default value or making it optional can be used. When a field is optional without a default value, none will be set as default value so that the field is available.
- Fields that should be ignored, can be added to __ignore_fields__ This attribute is a tuple of strings
- Fields that are part of the primary key should be added to the __primary_key__ attributes. This attribute is a tuple of strings.

async classmethod get_by_id(*doc_id: UUID*, *connection: Connection | None = None*) → TBaseDocument | None

Get a specific document based on its ID

Returns

An instance of this class with its fields filled from the database.

async classmethod get_list(*, *order_by_column: str | None = None*, *order: str | None = None*, *limit: int | None = None*, *offset: int | None = None*, *no_obj: bool | None = None*, *lock: RowLockMode | None = None*, *connection: Connection | None = None*, ***query: object*) → list[TBaseDocument]

Get a list of documents matching the filter args

class inmanta.data.Compile(*from_postgres: bool = False*, ***kwargs: object*)

Bases: `BaseDocument`

A run of the compiler

Parameters

- `environment` – The environment this resource is defined in
- `requested` – Time the compile was requested
- `started` – Time the compile started
- `completed` – Time to compile was completed
- `do_export` – should this compiler perform an export
- `force_update` – should this compile definitely update
- `metadata` – exporter metadata to be passed to the compiler
- `requested_environment_variables` – environment variables requested to be passed to the compiler
- `mergeable_environment_variables` – environment variables to be passed to the compiler. These env vars can be compacted over multiple compiles. If multiple values are compacted, they will be joined using spaces.
- `used_environment_variables` – environment variables passed to the compiler, None before the compile is started
- `success` – was the compile successful
- `handled` – were all registered handlers executed?
- `version` – version exported by this compile
- `remote_id` – id as given by the requestor, used by the requestor to distinguish between different requests
- `compile_data` – json data as exported by compiling with the –export-compile-data parameter
- `substitute_compile_id` – id of this compile's substitute compile, i.e. the compile request that is similar to this one that actually got compiled.
- `partial` – True if the compile only contains the entities/resources for the resource sets that should be updated
- `removed_resource_sets` – indicates the resource sets that should be removed from the model
- `exporter_plugin` – Specific exporter plugin to use
- `notify_failed_compile` – if true use the notification service to notify that a compile has failed. By default, notifications are enabled only for exporting compiles.
- `failed_compile_message` – Optional message to use when a notification for a failed compile is created
- `soft_delete` – Prevents deletion of resources in removed_resource_sets if they are being exported.

`async classmethod get_substitute_by_id`(*compile_id: UUID*, *connection: Connection | None = None*) → *Compile* | None

Get a compile's substitute compile if it exists, otherwise get the compile by id.

Parameters
`compile_id` – The id of the compile for which to get the substitute compile.

Returns
The compile object for compile c2 that is the substitute of compile c1 with the given id. If c1 does not have a substitute, returns c1 itself.

`to_dto`() → CompileRun

`class inmanta.data.ConfigurationModel`(***kwargs: object*)

Bases: `BaseDocument`

A specific version of the configuration model.

Parameters

- `version` – The version of the configuration model, represented by a unix timestamp.
- `environment` – The environment this configuration model is defined in
- `date` – The date this configuration model was created
- `partial_base` – If this version was calculated from a partial export, the version the partial was applied on.
- `released` – Is this model released and available for deployment?
- `deployed` – Is this model deployed?
- `result` – The result of the deployment. Success or error.
- `version_info` – Version metadata
- `total` – The total number of resources
- `is_suitable_for_partial_compiles` – This boolean indicates whether the model can later on be updated using a partial compile. In other words, the value is True iff no cross resource set dependencies exist between the resources.

`async classmethod get_versions`(*environment: UUID*, *start: int = 0*, *limit: int = 100000*, *connection: Connection | None = None*) → list[*ConfigurationModel*]

Get all versions for an environment ordered descending

`class inmanta.data.Environment`(*from_postgres: bool = False*, ***kwargs: object*)

Bases: `BaseDocument`

A deployment environment of a project

Parameters

- `id` – A unique, machine generated id
- `name` – The name of the deployment environment.
- `project` – The project this environment belongs to.
- `repo_url` – The repository url that contains the configuration model code for this environment.
- `repo_branch` – The repository branch that contains the configuration model code for this environment.

- **settings** – Key/value settings for this environment. This dictionary does not necessarily contain a key for every environment setting known by the server. This is done for backwards compatibility reasons. When a setting was renamed, we need to determine whether the old or the new setting has to be taken into account. The logic to decide that is the following:
  - When the name of the new setting is present in this settings dictionary or when the name of the old setting is not present in the settings dictionary, use the new setting.
  - Otherwise, use the setting with the old name.
- **last_version** – The last version number that was reserved for this environment
- **description** – The description of the environment
- **icon** – An icon for the environment

`class inmanta.data.`**`Report`**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: `BaseDocument`

A report of a substep of compilation

Parameters

- **started** – when the substep started
- **completed** – when it ended
- **command** – the command that was executed
- **name** – The name of this step
- **errstream** – what was reported on system err
- **outstream** – what was reported on system out

`class inmanta.data.`**`Resource`**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: `BaseDocument`

A specific version of a resource. This entity contains the desired state of a resource.

Parameters

- **environment** – The environment this resource version is defined in
- **rid** – The id of the resource and its version
- **resource** – The resource for which this defines the state
- **model** – The configuration model (versioned) this resource state is associated with
- **attributes** – The state of this version of the resource
- **attribute_hash** – hash of the attributes, excluding requires, provides and version, used to determine if a resource describes the same state across versions
- **resource_id_value** – The attribute value from the resource id

`async classmethod` **`get_resources_for_version`**(*environment: UUID*, *version: int*, *agent: str | None = None*, *no_obj: bool = False*, ***, *connection: Connection | None = None*) → list[*Resource*]

`class inmanta.data.`**`ResourceAction`**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: `BaseDocument`

Log related to actions performed on a specific resource version by Inmanta.

Parameters

- **environment** – The environment this action belongs to.

- **version** – The version of the configuration model this action belongs to.
- **resource_version_ids** – The resource version ids of the resources this action relates to.
- **action_id** – This id distinguishes the actions from each other. Action ids have to be unique per environment.
- **action** – The action performed on the resource
- **started** – When did the action start
- **finished** – When did the action finish
- **messages** – The log messages associated with this action
- **status** – The status of the resource when this action was finished
- **changes** – A dict with key the resource id and value a dict of fields -> value. Value is a dict that can contain old and current keys and the associated values. An empty dict indicates that the field was changed but not data was provided by the agent.
- **change** – The change result of an action

async classmethod get_logs_for_version(*environment: UUID*, *version: int*, *action: str | None = None*, *limit: int = 0*, *connection: Connection | None = None*) → list[*ResourceAction*]

class inmanta.data.model.**BaseModel**

Bases: `DateTimeNormalizerModel`

Base class for all data objects in Inmanta

inmanta.data.model.**ResourceIdStr = inmanta.data.model.ResourceIdStr**

The resource id without the version

inmanta.data.model.**ResourceVersionIdStr = inmanta.data.model.ResourceVersionIdStr**

The resource id with the version included.

class inmanta.db.util.**PGRestore**(*script: list[str]*, *postgresql_client: Connection*)

Bases: `object`

Class that offers support to restore a database dump.

This class assumes that the names of schemas, tables and columns in the dump don't contain a dot, double quote or whitespace character.

async inmanta.db.util.**clear_database**(*postgresql_client: Connection*) → None

Remove all content from the database. Removes functions, tables and data types.

11.5.15 Domain conversion

This section describes methods for converting values between the plugin domain and the internal domain. This conversion is performed automatically for plugin arguments and return values so it is only required when bypassing the usual plugin workflow by calling internal methods directly.

class inmanta.execute.proxy.**DynamicProxy**

This class wraps an object and makes sure that a model is never modified by native code.

classmethod return_value(*value: object*) → None | str | tuple[object, ...] | int | float | bool | *DynamicProxy*

Converts a value from the internal domain to the plugin domain.

classmethod unwrap(*item: object*) → object

Converts a value from the plugin domain to the internal domain.

11.5.16 Rest API

The rest API is also available as a swagger spec

The (v2) API endpoints that offer paging, sorting and filtering follow a convention. They share the following parameters:

limit
: specifies the page size, so the maximum number of items returned from the query

start and first_id
: These parameters define the lower limit for the page,

end and last_id
: These parameters define the upper limit for the page (only one of the (*start*, *first_id*), (*end*, *last_id*) pairs should be specified at the same time).

Note: The return value of these methods contain a *links* tag, with the urls of the *next* and *prev* pages, so for simply going through the pages a client only needs to follow these links.

---

filter
: The *filter* parameter is used for filtering the result set.

  Filters should be specified with the syntax *?filter.<filter_key>=value*.

  It's also possible to provide multiple values for the same filter, in this case results are returned, if they match any of these filter values: *?filter.<filter_key>=value&filter.<filter_key>=value2*

  Multiple different filters narrow the results however (they are treated as an 'AND' operator). For example *?filter.<filter_key>=value&filter.<filter_key2>=value2* returns results that match both filters.

  The documentation of each method describes the supported filters.

sort
: The sort parameter describes how the result set should be sorted.

  It should follow the pattern *?<attribute_to_sort_by>.<order>*, for example *?value.desc* (case insensitive).

  The documentation of each method describes the supported attributes to sort by.

Module defining the v1 rest api

`inmanta.protocol.methods.`**`clear_environment`**`(`*id: UUID*`)`

Clears an environment by removing most of its associated data. This method deletes various components associated with the specified environment from the database, including agents, compile data, parameters, notifications, code, resources, and configuration models. However, it retains the entry in the Environment table itself and settings are kept. The environment will be temporarily halted during the decommissioning process.

Parameters
: **`id`** – The id of the environment to be cleared.

Raises

- *`NotFound`* – The given environment doesn't exist.
- *`Forbidden`* – The given environment is protected.

`inmanta.protocol.methods.`**`create_environment`**`(`*project_id: UUID*, *name: str*, *repository: str | None = None*, *branch: str | None = None*, *environment_id: UUID | None = None*`)`

Create a new environment

Parameters

- **`project_id`** – The id of the project this environment belongs to

- **`name`** – The name of the environment.
- **`repository`** – Optional. The URL of the repository.
- **`branch`** – Optional. The name of the branch in the repository.
- **`environment_id`** – Optional. A unique environment id, if none an id is allocated by the server.

`inmanta.protocol.methods.`**`create_project`**(*name: str*, *project_id: UUID | None = None*)

Create a new project

Parameters

- **`name`** – The name of the project
- **`project_id`** – Optional. A unique uuid, when it is not provided the server generates one

`inmanta.protocol.methods.`**`create_token`**(*tid: UUID*, *client_types: list*, *idempotent: bool = True*)

Create or get a new token for the given client types.

Parameters

- **`tid`** – The environment id.
- **`client_types`** – The client types for which this token is valid (api, agent, compiler).
- **`idempotent`** – Optional. The token should be idempotent, meaning it does not have an expire or issued at set, so its value will not change.

`inmanta.protocol.methods.`**`delete_environment`**(*id: UUID*)

Delete the given environment and all related data.

Parameters
`id` – The id of the environment to be deleted.

Raises

- *`NotFound`* – The given environment doesn't exist.
- *`Forbidden`* – The given environment is protected.

`inmanta.protocol.methods.`**`delete_param`**(*tid: UUID*, *id: str*, *resource_id: str | None = None*)

Delete a parameter on the server

Parameters

- **`tid`** – The id of the environment
- **`id`** – The name of the parameter
- **`resource_id`** – Optional. The resource id of the parameter

`inmanta.protocol.methods.`**`delete_project`**(*id: UUID*)

Delete the given project and all related data.

Parameters
`id` – The id of the project to be deleted.

`inmanta.protocol.methods.`**`delete_setting`**(*tid: UUID*, *id: str*)

Restore the given setting to its default value.

Parameters

- **`tid`** – The id of the environment from which the setting is to be deleted.
- **`id`** – The key of the setting to delete.

`inmanta.protocol.methods.`**`delete_version`**(*tid: UUID*, *id: int*)

Delete a particular version and resources

Parameters

- **`tid`** – The id of the environment
- **`id`** – The id of the version to retrieve

`inmanta.protocol.methods.`**`deploy`**(*tid: UUID*, *agent_trigger_method: AgentTriggerMethod = AgentTriggerMethod.push_full_deploy*, *agents: list | None = None*)

Notify agents to perform a deploy now.

Parameters

- **`tid`** – The id of the environment.
- **`agent_trigger_method`** – Indicates whether the agents should perform a full or an incremental deploy.
- **`agents`** – Optional, names of specific agents to trigger

`inmanta.protocol.methods.`**`diff`**(*file_id_1: str*, *file_id_2: str*)

Returns the diff of the files with the two given ids

Parameters

- **`file_id_1`** – The identifier of the first file.
- **`file_id_2`** – The identifier of the second file.

Returns

A string representing the diff between the two files.

`inmanta.protocol.methods.`**`do_dryrun`**(*tid: UUID*, *id: UUID*, *agent: str*, *version: int*)

Do a dryrun on an agent

Parameters

- **`tid`** – The environment id
- **`id`** – The id of the dryrun
- **`agent`** – The agent to do the dryrun for
- **`version`** – The version of the model to dryrun

`inmanta.protocol.methods.`**`dryrun_list`**(*tid: UUID*, *version: int | None = None*)

Get the list of dry runs for an environment. The results are sorted by dry run id.

Parameters

- **`tid`** – The id of the environment
- **`version`** – Optional. Only for this version

`inmanta.protocol.methods.`**`dryrun_report`**(*tid: UUID*, *id: UUID*)

Create a dryrun report

Parameters

- **`tid`** – The id of the environment
- **`id`** – The version dryrun to report

`inmanta.protocol.methods.`**`dryrun_request`**(*tid: UUID*, *id: int*)

Do a dryrun

Parameters

- **`tid`** – The id of the environment

- `id` – The version of the CM to deploy

`inmanta.protocol.methods.`**`dryrun_update`**`(`*tid: UUID*, *id: UUID*, *resource: str*, *changes: dict*`)`

Store dryrun results at the server

Parameters

- `tid` – The id of the environment
- `id` – The version dryrun to report
- `resource` – The id of the resource
- `changes` – The required changes

`inmanta.protocol.methods.`**`get_agent_process`**`(`*id: UUID*`)`

Return a detailed report for a node

Parameters
`id` – The session id of the agent

Returns
The requested node

`inmanta.protocol.methods.`**`get_code`**`(`*tid: UUID*, *id: int*, *resource: str*`)`

Retrieve the source code associated with a specific version of a configuration model for a given resource in an environment.

Parameters

- `tid` – The id of the environment to which the code belongs.
- `id` – The version number of the configuration model.
- `resource` – The identifier of the resource. This should be a resource ID, not a resource version ID.

`inmanta.protocol.methods.`**`get_compile_queue`**`(`*tid: UUID*`)` → list[CompileRun]

Get the current compiler queue on the server, ordered by increasing *requested* timestamp.

Parameters
`tid` – The id of the environment for which to retrieve the compile queue.

Returns
A list of CompileRun objects representing the current state of the compiler queue, with each entry detailing a specific compile run.

`inmanta.protocol.methods.`**`get_environment`**`(`*id: UUID*, *versions: int | None = None*, *resources: int | None = None*`)`

Get an environment and all versions associated.

Parameters

- `id` – The id of the environment to return.
- `versions` – Optional. If provided and greater than 0, include this many of the most recent versions for this environment, ordered in descending order of their version number. If not provided or 0, no version information is included.
- `resources` – Optional. If provided and greater than 0, include a summary of the resources in the environment.

`inmanta.protocol.methods.`**`get_file`**`(`*id: str*`)`

Retrieve a file

Parameters
`id` – The id of the file to retrieve

`inmanta.protocol.methods.`**`get_param`**`(`*tid: UUID*, *id: str*, *resource_id: str | None = None*`)`

Get a parameter from the server.

Parameters

- **`tid`** – The id of the environment
- **`id`** – The name of the parameter
- **`resource_id`** – Optional. scope the parameter to resource (fact), if the resource id should not contain a version, the latest version is used

Returns

Returns the following status codes:

- 200: The parameter content is returned
- 404: The parameter is not found and unable to find it because its resource is not known to the server
- 410: The parameter has expired
- 503: The parameter is not found but its value is requested from an agent

`inmanta.protocol.methods.`**`get_parameter`**`(`*tid: UUID*, *agent: str*, *resource: dict*`)`

Get all parameters/facts known by the agents for the given resource

This method will not actually return them. This call wil register the request with the agent and return, The agent will push the parameters back to the server when they are available.

Parameters

- **`tid`** – The environment
- **`agent`** – The agent to get the parameters from
- **`resource`** – The resource to query the parameters from

`inmanta.protocol.methods.`**`get_project`**`(`*id: UUID*`)`

Get a project and a list of the ids of all environments.

Parameters

`id` – The id of the project to retrieve.

`inmanta.protocol.methods.`**`get_report`**`(`*id: UUID*`)`

Get a compile report from the server

Parameters

`id` – The id of the compile and its reports to fetch.

`inmanta.protocol.methods.`**`get_reports`**`(`*tid: UUID*, *start: str | None = None*, *end: str | None = None*, *limit: int | None = None*`)`

Return compile reports newer then start

Parameters

- **`tid`** – The id of the environment to get a report from
- **`start`** – Optional. Reports after start
- **`end`** – Optional. Reports before end
- **`limit`** – Optional. Maximum number of results, up to a maximum of 1000 If None, a default limit (set to 1000) is applied.

`inmanta.protocol.methods.`**`get_resource`**`(`*tid: UUID*, *id: str*, *logs: bool | None = None*, *status: bool | None = None*, *log_action:* ResourceAction *| None = None*, *log_limit: int = 0*`)`

Return a resource with the given id.

Parameters

- **tid** – The id of the environment this resource belongs to
- **id** – Get the resource with the given resource version id
- **logs** – Optional. Include the logs in the response
- **status** – Optional. Only return the status of the resource
- **log_action** – Optional. The log action to include, leave empty/none for all actions. Valid actions are one of the action strings in const.ResourceAction
- **log_limit** – Optional. Limit the number of logs included in the response, up to a maximum of 1000. To retrieve more entries, use /api/v2/resource_actions (`get_resource_actions()`) If None, a default limit (set to 1000) is applied.

`inmanta.protocol.methods.`**`get_resources_for_agent`**`(`*tid: UUID*, *agent: str*, *sid: UUID | None = None*, *version: int | None = None*, *incremental_deploy: bool = False*`)`

Return the most recent state for the resources associated with agent, or the version requested

Parameters

- **tid** – The environment ID this resource belongs to.
- **agent** – The agent name.
- **sid** – Optional. Session id of the agent (transparently added by agent client).
- **version** – Optional. The version to retrieve. If none, the latest available version is returned. With a specific version that version is returned, even if it has not been released yet.
- **incremental_deploy** – Optional. Indicates whether the server should only return the resources that changed since the previous deployment.

`inmanta.protocol.methods.`**`get_server_status`**`()` → StatusResponse

Get the status of the server

`inmanta.protocol.methods.`**`get_setting`**`(`*tid: UUID*, *id: str*`)`

Get the value of a setting.

Parameters

- **tid** – The id of the environment.
- **id** – The id of the setting to retrieve.

`inmanta.protocol.methods.`**`get_state`**`(`*tid: UUID*, *sid: UUID*, *agent: str*`)`

Get the state for this agent.

Parameters

- **tid** – The id of the environment.
- **sid** – The session ID associated with this agent.
- **agent** – The name of the agent.

Returns

A map with key enabled and value a boolean.

`inmanta.protocol.methods.`**`get_status`**`()`

A call from the server to the agent to report its status to the server

Returns

A map with report items

`inmanta.protocol.methods.`**`get_version`**(*tid: UUID*, *id: int*, *include_logs: bool | None = None*, *log_filter: str | None = None*, *limit: int | None = None*)

Get a particular version and a list of all resources in this version

Parameters

- **`tid`** – The id of the environment
- **`id`** – The id of the version to retrieve
- **`include_logs`** – Optional. If true, a log of all operations on all resources is included
- **`log_filter`** – Optional. Filter log to only include actions of the specified type
- **`limit`** – Optional. The maximal number of actions to return per resource (starting from the latest), up to a maximum of 1000. To retrieve more entries, use /api/v2/resource_actions (`get_resource_actions()`) If None, a default limit (set to 1000) is applied.

`inmanta.protocol.methods.`**`heartbeat`**(*sid: UUID*, *tid: UUID*, *endpoint_names: list*, *nodename: str*, *no_hang: bool = False*)

Send a heartbeat to the server

Parameters

- **`sid`** – The session ID used by this agent at this moment
- **`tid`** – The environment this node and its agents belongs to
- **`endpoint_names`** – The names of the endpoints on this node
- **`nodename`** – The name of the node from which the heart beat comes
- **`no_hang`** – Optional. don't use this call for long polling, but for connectivity check

also registered as API method, because it is called with an invalid SID the first time

`inmanta.protocol.methods.`**`heartbeat_reply`**(*sid: UUID*, *reply_id: UUID*, *data: dict*)

Send a reply back to the server

Parameters

- **`sid`** – The session ID used by this agent at this moment
- **`reply_id`** – The id data is a reply to
- **`data`** – The data as a response to the reply

`async` `inmanta.protocol.methods.`**`ignore_env`**(*obj: Any*, *metadata: dict*) → Any

This mapper only adds an env all for authz

`inmanta.protocol.methods.`**`is_compiling`**(*id: UUID*)

Is a compiler running for the given environment

Parameters

`id` – The environment id

`inmanta.protocol.methods.`**`list_agent_processes`**(*environment: UUID | None = None*, *expired: bool = True*, *start: UUID | None = None*, *end: UUID | None = None*, *limit: int | None = None*)

Return a list of all nodes and the agents for these nodes

Parameters

- **`environment`** – Optional. An optional environment. If set, only the agents that belong to this environment are returned
- **`expired`** – Optional. if true show expired processes, otherwise only living processes are shown. True by default

- **`start`** – Optional. Agent processes after start (sorted by sid in ASC)
- **`end`** – Optional. Agent processes before end (sorted by sid in ASC)
- **`limit`** – Optional. Maximum number of results, up to a maximum of 1000 If None, a default limit (set to 1000) is applied.

Raises

- *`BadRequest`* – limit parameter can not exceed 1000
- *`NotFound`* – The given environment id does not exist!

Returns

A list of nodes

`inmanta.protocol.methods.`**`list_agents`**`(`*tid: UUID*, *start: str | None = None*, *end: str | None = None*, *limit: int | None = None*`)`

List all agent for an environment

Parameters

- **`tid`** – The environment the agents are defined in
- **`start`** – Optional. Agent after start (sorted by name in ASC)
- **`end`** – Optional. Agent before end (sorted by name in ASC)
- **`limit`** – Optional. Maximum number of results, up to a maximum of 1000. If None, a default limit (set to 1000) is applied.

Raises

- *`BadRequest`* – limit parameter can not exceed 1000
- *`NotFound`* – The given environment id does not exist!

`inmanta.protocol.methods.`**`list_environments`**`()`

Returns a list of environments. The results are sorted by (project id, environment name, environment id).

`inmanta.protocol.methods.`**`list_params`**`(`*tid: UUID*, *query: dict = {}*`)`

List/query parameters in this environment. The results are ordered alphabetically by parameter name.

Parameters

- **`tid`** – The id of the environment
- **`query`** – Optional. A query to match against metadata

`inmanta.protocol.methods.`**`list_projects`**`()`

Returns a list of projects ordered alphabetically by name. The environments within each project are also sorted by name.

`inmanta.protocol.methods.`**`list_settings`**`(`*tid: UUID*`)`

List the settings in the current environment ordered by name alphabetically.

Parameters

`tid` – The id of the environment to list settings for.

`inmanta.protocol.methods.`**`list_versions`**`(`*tid: UUID*, *start: int | None = None*, *limit: int | None = None*`)`

Returns a list of all available versions, ordered by version number, descending

Parameters

- **`tid`** – The id of the environment
- **`start`** – Optional. parameter to control the amount of results that are returned. 0 is the latest version.
- **`limit`** – Optional. parameter to control the amount of results returned, up to a maximum of 1000. If None, a default limit (set to 1000) is applied.

inmanta.protocol.methods.**modify_environment**(*id: UUID*, *name: str*, *repository: str | None = None*, *branch: str | None = None*)

Modify the given environment.

Parameters

- **id** – The id of the environment to modify.
- **name** – The new name for the environment.
- **repository** – Optional. The URL of the repository.
- **branch** – Optional. The name of the branch in the repository.

If 'repository' or 'branch' is provided as None, the corresponding attribute of the environment remains unchanged.

inmanta.protocol.methods.**modify_project**(*id: UUID*, *name: str*)

Modify the given project.

Parameters

- **id** – The id of the project to modify.
- **name** – The new name for the project.

inmanta.protocol.methods.**notify_change**(*id: UUID*, *update: bool = True*, *metadata: dict = {}*)

Notify the server that the repository of the environment with the given id, has changed.

Parameters

- **id** – The id of the environment
- **update** – Optional. Update the model code and modules. Default value is true
- **metadata** – Optional. The metadata that indicates the source of the compilation trigger.

inmanta.protocol.methods.**notify_change_get**(*id: UUID*, *update: bool = True*)

Simplified GET version of the POST method

Parameters

- **id** – The id of the environment.
- **update** – Optional. Indicates whether to update the model code and modules. Defaults to true.

inmanta.protocol.methods.**put_version**(*tid: UUID*, *version: int*, *resources: list*, *resource_state: dict[ResourceIdStr, Literal[ResourceState.available, ResourceState.undefined]] = {}*, *unknowns: list[dict[str, UUID | bool | int | float | datetime | str]] | None = None*, *version_info: dict | None = None*, *compiler_version: str | None = None*, *resource_sets: dict[ResourceIdStr, str | None] = {}*, *pip_config: PipConfig | None = None*)

Store a new version of the configuration model

The version number must be obtained through the reserve_version call

Parameters

- **tid** – The id of the environment
- **version** – The version of the configuration model
- **resources** – A list of all resources in the configuration model (deployable)
- **resource_state** – A dictionary with the initial const.ResourceState per resource id. The ResourceState should be set to undefined when the resource depends on an unknown or available when it doesn't.

- **unknowns** – Optional. A list of unknown parameters that caused the model to be incomplete
- **version_info** – Optional. Module version information
- **compiler_version** – Optional. version of the compiler, if not provided, this call will return an error
- **resource_sets** – Optional. a dictionary describing which resource belongs to which resource set
- **pip_config** – Optional. Pip config used by this version

inmanta.protocol.methods.**release_version**(*tid: UUID*, *id: int*, *push: bool = False*, *agent_trigger_method: AgentTriggerMethod | None = None*)

Release version of the configuration model for deployment.

Parameters

- **tid** – The id of the environment
- **id** – The version of the CM to deploy
- **push** – Notify all agents to deploy the version
- **agent_trigger_method** – Optional. Indicates whether the agents should perform a full or an incremental deploy when push is true.

Returns

Returns the following status codes: 200: The version is released 404: The requested version does not exist 409: The requested version was already released

inmanta.protocol.methods.**resource_action_update**(*tid: UUID*, *resource_ids: list*, *action_id: UUID*, *action: ResourceAction*, *started: datetime | None = None*, *finished: datetime | None = None*, *status: ResourceState | DeprecatedResourceState | None = None*, *messages: list = []*, *changes: dict = {}*, *change: Change | None = None*, *send_events: bool = False*)

Send a resource update to the server

Parameters

- **tid** – The id of the environment this resource belongs to
- **resource_ids** – The resource with the given resource_version_id id from the agent
- **action_id** – A unique id to indicate the resource action that has be updated
- **action** – The action performed
- **started** – Optional. The timestamp when this action was started. When this action (action_id) has not been saved yet, started has to be defined.
- **finished** – Optional. The timestamp when this action was finished. Afterwards, no changes with the same action_id can be stored. The status field also has to be set.
- **status** – Optional. The current status of the resource (if known)
- **messages** – Optional. A list of log entries to add to this entry.
- **changes** – Optional. A dict of changes to this resource. The key of this dict indicates the attributes/fields that have been changed. The value contains the new value and/or the original value.
- **change** – Optional. The result of the changes
- **send_events** – Optional. [DEPRECATED] The value of this field is not used anymore.

`inmanta.protocol.methods.`**`resource_event`**(*tid: UUID*, *id: str*, *resource: str*, *send_events: bool*, *state: ResourceState*, *change: Change*, *changes={}*)

Tell an agent a resource it waits for has been updated

Parameters

- **`tid`** – The environment this agent is defined in
- **`id`** – The name of the agent
- **`resource`** – The resource ID of the resource being updated
- **`send_events`** – [DEPRECATED] The value of this field is not used anymore.
- **`state`** – State the resource acquired (deployed, skipped, canceled)
- **`change`** – The change that was made to the resource
- **`changes`** – Optional. The changes made to the resource

`inmanta.protocol.methods.`**`set_param`**(*tid: UUID*, *id: str*, *source: ParameterSource*, *value: str*, *resource_id: str | None = None*, *metadata: dict = {}*, *recompile: bool = False*, *expires: bool | None = None*)

Set a parameter on the server. If the parameter is an tracked unknown, it will trigger a recompile on the server. Otherwise, if the value is changed and recompile is true, a recompile is also triggered. [DEPRECATED] Please use the new endpoints */facts/<name>* and */parameters/<name>* instead.

Parameters

- **`tid`** – The id of the environment
- **`id`** – The name of the parameter
- **`source`** – The source of the parameter.
- **`value`** – The value of the parameter
- **`resource_id`** – Optional. Scope the parameter to resource (fact)
- **`metadata`** – Optional. Metadata about the parameter
- **`recompile`** – Optional. Whether to trigger a recompile
- **`expires`** – When setting a new parameter/fact: if set to None, then a sensible default will be provided (i.e. False for parameter and True for fact). When updating a parameter or fact, a None value will leave the existing value unchanged.

`inmanta.protocol.methods.`**`set_parameters`**(*tid: UUID*, *parameters: list*)

Set a parameter on the server

Parameters

- **`tid`** – The id of the environment
- **`parameters`** – A list of dicts with the following keys:
  - id The name of the parameter
  - source The source of the parameter. Valid values are defined in the ParameterSource enum (see: inmanta/const.py)
  - value The value of the parameter
  - resource_id Optionally, scope the parameter to resource (fact)
  - expires Defaults to true. Set to false to create a never expiring fact
  - metadata metadata about the parameter

inmanta.protocol.methods.**set_setting**(*tid: UUID*, *id: str*, *value: UUID | bool | int | float | datetime | str | dict[str, Any]*)

Set a value for a setting.

Parameters

- **tid** – The id of the environment.
- **id** – The id of the setting to set.
- **value** – The value to set for the setting.

inmanta.protocol.methods.**set_state**(*agent: str*, *enabled: bool*)

Set the state of the agent.

Parameters

- **agent** – The name of the agent.
- **enabled** – A boolean value indicating whether the agent should be paused (enabled=False) or unpaused (enabled=True).

inmanta.protocol.methods.**stat_file**(*id: str*)

Does the file exist

Parameters
id – The id of the file to check

inmanta.protocol.methods.**stat_files**(*files: list*)

Check which files exist in the given list

Parameters
files – A list of file ids to check

Returns
A list of files that do not exist.

inmanta.protocol.methods.**trigger**(*tid: UUID*, *id: str*, *incremental_deploy: bool*)

Request an agent to reload resources

Parameters

- **tid** – The environment this agent is defined in
- **id** – The name of the agent
- **incremental_deploy** – Indicates whether the agent should perform an incremental deploy or a full deploy

inmanta.protocol.methods.**trigger_agent**(*tid: UUID*, *id: str*)

Request the server to reload an agent

Parameters

- **tid** – The environment this agent is defined in
- **id** – The name of the agent

Returns
The requested node

inmanta.protocol.methods.**upload_code_batched**(*tid: UUID*, *id: int*, *resources: dict*)

Upload batches of code for various resources associated with a specific version of a configuration model in an environment.

Parameters

- **tid** – The id of the environment to which the code belongs.
- **id** – The version number of the configuration model.

- **resources** – A dictionary where each key is a string representing a resource type. For each resource type, the value is a dictionary. This nested dictionary's keys are file names, and each key maps to a tuple. This tuple contains three elements: the file name, the module name, and a list of requirements.

The endpoint validates that all provided file references are valid and checks for conflicts with existing code entries.

inmanta.protocol.methods.**upload_file**(*id: str*, *content: str*)

Upload a new file

Parameters

- **id** – The id of the file
- **content** – The base64 encoded content of the file

Module defining the v2 rest api

inmanta.protocol.methods_v2.**add_user**(*username: str*, *password: str*) → User

Add a new user to the system

Parameters

- **username** – The username of the new user. The username cannot be an empty string.
- **password** – The password of this new user. The password should be at least 8 characters long.

Raises

- *Conflict* – Raised when there is already a user with this user_name
- *BadRequest* – Raised when server authentication is not enabled

inmanta.protocol.methods_v2.**agent_action**(*tid: UUID*, *name: str*, *action: AgentAction*) → None

Execute an action on an agent

Parameters

- **tid** – The environment this agent is defined in.
- **name** – The name of the agent.
- **action** – The type of action that should be executed on an agent. Pause and unpause can only be used when the environment is not halted, while the on_resume actions can only be used when the environment is halted. * pause: A paused agent cannot execute any deploy operations. * unpause: A unpaused agent will be able to execute deploy operations. * keep_paused_on_resume: The agent will still be paused when the environment is resumed * unpause_on_resume: The agent will be unpaused when the environment is resumed

Raises

Forbidden – The given environment has been halted and the action is pause/unpause, or the environment is not halted and the action is related to the on_resume behavior

inmanta.protocol.methods_v2.**all_agents_action**(*tid: UUID*, *action: AgentAction*) → None

Execute an action on all agents in the given environment.

Parameters

- **tid** – The environment of the agents.
- **action** – The type of action that should be executed on the agents. Pause and unpause can only be used when the environment is not halted, while the on_resume actions can only be used when the environment is halted. * pause: A paused agent cannot execute any deploy operations. * unpause: A unpaused agent will be able to execute deploy

operations. * keep_paused_on_resume: The agents will still be paused when the environment is resumed * unpause_on_resume: The agents will be unpaused when the environment is resumed

Raises

Forbidden – The given environment has been halted and the action is pause/unpause, or the environment is not halted and the action is related to the on_resume behavior

`inmanta.protocol.methods_v2.`**`compile_details`**`(`*tid: UUID*, *id: UUID*`)` → CompileDetails

Parameters

- **tid** – The id of the environment in which the compilation process occurred.
- **id** – The id of the compile for which the details are being requested.

Returns

The details of a compile

Raises

NotFound – This exception is raised when the referenced environment or compile is not found

`inmanta.protocol.methods_v2.`**`delete_user`**`(`*username: str*`)` → None

Delete a user from the system with given username.

Parameters

username – The username to delete

Raises

- *NotFound* – Raised when the user does not exist
- *BadRequest* – Raised when server authentication is not enabled

`inmanta.protocol.methods_v2.`**`discovered_resource_create`**`(`*tid: UUID*, *discovered_resource_id: str*, ***kwargs: object*`)` → None

create a discovered resource.

Parameters

- **tid** – The id of the environment this resource belongs to
- **discovered_resource_id** – The id of the discovered_resource
- ****kwargs** – The following arguments are supported: values: The values associated with the discovered_resource

`inmanta.protocol.methods_v2.`**`discovered_resource_create_batch`**`(`*tid: UUID*, *discovered_resources: list[DiscoveredResource]*`)` → None

create multiple discovered resource in the DB :param tid: The id of the environment this resource belongs to :param discovered_resources: List of discovered_resources containing the discovered_resource_id and values for each resource

`inmanta.protocol.methods_v2.`**`discovered_resources_get`**`(`*tid: UUID*, *discovered_resource_id: ResourceIdStr*`)` → DiscoveredResource

Get a single discovered resource.

Parameters

- **tid** – the id of the environment in which to get the discovered resource.
- **discovered_resource_id** – The id of the discovered resource

inmanta.protocol.methods_v2.**discovered_resources_get_batch**(*tid: UUID*, *limit: int | None = None*, *start: str | None = None*, *end: str | None = None*, *sort: str = 'discovered_resource_id.asc'*) → list[DiscoveredResource]

Parameters

- **tid** – The id of the environment this resource belongs to
- **limit** – Limit the number of instances that are returned
- **start** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **end** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **sort** – Return the results sorted according to the parameter value. The following sorting attributes are supported: 'discovered_resource_id'. The following orders are supported: 'asc', 'desc'

Returns

A list of all matching released resources

Raises

- *NotFound* – This exception is raised when the referenced environment is not found
- *BadRequest* – When the parameters used for filtering, sorting or paging are not valid

inmanta.protocol.methods_v2.**dryrun_trigger**(*tid: UUID*, *version: int*) → UUID

Trigger a new dryrun

Parameters

- **tid** – The id of the environment
- **version** – The version of the configuration model to execute the dryrun for

Raises

NotFound – This exception is raised when the referenced environment or version is not found

Returns

The id of the new dryrun

inmanta.protocol.methods_v2.**environment_clear**(*id: UUID*) → None

Clear all data from this environment. The environment will be temporarily halted during the decommissioning process.

Parameters

id – The uuid of the environment.

Raises

- *NotFound* – The given environment doesn't exist.
- *Forbidden* – The given environment is protected.

inmanta.protocol.methods_v2.**environment_create**(*project_id: UUID*, *name: str*, *repository: str | None = None*, *branch: str | None = None*, *environment_id: UUID | None = None*, *description: str = ''*, *icon: str = ''*) → Environment

Create a new environment

Parameters

- **project_id** – The id of the project this environment belongs to

- **`name`** – The name of the environment. The name should be unique for each project.
- **`repository`** – The url (in git form) of the repository
- **`branch`** – The name of the branch in the repository
- **`environment_id`** – A unique environment id, if none an id is allocated by the server
- **`description`** – The description of the environment, maximum 255 characters
- **`icon`** – The data-url of the icon of the environment. It should follow the pattern *<mime-type>;base64,<image>*, where <mime-type> is one of: 'image/png', 'image/jpeg', 'image/webp', 'image/svg+xml', and <image> is the image in the format matching the specified mime-type, and base64 encoded. The length of the whole string should be maximum 64 kb.

Raises

`BadRequest` – When the parameters supplied are not valid.

`inmanta.protocol.methods_v2.`**`environment_create_token`**`(`*tid: UUID*, *client_types: list[str]*, *idempotent: bool = True*`)` → str

Create or get a new token for the given client types. Tokens generated with this call are scoped to the current environment.

Parameters

- **`tid`** – The environment id
- **`client_types`** – The client types for which this token is valid (api, agent, compiler)
- **`idempotent`** – The token should be idempotent, such tokens do not have an expire or issued at set so their value will not change.

`inmanta.protocol.methods_v2.`**`environment_delete`**`(`*id: UUID*`)` → None

Delete the given environment and all related data.

Parameters

`id` – The uuid of the environment.

Raises

- *`NotFound`* – The given environment doesn't exist.
- *`Forbidden`* – The given environment is protected.

`inmanta.protocol.methods_v2.`**`environment_get`**`(`*id: UUID*, *details: bool = False*`)` → Environment

Get an environment and all versions associated

Parameters

- **`id`** – The id of the environment to return
- **`details`** – Whether to include the icon and description of the environment

`inmanta.protocol.methods_v2.`**`environment_list`**`(`*details: bool = False*`)` → list[Environment]

Returns a list of environments

Parameters

`details` – Whether to include the icon and description of the environments in the results

`inmanta.protocol.methods_v2.`**`environment_modify`**`(`*id: UUID*, *name: str*, *repository: str | None = None*, *branch: str | None = None*, *project_id: UUID | None = None*, *description: str | None = None*, *icon: str | None = None*`)` → Environment

Modify the given environment The optional parameters that are unspecified will be left unchanged by the update.

Parameters

- **id** – The id of the environment
- **name** – The name of the environment
- **repository** – The url (in git form) of the repository
- **branch** – The name of the branch in the repository
- **project_id** – The id of the project the environment belongs to
- **description** – The description of the environment, maximum 255 characters
- **icon** – The data-url of the icon of the environment. It should follow the pattern *<mime-type>;base64,<image>* , where <mime-type> is one of: 'image/png', 'image/jpeg', 'image/webp', 'image/svg+xml', and <image> is the image in the format matching the specified mime-type, and base64 encoded. The length of the whole string should be maximum 64 kb. The icon can be removed by setting this parameter to an empty string.

Raises

- *BadRequest* – When the parameters supplied are not valid.
- *NotFound* – The given environment doesn't exist.

inmanta.protocol.methods_v2.**environment_setting_delete**(*tid: UUID*, *id: str*) → ReturnValue[None]

Reset the given setting to its default value.

Parameters

- **tid** – The id of the environment from which the setting is to be deleted.
- **id** – The identifier of the setting to be deleted.

inmanta.protocol.methods_v2.**environment_setting_get**(*tid: UUID*, *id: str*) → EnvironmentSettingsReponse

Retrieve a specific setting from an environment's configuration.

Parameters

- **tid** – The id of the environment from which the setting is being retrieved.
- **id** – The id of the setting to be retrieved.

inmanta.protocol.methods_v2.**environment_settings_list**(*tid: UUID*) → EnvironmentSettingsReponse

List the settings in the current environment ordered by name alphabetically.

Parameters
tid – The id of the environment for which the list of settings is being requested.

inmanta.protocol.methods_v2.**environment_settings_set**(*tid: UUID*, *id: str*, *value: bool | int | float | str | dict[str, str | int | bool]*) → ReturnValue[None]

Set a specific setting in an environment's configuration.

Parameters

- **tid** – The id of the environment where the setting is to be set or updated.
- **id** – The id of the setting to be set or updated.
- **value** – The new value for the setting.

inmanta.protocol.methods_v2.**get_agent_process_details**(*tid: UUID*, *id: UUID*, *report: bool = False*) → AgentProcess

Get the details of an agent process

Parameters

- **tid** – Id of the environment
- **id** – The id of the specific agent process
- **report** – Whether to include a report from the agent or not

Returns
The details of an agent process

Raises
NotFound – This exception is raised when the referenced environment or agent process is not found

`inmanta.protocol.methods_v2.get_agents`(*tid: UUID*, *limit: int | None = None*, *start: datetime | bool | str | None = None*, *end: datetime | bool | str | None = None*, *first_id: str | None = None*, *last_id: str | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'name.asc'*) → list[Agent]

Get all of the agents in the given environment

Parameters

- **tid** – The id of the environment the agents should belong to.
- **limit** – Limit the number of agents that are returned.
- **start** – The lower limit for the order by column (exclusive).
- **first_id** – The name to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values.
- **last_id** – The name to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values. Only one of 'start' and 'end' should be specified at the same time.
- **end** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **filter** – Filter the list of returned agents. Filtering by 'name', 'process_name' and 'status' is supported.
- **sort** – Return the results sorted according to the parameter value. Sorting by 'name', 'process_name', 'status', 'paused' and 'last_failover' is supported. The following orders are supported: 'asc', 'desc'

Returns
A list of all matching agents

Raises

- *NotFound* – This exception is raised when the referenced environment is not found
- *BadRequest* – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.get_all_facts`(*tid: UUID*, *limit: int | None = None*, *first_id: UUID | None = None*, *last_id: UUID | None = None*, *start: str | None = None*, *end: str | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'name.asc'*) → list[Fact]

List the facts in an environment.

Parameters

- **tid** – The id of the environment
- **limit** – Limit the number of facts that are returned
- **first_id** – The fact id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values

- **`last_id`** – The fact id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned facts. The following options are available: name: filter by the name of the fact resource_id: filter by the resource_id of the fact expires: filter on whether the fact expires or not
- **`sort`** – Return the results sorted according to the parameter value. The following sorting attributes are supported: 'name', 'resource_id'. The following orders are supported: 'asc', 'desc'

Returns
A list of all matching facts

Raises

- *`NotFound`* – This exception is raised when the referenced environment is not found
- *`BadRequest`* – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`get_api_docs`**`(`*format: ApiDocsFormat | None = ApiDocsFormat.swagger*`)` → ReturnValue[OpenAPI | str]

Get the OpenAPI definition of the API

Parameters
`format` – Use 'openapi' to get the schema in json format, leave empty or use 'swagger' to get the Swagger-UI view

`inmanta.protocol.methods_v2.`**`get_compile_data`**`(`*id: UUID*`)` → *CompileData* | None

Get the compile data for the given compile request.

Parameters
`id` – The id of the compile.

`inmanta.protocol.methods_v2.`**`get_compile_reports`**`(`*tid: UUID, limit: int | None = None, first_id: UUID | None = None, last_id: UUID | None = None, start: datetime | None = None, end: datetime | None = None, filter: dict[str, list[str]] | None = None, sort: str = 'requested.desc'*`)` → list[CompileReport]

Get the compile reports from an environment.

Parameters

- **`tid`** – The id of the environment
- **`limit`** – Limit the number of instances that are returned
- **`first_id`** – The id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **`last_id`** – The id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.

- **filter** – Filter the list of returned compile reports. Filters should be specified with the syntax *?filter.<filter_key>=value*, for example *?filter.success=True* It's also possible to provide multiple values for the same filter, in this case resources are returned, if they match any of these filter values. For example: *?filter.requested=ge:2021-08-18T09:21:30.568353&filter.requested=lt:2021-08-18T10:21:30.568353* returns compile reports that were requested between the specified dates. Multiple different filters narrow the results however (they are treated as an 'AND' operator). For example *?filter.success=True&filter.completed=True* returns compile reports that are completed and successful. The following options are available: success: whether the compile was successful or not started: whether the compile has been started or not completed: whether the compile has been completed or not

  requested: return the logs matching the timestamp constraints. Valid constraints are of the form
  "<lt|le|gt|ge>:<x>". The expected format is YYYY-MM-DDTHH:mm:ss.ssssss, so an ISO-8601 datetime string, in UTC timezone. Specifying microseconds is optional. For example: *?filter.requested=ge:2021-08-18T09:21:30.568353&filter.requested=lt:2021-08-18T10:21:30*. Multiple constraints can be specified, in which case only compile reports that match all constraints will be returned.

- **sort** – Return the results sorted according to the parameter value. It should follow the pattern *?sort=<attribute_to_sort_by>.<order>*, for example *?sort=requested.desc* (case insensitive). Only sorting by the *requested* timestamp is supported. The following orders are supported: 'asc', 'desc'

Returns
A list of all matching compile reports

Raises

- *NotFound* – This exception is raised when the referenced environment is not found
- *BadRequest* – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`get_diff_of_versions`**(*tid: UUID*, *from_version: int*, *to_version: int*) → list[ResourceDiff]

Compare two versions of desired states, and provide the difference between them, with regard to their resources and the attributes of these resources. Resources that are the same in both versions are not mentioned in the results.

A resource diff describes whether the resource was 'added', 'modified' or 'deleted', and what the values of their attributes were in the versions. The values are also returned in a stringified, easy to compare way, which can be used to calculate a *git diff*-like summary of the changes.

Parameters

- **tid** – The id of the environment
- **from_version** – The (lower) version number to compare
- **to_version** – The other (higher) version number to compare

Returns
The resource diffs between from_version and to_version

Raises

- *NotFound* – This exception is raised when the referenced environment or versions are not found
- *BadRequest* – When the version parameters are not valid

`inmanta.protocol.methods_v2.`**`get_dryrun_diff`**(*tid: UUID*, *version: int*, *report_id: UUID*) → DryRunReport

Get the report of a dryrun, describing the changes a deployment would make, with the difference between the current and target states provided in a form similar to the desired state diff endpoint.

Parameters

- **`tid`** – The id of the environment
- **`version`** – The version of the configuration model the dryrun belongs to
- **`report_id`** – The dryrun id to calculate the diff for

Raises

`NotFound` – This exception is raised when the referenced environment or version is not found

Returns

The dryrun report, with a summary and the list of differences.

`inmanta.protocol.methods_v2.`**`get_environment_metrics`**(*tid: UUID*, *metrics: list[str]*, *start_interval: datetime*, *end_interval: datetime*, *nb_datapoints: int*, *round_timestamps: bool = False*) → EnvironmentMetricsResult

Obtain metrics about the given environment for the given time interval.

Parameters

- **`tid`** – The id of the environment for which the metrics have to be collected.
- **`metrics`** – List of names of metrics that have to be returned.
- **`start_interval`** – The start of the time window for which the metrics should be returned.
- **`end_interval`** – The end of the time window for which the metrics should be returned.
- **`nb_datapoints`** – The amount of datapoint that will be returned within the given time interval for each metric.
- **`round_timestamps`** – If this parameter is set to True, the timestamps in the reply will be rounded to a full hour. All time windows in the reply will have an equal size. To achieve this the start_interval, end_interval and nb_datapoint in the reply may differ from the ones requested.
  - The start_interval may be smaller than requested
  - The end_interval may be larger than requested
  - The nb_datapoints may be larger than requested

Raises

- *`BadRequest`* – start_interval >= end_interval
- *`BadRequest`* – nb_datapoints < 0
- *`BadRequest`* – The provided metrics list is an empty list.
- *`BadRequest`* – The start_interval and end_interval are not separated from each other by at least nb_datapoints minutes separated from each other.
- *`BadRequest`* – The round_timestamps parameter is set to True and the amount of hours between start_interval and end_interval is less than the requested number of datapoints.

`inmanta.protocol.methods_v2.`**`get_fact`**(*tid: UUID*, *rid: ResourceIdStr*, *id: UUID*) → Fact

Get one specific fact :param tid: The id of the environment :param rid: The id of the resource :param id: The id of the fact :return: A specific fact corresponding to the id :raise NotFound: This status code is returned when the referenced environment or fact is not found `inmanta.protocol.methods_v2.`**`get_facts`**(*tid: UUID*, *rid: ResourceIdStr*) → list[Fact]

Get the facts related to a specific resource. The results are sorted alphabetically by name. :param tid: The id of the environment :param rid: Id of the resource :return: The facts related to this resource :raise NotFound: This status code is returned when the referenced environment is not found

`inmanta.protocol.methods_v2.`**`get_notification`**(*tid: UUID*, *notification_id: UUID*) → Notification

Get a single notification

Parameters

- **`tid`** – The id of the environment
- **`notification_id`** – The id of the notification

Returns
The notification with the specified id

Raises
`NotFound` – When the referenced environment or notification is not found

`inmanta.protocol.methods_v2.`**`get_parameters`**(*tid: UUID*, *limit: int | None = None*, *first_id: UUID | None = None*, *last_id: UUID | None = None*, *start: datetime | str | None = None*, *end: datetime | str | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'name.asc'*) → list[Parameter]

List the parameters in an environment

Parameters

- **`tid`** – The id of the environment
- **`limit`** – Limit the number of parameters that are returned
- **`first_id`** – The parameter id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **`last_id`** – The parameter id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned parameters.

  The following options are available:

  - name: filter by the name of the parameter
  - source: filter by the source of the parameter
  - updated: filter by the updated time of the parameter
- **`sort`** – Return the results sorted according to the parameter value. The following sorting attributes are supported: 'name', 'source', 'updated'. The following orders are supported: 'asc', 'desc'

Returns
A list of all matching parameters

Raises

- *`NotFound`* – This exception is raised when the referenced environment is not found
- *`BadRequest`* – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`get_pip_config`**(*tid: UUID*, *version: int*) → PipConfig | None

Get the pip config for the given version

Parameters

- **`tid`** – The id of the environment
- **`version`** – The id of the model version

Raises

`NotFound` – Raised when the version or environment is not found

`inmanta.protocol.methods_v2.`**`get_resource_actions`**(*tid: UUID*, *resource_type: str | None = None*, *agent: str | None = None*, *attribute: str | None = None*, *attribute_value: str | None = None*, *log_severity: str | None = None*, *limit: int | None = 0*, *action_id: UUID | None = None*, *first_timestamp: datetime | None = None*, *last_timestamp: datetime | None = None*, *exclude_changes: list[Change] | None = None*) → ReturnValue[list[ResourceAction]]

Return resource actions matching the search criteria.

Parameters

- **`tid`** – The id of the environment this resource belongs to
- **`resource_type`** – The resource entity type that should be queried
- **`agent`** – Agent name that is used to filter the results
- **`attribute`** – Attribute name used for filtering
- **`attribute_value`** – Attribute value used for filtering. Attribute and attribute value should be supplied together.
- **`log_severity`** – Only include ResourceActions which have a log message with this severity.
- **`limit`** – Limit the number of resource actions included in the response, up to 1000
- **`action_id`** – Start the query from this action_id. To be used in combination with either the first or last timestamp.
- **`first_timestamp`** – Limit the results to resource actions that started later than the value of this parameter (exclusive)
- **`last_timestamp`** – Limit the results to resource actions that started earlier than the value of this parameter (exclusive). Only the first_timestamp or last_timestamp parameter should be supplied
- **`exclude_changes`** – only return ResourceActions where the change type is different from the one in this list.

Returns

The list of matching Resource Actions. The order is ascending if first_timestamp is provided, otherwise descending. If a limit is specified, also return links to the next and previous pages. The "next" page refers to actions that started earlier, while the "prev" page refers to actions that started later.

Raises

`BadRequest` – When the supplied parameters are not valid.

`inmanta.protocol.methods_v2.`**`get_resource_events`**(*tid: UUID*, *rvid: ResourceVersionIdStr*, *exclude_change: Change | None = None*) → dict[ResourceIdStr, list[ResourceAction]]

Return relevant events for a resource, i.e. all deploy actions for each of its dependencies since this resources' last successful deploy or all deploy actions if this resources hasn't been deployed before. The resource actions are sorted in descending order according to their started timestamp. If exclude_change is set, exclude all resource actions with this specific type of change.

This method searches through all versions of this resource. This method should only be called when a deploy is in progress.

Parameters

- **`tid`** – The id of the environment this resource belongs to
- **`rvid`** – The id of the resource to get events for.
- **`exclude_change`** – Exclude all resource actions with this specific type of change.

Raises

`BadRequest` – When this endpoint in called while the resource with the given resource version is not in the deploying state.

`inmanta.protocol.methods_v2.`**`get_resources_in_version`**(*tid: UUID*, *version: int*, *limit: int | None = None*, *first_id: ResourceVersionIdStr | None = None*, *last_id: ResourceVersionIdStr | None = None*, *start: str | None = None*, *end: str | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'resource_type.desc'*) → list[VersionedResource]

Get the resources that belong to a specific version.

Parameters

- **`tid`** – The id of the environment
- **`version`** – The version number
- **`limit`** – Limit the number of resources that are returned
- **`first_id`** – The resource_version_id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **`last_id`** – The resource_version_id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned resources. The following options are available: agent: filter by the agent name of the resource resource_type: filter by the type of the resource resource_id_value: filter by the attribute values of the resource
- **`sort`** – Return the results sorted according to the parameter value. The following sorting attributes are supported: 'resource_type', 'agent', 'resource_id_value'. The following orders are supported: 'asc', 'desc'

Returns

A list of all matching resources

Raises

- *`NotFound`* – This exception is raised when the referenced environment is not found

- `BadRequest` – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`get_source_code`**(*tid: UUID*, *version: int*, *resource_type: str*) → list[Source]

Get the code for the given version and the given resource :param tid: The id of the environment :param version: The id of the model version :param resource_type: The type name of the resource :raises NotFound: Raised when the version or type is not found

`inmanta.protocol.methods_v2.`**`halt_environment`**(*tid: UUID*) → None

Halt all orchestrator operations for an environment. The environment will enter a state where all agents are paused and can not be unpaused. Incoming compile requests will still be queued but compilation will halt. Normal operation can be restored using the *resume_environment* endpoint.

Parameters

`tid` – The environment id

Raises

`NotFound` – The given environment doesn't exist.

`inmanta.protocol.methods_v2.`**`list_desired_state_versions`**(*tid: UUID*, *limit: int | None = None*, *start: int | None = None*, *end: int | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'version.desc'*) → list[DesiredStateVersion]

Get the desired state versions from an environment.

Parameters

- **`tid`** – The id of the environment
- **`limit`** – Limit the number of versions that are returned
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned desired state versions. Filtering by 'version' range, 'date' range and 'status' is supported.
- **`sort`** – Return the results sorted according to the parameter value. Only sorting by 'version' is supported. The following orders are supported: 'asc', 'desc'

Returns

A list of all matching compile reports

Raises

- `NotFound` – This exception is raised when the referenced environment is not found
- `BadRequest` – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`list_dryruns`**(*tid: UUID*, *version: int*) → list[DryRun]

Query a list of dry runs for a specific version

Parameters

- **`tid`** – The id of the environment
- **`version`** – The configuration model version to return dryruns for

Raises

`NotFound` – This exception is raised when the referenced environment or version is not found

Returns

The list of dryruns for the specified version in descending order by date `inmanta.protocol.methods_v2.`**`list_notifications`**`(`*tid: UUID*, *limit: int | None = None*, *first_id: UUID | None = None*, *last_id: UUID | None = None*, *start: datetime | None = None*, *end: datetime | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'created.desc'*`)` → list[Notification]

List the notifications in an environment.

Parameters

- **`tid`** – The id of the environment
- **`limit`** – Limit the number of notifications that are returned
- **`first_id`** – The notification id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **`last_id`** – The notification id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned notifications. The following options are available: read: Whether the notification was read or not cleared: Whether the notification was cleared or not severity: Filter by the severity field of the notifications title: Filter by the title of the notifications message: Filter by the message of the notifications
- **`sort`** – Return the results sorted according to the parameter value. Only sorting by the 'created' date is supported. The following orders are supported: 'asc', 'desc'

Returns

A list of all matching notifications

Raises

- *`NotFound`* – This exception is raised when the referenced environment is not found
- *`BadRequest`* – When the parameters used for filtering or paging are not valid

`inmanta.protocol.methods_v2.`**`list_users`**`()` → list[User]

List all users

Returns

A list of all users

`inmanta.protocol.methods_v2.`**`login`**`(`*username: str*, *password: str*`)` → ReturnValue[LoginReturn]

Login a user. When the login succeeds an authentication header is returned with the Bearer token set.

Parameters

- **`username`** – The user to login
- **`password`** – The password of this user

Raises

`UnauthorizedException` – Raised when the login failed or if server authentication is not enabled

`inmanta.protocol.methods_v2.`**`project_create`**`(`*name: str*, *project_id: UUID | None = None*`)` → Project

Create a new project

Parameters

- **`name`** – The name of the project

- **project_id** – A unique uuid, when it is not provided the server generates one

inmanta.protocol.methods_v2.**project_delete**(*id: UUID*) → None

Delete the given project and all related data

Parameters
id – The id of the project to be deleted.

inmanta.protocol.methods_v2.**project_get**(*id: UUID*, *environment_details: bool = False*) → Project

Get a project and a list of the environments under this project

Parameters

- **id** – The id for which the project's details are being requested.
- **environment_details** – Whether to include the icon and description of the environments in the results

inmanta.protocol.methods_v2.**project_list**(*environment_details: bool = False*) → list[Project]

Returns a list of projects ordered alphabetically by name. The environments within each project are also sorted by name.

Parameters
environment_details – Whether to include the icon and description of the environments in the results

inmanta.protocol.methods_v2.**project_modify**(*id: UUID*, *name: str*) → Project

Rename the given project

Parameters

- **id** – The id of the project to be modified.
- **name** – The new name for the project. This string value will replace the current name of the project.

inmanta.protocol.methods_v2.**promote_desired_state_version**(*tid: UUID*, *version: int*, *trigger_method: PromoteTriggerMethod | None = None*) → None

Promote a desired state version, making it the active version in the environment.

Parameters

- **tid** – The id of the environment
- **version** – The number of the version to promote
- **trigger_method** – If set to 'push_incremental_deploy' or 'push_full_deploy', the agents will perform an incremental or full deploy, respectively. If set to 'no_push', the new version is not pushed to the agents. If the parameter is not set (or set to null), the new version is pushed and the environment setting 'environment_agent_trigger_method' decides if the deploy should be full or incremental

inmanta.protocol.methods_v2.**put_partial**(*tid: UUID*, *resource_state: dict[ResourceIdStr, Literal[ResourceState.available, ResourceState.undefined]] | None = None*, *unknowns: list[dict[str, UUID | bool | int | float | datetime | str]] | None = None*, *resource_sets: dict[ResourceIdStr, str | None] | None = None*, *removed_resource_sets: list[str] | None = None*, *pip_config: PipConfig | None = None*, ***kwargs: object*) → ReturnValue[int]

Store a new version of the configuration model after a partial recompile. The partial is applied on top of the latest version. Dynamically acquires a new version and serializes concurrent calls. Python code for the new version is copied from the base version.

Concurrent put_partial calls are safe from race conditions provided that their resource sets are disjunct. A put_version call concurrent with a put_partial is not guaranteed to be safe. It is the caller's responsibility to appropriately serialize them with respect to one another. The caller must ensure the reserve_version + put_version operation is atomic with respect to put_partial. In other words, put_partial must not be called in the window between reserve_version and put_version. If not respected, either the full or the partial export might be immediately stale, and future exports will only be applied on top of the non-stale one.

Parameters

- **tid** – The id of the environment
- **resource_state** – A dictionary with the initial const.ResourceState per resource id. The ResourceState should be set to undefined when the resource depends on an unknown or available when it doesn't.
- **unknowns** – A list of unknown parameters that caused the model to be incomplete
- **resource_sets** – a dictionary describing which resources belong to which resource set
- **removed_resource_sets** – a list of resource_sets that should be deleted from the model
- ****kwargs** – The following arguments are supported: * resources: a list of resource objects. Since the version is not known yet resource versions should be set to 0. * version_info: Model version information
- **pip_config** – Pip config used by this version

Returns
The newly stored version number.

`inmanta.protocol.methods_v2.`**`reserve_version`**(*tid: UUID*) → int

Reserve a version number in this environment.

Parameters
tid – The id of the environment in which the version number is to be reserved.

`inmanta.protocol.methods_v2.`**`resource_deploy_done`**(*tid: UUID*, *rvid: ResourceVersionIdStr*, *action_id: UUID*, *status: ResourceState*, *messages: list[LogLine] = []*, *changes: dict[str, AttributeStateChange] = {}*, *change: Change | None = None*) → None

Report to the server that an agent has finished the deployment of a certain resource.

Parameters

- **tid** – The id of the environment the resource belongs to
- **rvid** – The resource version id of the resource for which the deployment is finished.
- **action_id** – A unique ID associated with this resource deployment action. This should be the same ID that was passed to the */resource/<resource_id>/deploy/start* API call.
- **status** – The current status of the resource (if known)
- **messages** – A list of log entries produced by the deployment action.
- **changes** – A dict of changes to this resource. The key of this dict indicates the attributes/fields that have been changed. The value contains the new value and/or the original value.
- **change** – The type of change that was done the given resource.

`inmanta.protocol.methods_v2.`**`resource_deploy_start`**(*tid: UUID*, *rvid: ResourceVersionIdStr*, *action_id: UUID*) → dict[ResourceVersionIdStr, ResourceState]

Report to the server that the agent will start the deployment of the given resource.

Parameters

- **tid** – The id of the environment the resource belongs to
- **rvid** – The resource version id of the resource for which the deployment will start
- **action_id** – A unique id used to track the action of this deployment

Returns

A dict mapping the resource version id of each dependency of resource_id to the last deployment status of that resource.

`inmanta.protocol.methods_v2.`**`resource_details`**(*tid: UUID*, *rid: ResourceIdStr*) → ReleasedResourceDetails

Parameters

- **tid** – The id of the environment from which the resource's details are being requested.
- **rid** – The unique identifier (ResourceIdStr) of the resource. This value specifies the particular resource for which detailed information is being requested.

Returns

The details of the latest released version of a resource

Raises

NotFound – This exception is raised when the referenced environment or resource is not found

`inmanta.protocol.methods_v2.`**`resource_did_dependency_change`**(*tid: UUID*, *rvid: ResourceVersionIdStr*) → bool

Returns True iff this resources' events indicate a change in its dependencies since the resource's last deployment.

This method searches through all versions of this resource. This method should only be called when a deploy is in progress.

Parameters

- **tid** – The id of the environment this resource belongs to
- **rvid** – The id of the resource.

Raises

BadRequest – When this endpoint in called while the resource with the given resource version is not in the deploying state.

`inmanta.protocol.methods_v2.`**`resource_history`**(*tid: UUID*, *rid: ResourceIdStr*, *limit: int | None = None*, *first_id: str | None = None*, *last_id: str | None = None*, *start: datetime | None = None*, *end: datetime | None = None*, *sort: str = 'date.desc'*) → list[ResourceHistory]

Parameters

- **tid** – The id of the environment this resource belongs to
- **rid** – The id of the resource
- **limit** – Limit the number of instances that are returned
- **first_id** – The attribute_hash to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **last_id** – The attribute_hash to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **start** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.

- **end** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **sort** – Return the results sorted according to the parameter value. It should follow the pattern *<attribute_to_sort_by>.<order>*, for example *date.desc* (case insensitive). Sorting by *date* is supported. The following orders are supported: 'asc', 'desc'

Returns
The history of a resource, according to its attributes

Raises

- ***NotFound*** – This exception is raised when the referenced environment is not found
- ***BadRequest*** – When the parameters used for sorting or paging are not valid

inmanta.protocol.methods_v2.**resource_list**(*tid: UUID*, *limit: int | None = None*, *first_id: ResourceVersionIdStr | None = None*, *last_id: ResourceVersionIdStr | None = None*, *start: str | None = None*, *end: str | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'resource_type.desc'*, *deploy_summary: bool = False*) → list[LatestReleasedResource]

Parameters

- **tid** – The id of the environment this resource belongs to
- **limit** – Limit the number of instances that are returned
- **first_id** – The resource_version_id to use as a continuation token for paging, in combination with the 'start' value, because the order by column might contain non-unique values
- **last_id** – The resource_version_id to use as a continuation token for paging, in combination with the 'end' value, because the order by column might contain non-unique values
- **start** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **end** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **filter** – Filter the list of returned resources. Filters should be specified with the syntax *?filter.<filter_key>=value*, for example *?filter.status=deployed* It's also possible to provide multiple values for the same filter, in this case resources are returned, if they match any of these filter values. For example: *?filter.status=deployed&filter.status=available* returns instances with either of the statuses deployed or available. Multiple different filters narrow the results however (they are treated as an 'AND' operator). For example *filter.status=deployed&filter.agent=internal* returns resources with 'deployed' status, where the 'agent' is set to 'internal_agent'. The following options are available: agent: filter by the agent of the resource resource_type: filter by the type of the resource resource_id_value: filter by the attribute values of the resource status: filter by the current status of the resource. For status filters it's also possible to invert the condition with '!', for example *filter.status=!orphaned* will return all the resources that are not in 'orphaned' state The values for the 'agent', 'resource_type' and 'value' filters are matched partially.
- **sort** – Return the results sorted according to the parameter value. It should follow the pattern *<attribute_to_sort_by>.<order>*, for example *resource_type.desc* (case insensitive). The following sorting attributes are supported: 'resource_type', 'agent', 'resource_id_value', 'status'. The following orders are supported: 'asc', 'desc'
- **deploy_summary** – If set to true, returns a summary of the deployment status of the resources in the environment in the metadata, describing how many resources are in

each state as well as the total number of resources. The summary does not take into account the current filters or paging parameters. Orphaned resources are not included in the summary

Returns

A list of all matching released resources

Raises

- `NotFound` – This exception is raised when the referenced environment is not found
- `BadRequest` – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`resource_logs`**(*tid: UUID*, *rid: ResourceIdStr*, *limit: int | None = None*, *start: datetime | None = None*, *end: datetime | None = None*, *filter: dict[str, list[str]] | None = None*, *sort: str = 'timestamp.desc'*) → list[ResourceLog]

Get the logs of a specific resource.

Parameters

- **`tid`** – The id of the environment this resource belongs to
- **`rid`** – The id of the resource
- **`limit`** – Limit the number of instances that are returned
- **`start`** – The lower limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`end`** – The upper limit for the order by column (exclusive). Only one of 'start' and 'end' should be specified at the same time.
- **`filter`** – Filter the list of returned logs. Filters should be specified with the syntax *?filter.<filter_key>=value*, for example *?filter.minimal_log_level=INFO*. It's also possible to provide multiple values for the same filter, in this case resources are returned, if they match any of these filter values.

  For example: *?filter.action=pull&filter.action=deploy* returns logs with either of the actions pull or deploy. Multiple different filters narrow the results however (they are treated as an 'AND' operator). For example *filter.minimal_log_level=INFO&filter.action=deploy* returns logs with 'deploy' action, where the 'log_level' is at least 'INFO'.

  The following options are available:

  - action: filter by the action of the log
  - timestamp: return the logs matching the timestamp constraints. Valid constraints are of the form "<lt|le|gt|ge>:<x>". The expected format is YYYY-MM-DDTHH:mm:ss.ssssss, so an ISO-8601 datetime string, in UTC timezone.

    For example: *?filter.timestamp=ge:2021-08-18T09:21:30.568353&filter.timestamp=lt:2021-08-18T10:21:30.568353*. Multiple constraints can be specified, in which case only log messages that match all constraints will be returned.

  - message: filter by the content of the log messages. Partial matches are allowed. (case-insensitive)
  - minimal_log_level: filter by the log level of the log messages. The filter specifies the minimal level, so messages with either this level, or a higher severity level are going to be included in the result.

    For example, for *filter.minimal_log_level=INFO*, the log messages with level *INFO, WARNING, ERROR, CRITICAL* all match the query.

- **sort** – Return the results sorted according to the parameter value. It should follow the pattern *<attribute_to_sort_by>.<order>*, for example *timestamp.desc* (case insensitive). Only sorting by *timestamp* is supported. The following orders are supported: 'asc', 'desc'

Returns
A list of all matching resource logs

Raises

- *NotFound* – This exception is raised when the referenced environment is not found
- *BadRequest* – When the parameters used for filtering, sorting or paging are not valid

`inmanta.protocol.methods_v2.`**`resources_status`**(*tid: UUID*, *version: int*, *rids: list[ResourceIdStr]*) → dict[ResourceIdStr, ResourceState]

Get the deployment status for a batch of resource ids

Parameters

- **tid** – The id of the environment the resources belong to
- **version** – Version of the model to get the status for
- **rids** – List of resource ids to fetch the status for.

`inmanta.protocol.methods_v2.`**`resume_environment`**(*tid: UUID*) → None

Resume all orchestrator operations for an environment. Resumes normal environment operation and unpauses all agents that were active when the environment was halted.

Parameters
tid – The environment id

Raises
NotFound – The given environment doesn't exist.

`inmanta.protocol.methods_v2.`**`set_fact`**(*tid: UUID*, *name: str*, *source: ParameterSource*, *value: str*, *resource_id: str*, *metadata: dict[str, str] | None = None*, *recompile: bool = False*, *expires: bool | None = True*) → ReturnValue[Fact]

Set a fact on the server. If the fact is a tracked unknown, it will trigger a recompile on the server. Otherwise, if the value is changed and recompile is true, a recompile is also triggered.

Parameters

- **tid** – The id of the environment
- **name** – The name of the fact
- **source** – The source of the fact
- **value** – The value of the fact
- **resource_id** – The resource this fact belongs to
- **metadata** – Optional. Metadata about the fact
- **recompile** – Optional. Whether to trigger a recompile if the value of the fact changed.
- **expires** – Optional. If the fact should expire or not. By default, facts expire.

`inmanta.protocol.methods_v2.`**`set_parameter`**(*tid: UUID*, *name: str*, *source: ParameterSource*, *value: str*, *metadata: dict[str, str] | None = None*, *recompile: bool = False*) → ReturnValue[Parameter]

Set a parameter on the server. If the parameter is an tracked unknown, it will trigger a recompile on the server. Otherwise, if the value is changed and recompile is true, a recompile is also triggered.

Parameters

- **tid** – The id of the environment
- **name** – The name of the parameter
- **source** – The source of the parameter.
- **value** – The value of the parameter
- **metadata** – Optional. Metadata about the parameter
- **recompile** – Optional. Whether to trigger a recompile if the value of the parameter changed.

`inmanta.protocol.methods_v2.`**`set_password`**(*username: str*, *password: str*) → None

Change the password of a user

Parameters

- **username** – The username of the user
- **password** – The password of this new user. The password should be at least 8 characters long.

Raises

- *NotFound* – Raised when the user does not exist
- *BadRequest* – Raised when server authentication is not enabled

`inmanta.protocol.methods_v2.`**`update_agent_map`**(*agent_map: dict[str, str]*) → None

Notify an agent about the fact that the autostart_agent_map has been updated.

Parameters

agent_map – The content of the new autostart_agent_map

`inmanta.protocol.methods_v2.`**`update_notification`**(*tid: UUID*, *notification_id: UUID*, *read: bool | None = None*, *cleared: bool | None = None*) → Notification

Update a notification by setting its flags

Parameters

- **tid** – The id of the environment
- **notification_id** – The id of the notification to update
- **read** – Whether the notification has been read
- **cleared** – Whether the notification has been cleared

Returns

The updated notification

Raises

NotFound – When the referenced environment or notification is not found

`inmanta.protocol.methods_v2.`**`versioned_resource_details`**(*tid: UUID*, *version: int*, *rid: ResourceIdStr*) → VersionedResourceDetails

Parameters

- **tid** – The id of the environment
- **version** – The version number of the resource
- **rid** – The id of the resource

Returns

The details of a specific version of a resource

Raises

NotFound – This exception is raised when the referenced environment or resource is not found

11.5.17 Server

`class inmanta.server.extensions.`**`ApplicationContext`**

Bases: `object`

`class inmanta.server.bootloader.`**`InmantaBootloader`**(*configure_logging: bool = False*)

Bases: `object`

The inmanta bootloader is responsible for: - discovering extensions - loading extensions - loading core and extension slices - starting the server and its slices in the correct order

11.6 Inmanta Compile Data Reference

This page documents the compile data output when compiling with the *–export-compile-data* flag. The structure of this JSON is defined by *`inmanta.data.model.CompileData`* which inherits from `pydantic.BaseModel`. To produce the JSON representation of the object, *model.json()* is called. See the pydantic documentation for more information on how exactly a JSON is generated from a model.

`class inmanta.data.model.`**`CompileData`**(*, *errors: list[Error]*)

Bases: *`BaseModel`*

Top level structure of compiler data to be exported.

`errors: list[Error]`

All errors occurred while trying to compile.

`model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}`

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

`model_config: ClassVar[ConfigDict] = {'use_enum_values': True}`

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

`model_fields: ClassVar[dict[str, FieldInfo]] = {'errors': FieldInfo(annotation=list[Error], required=True)}`

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

`class inmanta.ast.export.`**`Error`**(*, *category: ErrorCategory = ErrorCategory.runtime*, *type: str*, *message: str*, *location: Location | None = None*, ***extra_data: Any*)

Bases: `BaseModel`

Error occurred while trying to compile.

`category:` *`ErrorCategory`*

Category of this error.

`location:` *`Location`* **`| None`**

Location where this error occurred.

`message: str`

Error message.

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {'extra': 'allow', 'validate_assignment': True}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'category': FieldInfo(annotation=ErrorCategory, required=False, default=<ErrorCategory.runtime: 'runtime_error'>), 'location': FieldInfo(annotation=Union[Location, NoneType], required=False, default=None), 'message': FieldInfo(annotation=str, required=True), 'type': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

type: str

Fully qualified name of the actual exception.

class inmanta.ast.export.**ErrorCategory**(*value*, *names=None*, ***, *module=None*, *qualname=None*, *type=None*, *start=1*, *boundary=None*)

Bases: `str`, `Enum`

Category of an error.

parser = 'parse_error'

Error occurred while parsing.

plugin = 'plugin_exception'

A plugin explicitly raised an *inmanta.plugins.PluginException*.

runtime = 'runtime_error'

Error occurred after parsing.

class inmanta.ast.export.**Location**(***, *uri: str*, *range:* Range)

Bases: `BaseModel`

Location in a file. Based on the LSP spec 3.15

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'range': FieldInfo(annotation=Range, required=True), 'uri': FieldInfo(annotation=str, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

range: *Range*

uri: str

class inmanta.ast.export.**Range**(**, start: Position, end: Position*)

Bases: `BaseModel`

Range in a file. Based on the LSP spec 3.15

end: *Position*

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'end': FieldInfo(annotation=Position, required=True), 'start': FieldInfo(annotation=Position, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

start: *Position*

class inmanta.ast.export.**Position**(**, line: int, character: int*)

Bases: `BaseModel`

Position in a file. Based on the LSP spec 3.15

character: int

line: int

model_computed_fields: ClassVar[dict[str, ComputedFieldInfo]] = {}

A dictionary of computed field names and their corresponding *ComputedFieldInfo* objects.

model_config: ClassVar[ConfigDict] = {}

Configuration for the model, should be a dictionary conforming to [*ConfigDict*][pydantic.config.ConfigDict].

model_fields: ClassVar[dict[str, FieldInfo]] = {'character': FieldInfo(annotation=int, required=True), 'line': FieldInfo(annotation=int, required=True)}

Metadata about the fields defined on the model, mapping of field names to [*FieldInfo*][pydantic.fields.FieldInfo].

This replaces *Model.__fields__* from Pydantic V1.

11.7 Inmanta modules

11.7.1 Module apt

- License: Apache 2.0
- Version: 0.4.25
- This module requires compiler version 2017.1 or higher
- Upstream project: https://github.com/inmanta/apt.git

Entities

entity `apt::Repository`

Parents: *`std::Entity`*

An apt repository

attribute string `name`

attribute string `base_url`

attribute string `release`

attribute string `repo`

attribute bool `trusted`=false

relation std::Host `host` [1]

other end: *`std::Host.repository [0:*]`*

The following implementations are defined for this entity:

- *`apt::simpleRepo`*

The following implements statements select implementations for this entity:

- *`apt::simpleRepo`* constraint `true`

Implementations

implementation `apt::simpleRepo`

Handlers

`class apt.`**`AptPackage`**

A Package handler that uses apt

- Handler name `apt`
- Handler for entity *`std::Package`*

11.7.2 Module aws

- License: Apache 2.0
- Version: 4.0.2
- This module requires compiler version 2017.2 or higher
- Upstream project: https://github.com/inmanta/aws.git

Typedefs

typedef `aws::direction`

- Base type `string`
- Type constraint `((self == 'ingress') or (self == 'egress'))`

typedef `aws::instance_tenancy`

- Base type `string`
- Type constraint `/^(default|dedicated|host)$/`

typedef `aws::protocol`

- Base type `string`
- Type constraint `(self in ['tcp', 'udp', 'icmp', 'sctp', 'all'])`

Entities

entity `aws::AWSResource`

Parents: *std::PurgeableResource*, *std::ManagedResource*

relation aws::Provider `provider` [1]

entity `aws::ELB`

Parents: *aws::AWSResource*

An ELB load balancer

attribute string `name`

attribute string `security_group`='default'

attribute std::port `listen_port`=80

attribute std::port `dest_port`=80

attribute string `protocol`='http'

relation aws::VirtualMachine `instances` [0:*]

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `aws::GroupRule`

Parents: *aws::SecurityRule*

relation aws::SecurityGroup `remote_group` [1]

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `aws::IPrule`

Parents: *aws::SecurityRule*

attribute std::ipv4_network `remote_prefix`

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity aws::InternetGateway

Parents: *aws::AWSResource*

An Internet gateway for use with a VPC.

attribute string name

relation aws::VPC vpc [0:1]

other end: *aws::VPC.internet_gateway [0:1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity aws::Provider

Parents: *std::Entity*

The configuration to access Amazon Web Services

attribute string name

attribute string region

attribute string availability_zone

attribute string? access_key=null

attribute string? secret_key=null

attribute bool auto_agent=true

The following implementations are defined for this entity:

- *aws::agentConfig*

The following implements statements select implementations for this entity:

- *std::none* constraint true
- *aws::agentConfig* constraint auto_agent

entity aws::Route

Parents: *aws::AWSResource*

A route entry in the main VPC routing table

attribute std::ipv4_network destination

The destination route

attribute std::ipv4_address nexthop

The private ip associated with a ENI in the VPC.

relation aws::VPC vpc [1]

other end: *aws::VPC.routes [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity aws::SecurityGroup

Parents: *aws::AWSResource*

attribute string description=''

attribute string name

attribute bool manage_all=true

attribute int `retries`=10

A security group can only be deleted when it is no longer in use. The API confirms the delete of a virtual machine for example, but it might still be in progress. This results in a failure to delete the security group. To speed up deployments, the handler can retry this number of times before skipping the resource.

attribute int `wait`=5

The number of seconds to wait between retries.

relation aws::SecurityRule `rules` [0:*]

other end: *aws::SecurityRule.group [1]*

relation aws::VPC `vpc` [1]

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `aws::SecurityRule`

Parents: *std::Entity*

A filter rule in the a security group

attribute aws::protocol `ip_protocol`

The type of ip protocol to allow. Currently this support tcp/udp/icmp/sctp or all

attribute std::port `port_min`=0

attribute std::port `port_max`=0

attribute std::port `port`=0

attribute aws::direction `direction`

relation aws::SecurityGroup `group` [1]

other end: *aws::SecurityGroup.rules [0:*]*

entity `aws::Subnet`

Parents: *aws::AWSResource*

A subnet in a vpc

attribute string `name`

The name of the subnet. Inmanta uses this name to idenfiy the subnet. It is set as the name tag on the subnet resource.

attribute string? `availability_zone`=null

The Availability Zone for the subnet.

attribute std::ipv4_network `cidr_block`

The IPv4 network range for the VPC, in CIDR notation. For example, 10.0.0.0/24.

attribute bool `map_public_ip_on_launch`=false

Specify true to indicate that network interfaces created in the specified subnet should be assigned a public IPv4 address. This includes a network interface that's created when launching an instance into the subnet (the instance therefore receives a public IPv4 address).

relation aws::VPC `vpc` [1]

The VPC the subnet is created in.

other end: *aws::VPC.subnets [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity aws::VMAttributes

Parents: *std::Entity*

attribute string flavor

attribute string image

attribute string user_data

attribute string? subnet_id=null

attribute bool source_dest_check=true

attribute bool ebs_optimized=false

attribute bool ignore_extra_volumes=false

attribute bool ignore_wrong_image=false

attribute int root_volume_size=16

attribute string root_volume_type='gp2'

entity aws::VPC

Parents: *aws::AWSResource*

A VPC on Amazon

attribute string name

The name of the VPC. Inmanta uses this name to idenfiy the vpc. It is set as the name tag on the vpc resource.

attribute std::ipv4_network cidr_block

The IPv4 network range for the VPC, in CIDR notation. For example, 10.0.0.0/16.

attribute aws::instance_tenancy instance_tenancy='default'

The tenancy options for instances launched into the VPC. For default , instances are launched with shared tenancy by default. You can launch instances with any tenancy into a shared tenancy VPC. For dedicated , instances are launched as dedicated tenancy instances by default. You can only launch instances with a tenancy of dedicated or host into a dedicated tenancy VPC.

attribute bool enableDnsHostnames=false

attribute bool enableDnsSupport=false

relation aws::Subnet subnets [0:*]

The VPC the subnet is created in.

other end: *aws::Subnet.vpc [1]*

relation aws::InternetGateway internet_gateway [0:1]

other end: *aws::InternetGateway.vpc [0:1]*

relation aws::Route routes [0:*]

other end: *aws::Route.vpc [1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity aws::VirtualMachine

Parents: *aws::VMAttributes*, *aws::AWSResource*

This entity represents a virtual machine that is hosted on an IaaS

attribute string `name`

attribute dict `tags`={}

relation ssh::Key `public_key` [1]

relation aws::Subnet `subnet` [0:1]

Boot the vm in this subnet. Either use this relation or provide a subnet id directly.

relation aws::SecurityGroup `security_groups` [0:*]

The security groups that apply to this vm. If no group is supplied the default security group will be applied by EC2

relation aws::Volume `volumes` [0:*]

other end: *aws::Volume.vm [0:1]*

The following implementations are defined for this entity:

- *aws::req*

The following implements statements select implementations for this entity:

- *aws::req* constraint `true`

entity `aws::Volume`

Parents: *aws::AWSResource*

attribute string `name`

attribute string `attachmentpoint`='/dev/sdb'

attribute string `availability_zone`

attribute bool `encrypted`=false

attribute int `size`=10

attribute string `volume_type`='gp2'

attribute dict `tags`={}

relation aws::VirtualMachine `vm` [0:1]

other end: *aws::VirtualMachine.volumes [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `aws::analytics::ElasticSearch`

Parents: *aws::AWSResource*

Amazon Elasticsearch Service (Amazon ES) is a managed service that makes it easy to create a domain and deploy, operate, and scale Elasticsearch clusters in the AWS Cloud.

attribute string `domain_name`

attribute string `elasticsearch_version`

attribute string `instance_type`

attribute number `instance_count`=1

attribute bool `dedicated_master_enabled`=false

attribute bool `zone_awareness_enabled`=false

attribute string `dedicated_master_type`=''

attribute number `dedicated_master_count`=1

attribute bool `ebs_enabled`=true

attribute string `volume_type`='gp2'

attribute number `volume_size`

attribute string `access_policies`

attribute number `automated_snapshot_start_hour`=0

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `aws::database::RDS`

Parents: *`aws::AWSResource`*

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the cloud.

attribute string `name`

attribute number `allocated_storage`=10

attribute string `flavor`='db.t2.small'

attribute string `engine`='mysql'

attribute string `engine_version`='5.7.17'

attribute string `master_user_name`='root'

attribute string `master_user_password`

attribute string `subnet_group`

attribute std::port `port`=3306

attribute bool `public`=false

attribute dict `tags`={}

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

Implementations

implementation `aws::agentConfig`

implementation `aws::req`

Plugins

`aws.`**`elbid`**`(`*name: 'string'*`)` → 'string'

`aws.`**`get_api_id`**`(`*provider: 'aws::Provider'*, *api_name: 'string'*`)` → 'string'

Resources

`class aws.`**`ELB`**

Amazon Elastic loadbalancer

- Resource for entity *`aws::ELB`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`aws.ELBHandler`*

`class aws.`**`InternetGateway`**

- Resource for entity *`aws::InternetGateway`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`aws.InternetGatewayHandler`*

`class aws.`**`Route`**

- Resource for entity *`aws::Route`*
- Id attribute `destination`
- Agent name `provider.name`
- Handlers *`aws.RouteHandler`*

`class aws.`**`SecurityGroup`**

A security group in an OpenStack tenant

- Resource for entity *`aws::SecurityGroup`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`aws.SecurityGroupHandler`*

`class aws.`**`Subnet`**

- Resource for entity *`aws::Subnet`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`aws.SubnetHandler`*

`class aws.`**`VPC`**

- Resource for entity *`aws::VPC`*
- Id attribute `name`
- Agent name `provider.name`

- Handlers *aws.VPCHandler*

class aws.**VirtualMachine**

- Resource for entity *aws::VirtualMachine*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *aws.VirtualMachineHandler*

class aws.**Volume**

- Resource for entity *aws::Volume*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *aws.VolumeHandler*

class aws.**ElasticSearch**

- Resource for entity *aws::analytics::ElasticSearch*
- Id attribute `domain_name`
- Agent name `provider.name`
- Handlers *aws.ElasticSearchHandler*

class aws.**RDS**

- Resource for entity *aws::database::RDS*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *aws.RDSHandler*

Handlers

class aws.**ELBHandler**

This class manages ELB instances on amazon ec2

- Handler name `ec2`
- Handler for entity *aws::ELB*

class aws.**VirtualMachineHandler**

- Handler name `ec2`
- Handler for entity *aws::VirtualMachine*

class aws.**VolumeHandler**

- Handler name `volume`
- Handler for entity *aws::Volume*

class aws.**ElasticSearchHandler**

- Handler name `elasticsearch`
- Handler for entity *aws::analytics::ElasticSearch*

class aws.**RDSHandler**

- Handler name `elasticsearch`
- Handler for entity *aws::database::RDS*

class aws.**VPCHandler**

- Handler name `ec2`
- Handler for entity *aws::VPC*

class aws.**RouteHandler**

- Handler name `ec2`
- Handler for entity *aws::Route*

class aws.**SubnetHandler**

- Handler name `ec2`
- Handler for entity *aws::Subnet*

class aws.**InternetGatewayHandler**

- Handler name `ec2`
- Handler for entity *aws::InternetGateway*

class aws.**SecurityGroupHandler**

- Handler name `ec2`
- Handler for entity *aws::SecurityGroup*

11.7.3 Module exec

- License: Apache 2.0
- Version: 1.1.21
- This module requires compiler version 2017.1 or higher
- Upstream project: https://github.com/inmanta/exec.git

Entities

entity exec::Run

Parents: *std::Resource*

Run a command with almost exact semantics as the exec type of puppet

The command is not executed in a shell! This means:

- shell operators like ;, |, > don't work
- variable substitution doesn't work: *echo $PATH* will literally print *$PATH*
- variable substitution doesn't work in environment variables either: setting *PATH* to *$PATH* will result in *command not found*

If want to run a command in a shell, use the plugin 'in_shell':

```
exec::Run(host=host, command=exec::in_shell(command))
```

If you want variable substitution on environment variables, use the export command in the shell:

```
exec::Run(host=host, command=exec::in_shell("export PATH=$PATH:/usr/local/bin; {
↪{command}}"))
```

attribute string command

The actual command to execute. The command should be almost always be idempotent.

attribute string creates=''

A file that the command creates, when the file already exists the command will not be executed. This helps to make simple commands idempotent

attribute string cwd=''

The directory from which to run the command. WARNING: Command is spawned in a subshell. This implies that the real path of cwd is used and not a possible symlinked path.

attribute dict environment={ }

Environment variables to set before the command is executed. A dictionary of variables can be passed in the form {"var": "value"}.

attribute string onlyif=''

Only execute the command if this command is true (returns 0)

attribute string path=''

The path to search the command in

attribute string reload=''

The command to execute when this run needs to reload. If empty the command itself will be executed again.

attribute bool reload_only=false

Only use this command to reload

attribute number[] returns=List()

A list of valid return codes, by default this is only 0

attribute number timeout=300

The maximum time the command should take. If the command takes longer, the deploy agent will try to end it.

attribute string unless=''

If this attribute is set, the command will only execute if the command in this attribute is not successful (returns not 0). If the command passed to this attribute does not exist, this is interpreted as a non-successful execution.

attribute bool skip_on_fail=false

Report this resource as skipped instead of failed.

relation std::Host host [1]

The following implementations are defined for this entity:

- *exec::execHost*

The following implements statements select implementations for this entity:

- *exec::execHost* constraint true

Implementations

implementation `exec::execHost`

Plugins

`exec.`**`in_shell`**`(`*command: 'string'*`)` → 'any'

Wrap the command such that it is executed in a shell

Resources

class `exec.`**`Run`**

This class represents a service on a system.

- Resource for entity *`exec::Run`*
- Id attribute `command`
- Agent name `host.name`
- Handlers *`exec.PosixRun`*

Handlers

class `exec.`**`PosixRun`**

A handler to execute commands on posix compatible systems. This is a very atypical resource as this executes a command. The check_resource method will determine based on the "reload_only", "creates", "unless" and "onlyif" attributes if the command will be executed.

- Handler name `posix`
- Handler for entity *`exec::Run`*

11.7.4 Module lsm

- License: Inmanta EULA
- Version: 2.29.2
- This module requires compiler version 2023.5 or higher
- Upstream project: https://github.com/inmanta/lsm.git

Typedefs

typedef `lsm::attribute_modifier`

- Base type `string`
- Type constraint `(self in ['r', 'rw', 'rw+'])`

typedef `lsm::attribute_set_opt`

- Base type `string`
- Type constraint `(self in ['candidate', 'active', 'rollback'])`

typedef `lsm::labels`

- Base type `string`
- Type constraint `((((self == 'success') or (self == 'info')) or (self == 'warning')) or (self == 'danger'))`

typedef `lsm::operation`

- Base type `string`
- Type constraint `/^(clear (candidate|active|rollback)|promote|rollback)$/`

Entities

entity `lsm::EmbeddedEntity`

Parents: *`std::Entity`*

An entity containing attributes that should be embedded into a ServiceEntity or another EmbeddedEntity.

attribute string[]? `__lsm_key_attributes`=null

list of attributes that uniquely identify the embedded entity. The embedded entity must have an index defined on these attribute.

attribute dict `__annotations`={ }

The annotations that should be associated with this ServiceEntity. The key-value pairs in this dictionary represent respectively the name and the value of the annotation.

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `lsm::InterServiceRelation`

Parents: *`std::Entity`*

An Entity used to create relations between different services. This Entity should be used as an annotation on a relationship definition and should only be used through the __service__ variable

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `lsm::LifecycleStateMachine`

Parents: *`std::Entity`*

The lifecycle statemachine definition.

attribute string `name`

The name of the lifecycle used for reporting on the lifecycle

attribute bool `render_graph`=false

When set to true the state machine is renderd to a dot graph for debug purposes.

relation lsm::State `initial_state` [1]

relation lsm::StateTransfer `transfers` [0:*]

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `lsm::LifecycleTransfer`

Parents: *`std::PurgeableResource`*

This resource reports the result of all events its receives to the LSM for handling transfers

attribute string `service_entity`

The service entity from the service catalog.

attribute string `instance_id`

The id of the service instance.

attribute number `next_version`

The next version of the instance in the LSM

attribute bool `purge_on_delete`=false

attribute string `agent`='internal'

relation std::Resource `resources` [0:*]

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `lsm::RelationAnnotations`

Parents: *std::Entity*

An entity that holds the annotations set on a relationship.

attribute dict `annotations`

Annotations that should be associated with a relationship.

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `lsm::RelationshipMetadata`

Parents: *std::Entity*

The metadata belonging to the relationship to an embedded entity.

attribute lsm::attribute_modifier `modifier`='rw'

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `lsm::ServiceBase`

Parents: *std::Entity*

A baseclass used for entities that refine a service instance to resources.

attribute bool `purge_resources`

Remove the resources associated with this service

relation lsm::ServiceBase `children` [0:*]

This relation connects a parent entity to its children entities. These children can be:

- entities that are refined from the parent entity.
- entities that are owned by the parent entity.
- embedded entities that are part of the parent entity.

This relation is required to collect all resources a service consists of.

other end: *lsm::ServiceBase.parent [0:1]*

relation lsm::ServiceBase `parent` [0:1]

This relation connects a parent entity to its children entities. These children can be:

- entities that are refined from the parent entity.

- entities that are owned by the parent entity.
- embedded entities that are part of the parent entity.

This relation is required to collect all resources a service consists of.

other end: *lsm::ServiceBase.children [0:*]*

relation std::Resource `resources` [0:*]

A list of resources that this entity refines to. This relation will also contain all resources from entities collected through child service base entities.

relation std::Resource `owned_resources` [0:*]

A sublist of ServiceBase.resources containing only those resources that are exclusively used by this service instance. This list is used when the lsm_partial_compile environment option is enabled to determine the resources for the resource set of this service.

The following implementations are defined for this entity:

- *lsm::serviceBase*

The following implements statements select implementations for this entity:

- *lsm::serviceBase* constraint `true`

entity `lsm::ServiceEntity`

Parents: *lsm::ServiceBase*

A high level service that become available in the service catalog of the LSM. Based on this definition a service entity is generated in the catalog and an API endpoint becomes available in the service inventory.

Attributes starting with '_' are excluded from the API specification. Attributes starting with '__' are treated as metadata for the ServiceEntity. The values are taken from the defaults.

Service attributes are defined by adding an attribute to this entity. Metadata is specified by adding attributes with default values that start with the name of the attribute, a double underscore and then the name of the metadata. For example, for the attribute service_id, service_id__modifier provides the modifier of the service_id attribute. The following metadata is available. - modifier: Defines when the attribute can be set. See the type definition for more information.

attribute std::uuid `instance_id`

A unique id allocated by the LSM for each service instance.

attribute dict `__annotations={}`

Annotations that should be associated with this ServiceEntity. The key-value pairs in this dictionary represent respectively the name and the value of the annotation.

relation lsm::State `current_state` [1]

A reference to the current state the state machine is in.

relation lsm::ServiceEntityBinding `entity_binding` [1]

The binding of the service entity to the LSM service

The following implementations are defined for this entity:

- *lsm::stateConfig*
- *lsm::setResourceSet*

The following implements statements select implementations for this entity:

- constraint `true`
- *lsm::stateConfig* constraint `true`
- *lsm::setResourceSet* constraint `true`

entity `lsm::ServiceEntityBinding`

Parents: *`std::Entity`*

Instances of this entity are used to bind entities to the service catalog

attribute string `service_entity`

The fully qualified named of the entity to register in the catalog

attribute string `service_entity_name`

The name of the service instance. This name is used to register the service in the service catalog and generate the REST API on the LSM

attribute string? `allocation_spec`=null

name of the allocation specification for this ServiceEntityBinding, check the product documentation for more information

attribute string? `service_identity`=null

name of the attribute to be used as a service identity

attribute string? `service_identity_display_name`=null

The display name of the service identity, to be used by the frontend

attribute string? `relation_to_owner`=null

the name of the InterServiceRelation to use to find the owning instance

attribute bool `strict_modifier_enforcement`=false

Boolean value to control if strict validation of embedded entities' attributes with respect to modifiers is on or off

relation lsm::LifecycleStateMachine `lifecycle` [1]

The statemachine that represents the lifecycle

relation lsm::ServiceEntityBinding `owned` [0:*]

The ServiceEntityBinding in whose ResourceSet the resources for this entity should be placed in case of partial compile

other end: *`lsm::ServiceEntityBinding.owner [0:1]`*

relation lsm::ServiceEntityBinding `owner` [0:1]

The ServiceEntityBinding in whose ResourceSet the resources for this entity should be placed in case of partial compile

other end: *`lsm::ServiceEntityBinding.owned [0:*]`*

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `lsm::ServiceEntityBindingV2`

Parents: *`lsm::ServiceEntityBinding`*

Version 2 of the ServiceEntityBinding entity. In contrast to the ServiceEntityBinding entity, this entity enabled *strict_modifier_enforcement* by default.

attribute bool `strict_modifier_enforcement`=true

The following implements statements select implementations for this entity:

- constraint `true`

entity `lsm::ServiceInstance`

Parents: *`std::PurgeableResource`*

Instances of this entity are instances of a service entity in the catalog

attribute string `service_entity`

The name of the service entity this instance belongs to

attribute std::uuid `instance_id`

ID of this instance

attribute dict `attributes`

The desired attributes of this instance. The attribute names and types are determined by the service entity

attribute string[] `skip_update_states`

The Lifecycle States where updating the service instance is not possible, and should be marked as skipped

attribute string[] `rejected_states`

A list of unrecoverable states. The ServiceInstance resource will end up in the failed state when the service instance is in one of these rejected_states. In a rejected state the resource can only be purged.

attribute bool `purge_on_delete`=false

attribute string `agent`='internal'

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `lsm::State`

Parents: *std::Entity*

A state in the state machine

attribute string `name`

The name of a state

attribute lsm::labels? `label`=null

A label that is used in the UI

attribute bool `export_resources`

Indicates whether the resources of a service instance should be exported when that service instance is in this state.

attribute lsm::attribute_set_opt? `validate_self`=null

The attribute set of a service instance that should be taken into account when that service instance is being validated while it is in this state of the its lifecycle.

attribute lsm::attribute_set_opt? `validate_others`=null

The attribute set of a service instance that should be taken into account when that service instance is involved in the validation of another service instance while it is in this state of the its lifecycle.

attribute bool `purge_resources`=false

Should the service purge resources

attribute bool `deleted`=false

The service is marked as deleted. This indicates the lifeycle has ended.

attribute dict `values`={}

A dictionary with values associated with this state. Use the api::fsmvalue plugin to access a value.

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `lsm::StateTransfer`

Parents: *`std::Entity`*

Configure an allowed state transfer. In a state machine transfers have to adhere to the following rules:

- if multiple transfers between two states, only one active transfer can have auto set to true

attribute bool `on_update`=false

Trigger this state transfer when the service is updated (API)

attribute bool `on_delete`=false

Trigger this state transfer when the service is deleted (API)

attribute bool `api_set_state`=false

Allow this state transfer using the API.

attribute bool `resource_based`=false

Do the state transfer using a StateParam resource. The state transfer either happens always, or when there are requires, when these requires are deployed.

attribute bool `auto`=false

The lifecycle manager executes this step automatically when it gets to the from state. Used often in combination with validate. Auto transfers can be configured to conditionally trigger via the config_name attribute.

attribute bool `validate`=false

Run a compiler validation step when following this state transfer.

attribute string? `description`=null

name of description to aid debugging

attribute string? `config_name`=null

For auto transfers only, an optional configuration name that determines whether this transfer is enabled or not.

attribute lsm::operation? `target_operation`=null

The operation to perform on the attribute sets when the transfer goes to the target state.

attribute lsm::operation? `error_operation`=null

The operation to perform on the attribute sets when the transfer goes to the error state.

relation lsm::State `source` [1]

The start of the transfer edge

relation lsm::State `target` [1]

The end of the transfer edge

relation lsm::State `error` [0:1]

The target state in cause of failure. There is an implicit error edge for each statetransfer to this error state.

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

Implementations

implementation `lsm::serviceBase`

implementation `lsm::setResourceSet`

Associate a resource set with this ServiceEntity.

implementation `lsm::stateConfig`

This implementation configures the state machine

Plugins

`lsm.`**`all`**(*binding: 'lsm::ServiceEntityBinding'*) → 'list'

Returns a list of records for the given binding.

Parameters

`binding` – The entity binding

Returns

A list of dict with all the defined records.

`lsm.`**`context_v2_unwrapper`**(*assignments: 'dict[]'*, *fallback_attribute: 'string'*, *track_deletes: 'bool' = False*) → 'dict[]'

This plugin can be used to wrap the instances coming out of lsm::all and place all allocated values in :param fallback_attribute: where they should go. The returned value is what has been given as input, except for the allocated values being set where they should. :param track_deletes: drop deleted embedded entities, even if they still exist in the fallback set.

This should be used together with ContextV2Wrapper allocator.

Each assignment is an attribute dict containing a fallback attribute assigned with allocated values as produced by the ContextV2 (one-level deep, keys are string representation of the paths, values are allocated values) and update the dict placing values where their key-path would reach them.

Parameters

- **`assignments`** – The list of service instance dictionaries as returned by lsm::all
- **`fallback_attributes`** – The attribute name at the root of the instance attributes that contains all the allocated values

e.g.:

```
context_v2_unwrapper(
    [
        {
            "environment": "8f7bf3a5-d655-4bcb-bbd4-6222407be999",
            "id": "f93acfad-7894-4a12-9770-b27cbdd85c74",
            "service_entity": "carrierEthernetEvc",
            "version": 4,
            "config": {},
            "state": "allocating",
            "attributes": {
                "allocated": {
                    "evcEndPoints[identifier=my-evc-ep-1].uni": {
                        "uniref": "inmanta:456-852-789",
                        "test_value": "test_value",
                    },
                    "evcEndPoints[identifier=my-evc-ep-2].uni": {
                        "uniref": "inmanta:123-852-456",
                        "test_value": "test_value",
```

(continues on next page)

(continued from previous page)

```
                    },
                },
                "another_key": "any value",
            },
            "candidate_attributes": {
                "allocated": {
                    "evcEndPoints[identifier=my-evc-ep-1].uni": {
                        "uniref": "inmanta:456-852-789",
                        "test_value": "test_value",
                    },
                    "evcEndPoints[identifier=my-evc-ep-2].uni": {
                        "uniref": "inmanta:123-852-456",
                        "test_value": "test_value",
                    },
                },
                "another_key": "any value",
            },
            "active_attributes": {},
            "rollback_attributes": {},
        }
    ],
    "allocated",
)
```

will return:

```
[
    {
        "environment": "8f7bf3a5-d655-4bcb-bbd4-6222407be999",
        "id": "f93acfad-7894-4a12-9770-b27cbdd85c74",
        "service_entity": "carrierEthernetEvc",
        "version": 4,
        "config": {},
        "state": "allocating",
        "attributes": {
            "allocated": {
                "evcEndPoints[identifier=my-evc-ep-1].uni": {
                    "uniref": "inmanta:456-852-789",
                    "test_value": "test_value",
                },
                "evcEndPoints[identifier=my-evc-ep-2].uni": {
                    "uniref": "inmanta:123-852-456",
                    "test_value": "test_value",
                },
            },
            "evcEndPoints": [
                {
                    "identifier": "my-evc-ep-1",
                    "uni": {
                        "uniref": "inmanta:456-852-789",
                        "test_value": "test_value",
                    },
                },
                {
                    "identifier": "my-evc-ep-2",
                    "uni": {
```

(continues on next page)

(continued from previous page)

```
                        "uniref": "inmanta:123-852-456",
                        "test_value": "test_value",
                    },
                },
            ],
            "another_key": "any value",
        },
        "candidate_attributes": {
            "allocated": {
                "evcEndPoints[identifier=my-evc-ep-1].uni": {
                    "uniref": "inmanta:456-852-789",
                    "test_value": "test_value",
                },
                "evcEndPoints[identifier=my-evc-ep-2].uni": {
                    "uniref": "inmanta:123-852-456",
                    "test_value": "test_value",
                },
            },
            "evcEndPoints": [
                {
                    "identifier": "my-evc-ep-1",
                    "uni": {
                        "uniref": "inmanta:456-852-789",
                        "test_value": "test_value",
                    },
                },
                {
                    "identifier": "my-evc-ep-2",
                    "uni": {
                        "uniref": "inmanta:123-852-456",
                        "test_value": "test_value",
                    },
                },
            ],
            "another_key": "any value",
        },
        "active_attributes": {},
        "rollback_attributes": {},
    }

]
```

`lsm.`**`current_state`**(*fsm: 'lsm::ServiceEntity'*) → 'dict'

Returns the current state from the lifecycle and the next version of the instance

`lsm.`**`format`**(*__string: 'string'*, *args: 'list'*, *kwargs: 'dict'*) → 'string'

Format a string using python string formatter, and accepting statements which native inmanta f-string doesn't support (such as accessing dict values)

Parameters

- **`__string`** – The string to apply formatting to
- **`args`** – The positional arguments to feed into the *str.format* method
- **`kwargs`** – The keyword arguments to feed into the *str.format* method

`lsm.`**`fsm_to_dot`**(*config: 'lsm::LifecycleStateMachine'*) → 'string'

Generate a dot representation of the state machine lsm.**has_current_state**(*service_instance: 'lsm::ServiceEntity'*, *state_name: 'string'*) → 'bool'

Check whether the given service instance is currently in the given state of its lifecycle.

Parameters

- **service_instance** – The ServiceEntity object.
- **state_name** – The name of the lifecycle state

lsm.**is_validating**(*instance_id: 'string'*) → 'bool'

Return true if the current compile is a validating compile and the instance being validated has the given id.

Parameters
instance_id – The id of the instance we want to check for validation.

lsm.**render_dot**(*fsm: 'lsm::LifecycleStateMachine'*)

Render a dot graph in the current directory

lsm.**update_read_only_attribute**(*service: 'lsm::ServiceEntity'*, *attribute_path: 'string'*, *, *value: 'any'*) → 'any'

Update the value of a read-only (candidate) attribute in the service, in any compile, at any time. The value will first be compared to the previous set value, and only be sent to the server if it is different.

Parameters

- **service** – The service on which we want to set the value
- **attribute_path** – The path towards the service attribute
- **value** – The new value that we want to make sure is currently written in the service.

lsm.**validate_service_index**(*binding: 'lsm::ServiceEntityBinding'*, *attributes: 'dict'*, *ignored_states: 'string[]' = []*) → 'bool'

Validate that amongst all the services of the given binding that should be taken into account in this compile, only one of them has the provided set of attributes names and values.

Parameters

- **binding** – The binding for which we want to check the services.
- **attributes** – The attributes that we are looking for, the keys should be the attributes names, and the values the expected values.

lsm::allocators.allocate_value(service: 'lsm::ServiceEntity', attribute_path: 'string', *, value: 'any') -> 'any'

This simple allocator allows to store a value that we already know in the model in the service (mostly to make it more visible).

Parameters
value – The value to save in the service attributes.

lsm::allocators.combine_used_values(used_list: 'list') -> 'any'

lsm::allocators.get_first_free_integer(service: 'lsm::ServiceEntity', attribute_path: 'string', *, range_start: 'int', range_end: 'int', used_values: 'any') -> 'int'

Get the first free integer in the given range.

Check all services of the same type as the provided service for the value at attribute_path. Services can be filtered out by using the filters parameter. The filters parameters take as key the path to an attribute in the service and as value the attribute value that the service is expected at this path. If the value is no match, the service is ignored.

Parameters

- **range_start** – The lowest value that can be allocated.

- `range_end` – The highest value that can be allocated.
- `used_values` – A UsedValues object, as defined in inmanta_plugins.lsm.allocators.

`lsm::allocators.get_service_used_values(service: 'lsm::ServiceEntity', attributes_path: 'string', *, attribute_sets: 'string[]?' = None, attribute_sets_matching: 'dict?' = None) -> 'any'`

Get all the values used by this service at the given attribute path. The path can be a wild dict path, making use of wild cards to select multiple values within this service.

The values are checked in each resource set of the service. Some attributes sets can be excluded by providing a value to the *attribute_sets* param. When this is provided, we will only consider the attributes sets which are provided in the parameter.

The value returned by this plugin can not be used within the model. It can only be used as input for a V3 allocator.

Parameters

- `service` – The service in which we should look for the values.
- `attributes_path` – The wild dict path that points to the attributes we want to get the values of.
- `attribute_sets` – A list of attribute sets to consider when looking for a value. By default, check all attribute sets.
- `attribute_sets_matching` – A dict taking as key dict_path expressions, and as values literals. The dict will then be used to filter the instances in which we should look for the usage of values. Any attribute set for which any of the value at the given path doesn't match the value in this dict, will be left out.

`lsm::allocators.get_used_values(binding: 'lsm::ServiceEntityBindingV2', attributes_path: 'string', *, states: 'string[]?' = None, attribute_sets: 'string[]?' = None, attribute_sets_matching: 'dict?' = None) -> 'any'`

Get all the values used by the services of this binding at the given attribute path. The path can be a wild dict path, making use of wild cards to select multiple values within one service.

The values are checked in each resource set of each service in the inventory. Some attributes sets can be excluded by providing a value to the *attribute_sets* param. Some states can be excluded by providing a value to the *states* param. When those parameters are provided, we will only consider the attributes sets or states which are provided in the parameter.

The value returned by this plugin can not be used within the model. It can only be used as input for a V3 allocator.

Parameters

- `binding` – The binding defining the service in the catalog.
- `attributes_path` – The wild dict path that points to the attributes we want to get the values of.
- `attribute_sets` – A list of attribute sets to consider when looking for a value. By default, check all attribute sets.
- `attribute_sets_matching` – A dict taking as key dict_path expressions, and as values literals. The dict will then be used to filter the instances in which we should look for the usage of values. Any attribute set for which any of the value at the given path doesn't match the value in this dict, will be left out.
- `states` – A list of states to consider when looking for a value. By default, check all states. Any service that is not in one of the provided states is skipped.

`lsm::allocators.reallocate_value(service: 'lsm::ServiceEntity', attribute_path: 'string', previous_value: 'any' = None, *, value: 'any') -> 'any'`

This simple allocator allows to store a value that we already know in the model in the service (mostly to make it more visible). The difference with allocate_value is that this one will be called for any validation compile our service is in.

Parameters
`value` – The value to save in the service attributes.

Resources

class `lsm.`**`LifecycleTransferResource`**

A resource that collects deploy events and send them to the lsm

- Resource for entity *`lsm::LifecycleTransfer`*
- Id attribute `instance_id`
- Agent name `agent`
- Handlers *`lsm.LifecycleTransferHandler`*

class `lsm.`**`ServiceInstanceResource`**

- Resource for entity *`lsm::ServiceInstance`*
- Id attribute `instance_id`
- Agent name `agent`
- Handlers *`lsm.ServiceInstanceHandler`*

Handlers

class `lsm.`**`LifecycleTransferHandler`**

A handler that collects all resource statuses from resources that are part of a service instance. The deploy() method is used to determine whether the resources of a service instance are deployed successfully or not.

- Handler name `local_state`
- Handler for entity *`lsm::LifecycleTransfer`*

class `lsm.`**`ServiceInstanceHandler`**

- Handler name `service_instance`
- Handler for entity *`lsm::ServiceInstance`*

11.7.5 Module openstack

- License: Apache 2.0
- Version: 4.0.4
- This module requires compiler version 2024.1 or higher
- Upstream project: https://github.com/inmanta/openstack.git

Typedefs

typedef `openstack::admin_state`

- Base type `string`
- Type constraint `((self == 'up') or (self == 'down'))`

typedef `openstack::container_format`

- Base type `string`
- Type constraint `(self in ['ami', 'ari', 'aki', 'bare', 'ovf', 'ova', 'docker'])`

typedef `openstack::direction`

- Base type `string`
- Type constraint `((self == 'ingress') or (self == 'egress'))`

typedef `openstack::disk_format`

- Base type `string`
- Type constraint `(self in ['ami', 'ari', 'aki', 'vhd', 'vhdx', 'vmdk', 'raw', 'qcow2', 'vdi', 'iso', 'ploop'])`

typedef `openstack::mac_addr`

- Base type `string`
- Type constraint `std::validate_type('pydantic.constr',self,{'regex': '^([0-9a-fA-F]{2})(:[0-9a-fA-F]{2}){5}|$', 'strict':  True})`

typedef `openstack::protocol`

- Base type `string`
- Type constraint `(self in ['tcp', 'udp', 'icmp', 'sctp', 'all'])`

typedef `openstack::visibility`

- Base type `string`
- Type constraint `(self in ['public', 'private'])`

Entities

entity `openstack::AddressPair`

Parents: *`std::Entity`*

An address pair that is added to a host port

attribute std::ipv4_network `address`

The address range that is allowed on this port (network interface)

attribute openstack::mac_addr? `mac=null`

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `openstack::EndPoint`

Parents: *`openstack::OpenStackResource`*

attribute string `region`

attribute string `internal_url`

attribute string public_url

attribute string admin_url

attribute string service_id

relation openstack::Service service [1]

other end: *openstack::Service.endpoint [0:1]*

relation openstack::Provider provider [1]

other end: *openstack::Provider.endpoints [0:*]*

The following implementations are defined for this entity:

- *openstack::endPoint*

The following implements statements select implementations for this entity:

- *openstack::endPoint*, *openstack::providerRequire* constraint true

entity openstack::Flavor

Parents: *openstack::OpenStackResource*

A machine flavor for OpenStack VMs

attribute string name

Descriptive name of the flavor. While OpenStack does not consider the name unique, this module does.

attribute int ram

Memory in MB for the flavor

attribute int vcpus

Number of VCPUs for the flavor

attribute int disk

Size of local disk in GB

attribute string? flavor_id=null

OpenStack unique ID. You can use the reserved value "auto" to have Nova generate a UUID for the flavor in cases where you cannot simply pass null.

attribute int ephemeral=0

Ephemeral disk size in GB

attribute int swap=0

Swap space in MB

attribute float rxtx_factor=1.0

RX/TX factor

attribute bool is_public=true

Whether the flavor is publicly visible

attribute dict extra_specs={ }

Set extra specs on a flavor. See https://docs.openstack.org/nova/rocky/admin/flavors.html

relation openstack::Provider provider [1]

other end: *openstack::Provider.flavors [0:*]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint true

entity openstack::FloatingIP

Parents: *openstack::OpenStackResource*

attribute string `name`

attribute std::ipv4_address `address`

attribute bool `force_ip`=false

relation openstack::Project `project` [1]

other end: *openstack::Project.floating_ips [0:*]*

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.floating_ips [0:*]*

relation openstack::Network `external_network` [1]

other end: *openstack::Network.floating_ips [0:*]*

relation openstack::HostPort `port` [1]

other end: *openstack::HostPort.floating_ips [0:*]*

The following implementations are defined for this entity:

- *openstack::fipName*
- *openstack::fipAddr*

The following implements statements select implementations for this entity:

- *openstack::fipName*, *openstack::providerRequire* constraint `true`
- *openstack::fipAddr* constraint `(not force_ip)`

entity `openstack::GroupRule`

Parents: *openstack::SecurityRule*

relation openstack::SecurityGroup `remote_group` [1]

other end: *openstack::SecurityGroup.remote_group_rules [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `openstack::Host`

Parents: *std::Host*, *openstack::VMAttributes*

attribute bool `purged`=false

Set whether this Host should exist or not.

attribute bool `purge_on_delete`=false

Purge this Host when it is deleted from the configuration model.

relation openstack::VirtualMachine `vm` [1]

other end: *openstack::VirtualMachine.host [0:1]*

relation openstack::Subnet `subnet` [0:1]

relation ssh::Key `key_pair` [1]

relation openstack::Project `project` [1]

relation openstack::Provider `provider` [1]

relation openstack::SecurityGroup `security_groups` [0:*]

The following implementations are defined for this entity:

- *openstack::eth0Port*
- *openstack::openstackVM*

The following implements statements select implementations for this entity:

- *openstack::eth0Port* constraint `subnet is defined`
- *std::hostDefaults*, *openstack::openstackVM* constraint `true`

entity `openstack::HostPort`

Parents: *openstack::Port*

A port attached to a VM

attribute string `name`

The name of the host port.

attribute bool `portsecurity`=true

Enable or disable port security (security groups and spoofing filters)

attribute bool `dhcp`=true

Enable dhcp for this port or not for this port

attribute int `port_index`=0

The index of the port. This determines the order of the interfaces on the virtual machine. 0 means no specific order.

attribute int `retries`=20

A hostport can only be attached to a VM when it is in an active state. The handler will skip this port when the VM is not ready. To speed up deployments, the handler can retry this number of times before skipping the resource.

attribute int `wait`=5

The number of seconds to wait between retries.

relation openstack::Subnet `subnet` [1]

other end: *openstack::Subnet.host_ports [0:*]*

relation openstack::VirtualMachine `vm` [1]

other end: *openstack::VirtualMachine.ports [0:*]*

relation openstack::FloatingIP `floating_ips` [0:*]

other end: *openstack::FloatingIP.port [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity `openstack::IPrule`

Parents: *openstack::SecurityRule*

attribute std::ipv4_network `remote_prefix`

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `openstack::Image`

Parents: *openstack::OpenStackResource*

A machine image for OpenStack VMs

attribute string `name`

Name for the flavor. Inmanta treats image names as unique per provider.

attribute string `uri`

a link to the download location of the image.

attribute openstack::container_format? `container_format`='bare'

Must be one of [null, ami, ari, aki, bare, ovf, ova, docker].

attribute openstack::disk_format? `disk_format`='qcow2'

Must be one of [null, ami, ari, aki, vhd, vhdx, vmdk, raw, qcow2, vdi, iso, ploop].

attribute std::uuid? `image_id`=null

uuid to identify the image. Auto set by OpenStack if not set.

attribute openstack::visibility `visibility`='public'

Whether the image is visible across all projects. Can either be public or private. Shared and community are currently not implemented.

attribute bool `protected`=false

Whether the image can be deleted or not. Inmanta will never delete protected images.

attribute dict `metadata`={ }

Various metadata passed as a dict.

attribute bool `skip_on_deploy`=true

When set, inmanta will not wait for the image to be deployed and mark it as skipped.

attribute bool `purge_on_delete`=false

When set to true, the image will be removed when no longer present in the model.

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.images [0:*]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity `openstack::Network`

Parents: *openstack::OpenStackResource*

A neutron network owned by a project

attribute string `name`

attribute bool `external`=false

attribute string `physical_network`=''

attribute string `network_type`=''

attribute int `segmentation_id`=0

attribute bool `shared`=false

attribute bool? `vlan_transparent`=null

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.networks [0:*]*

relation openstack::Project `project` [1]

other end: *openstack::Project.networks [0:*]*

relation openstack::Subnet `subnets` [0:*]

other end: *openstack::Subnet.network [1]*

relation openstack::Router `routers` [0:*]

other end: *openstack::Router.ext_gateway [0:1]*

relation openstack::FloatingIP `floating_ips` [0:*]

other end: *openstack::FloatingIP.external_network [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity `openstack::OpenStackResource`

Parents: *std::PurgeableResource*, *std::ManagedResource*

Base class for all openstack resources

attribute bool `send_event`=true

Forced to default true. This means that all resources that subscribe to this resource will run their process events / reload.

The following implementations are defined for this entity:

- *openstack::providerRequire*

entity `openstack::Port`

Parents: *openstack::OpenStackResource*

A port on a network

attribute std::ipv4_address `address`

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.ports [0:*]*

relation openstack::Project `project` [1]

other end: *openstack::Project.ports [0:*]*

relation openstack::AddressPair `allowed_address_pairs` [0:*]

entity `openstack::Project`

Parents: *openstack::OpenStackResource*

A project / tenant in openstack

attribute string `name`

attribute bool `enabled`=true

attribute string `description`=''

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.projects [0:*]*

relation openstack::Role `roles` [0:*]

Each user can have multiple roles

other end: *openstack::Role.project [1]*

relation openstack::Network `networks` [0:*]

other end: *openstack::Network.project [1]*

relation openstack::Port `ports` [0:*]

other end: *openstack::Port.project [1]*

relation openstack::Subnet `subnets` [0:*]

other end: *openstack::Subnet.project [1]*

relation openstack::Router `routers` [0:*]

other end: *openstack::Router.project [1]*

relation openstack::SecurityGroup `security_groups` [0:*]

other end: *openstack::SecurityGroup.project [1]*

relation openstack::FloatingIP `floating_ips` [0:*]

other end: *openstack::FloatingIP.project [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity `openstack::Provider`

Parents: *std::Entity*

The configuration for accessing an Openstack based IaaS

attribute string `name`

attribute string `connection_url`

attribute bool `verify_cert`=true

Indicates whether the SSL/TLS certificate should be verified.

attribute string `username`

attribute string `password`

attribute string `tenant`

attribute string `token`=''

attribute string `admin_url`=''

attribute bool `auto_agent`=true

relation openstack::Project `projects` [0:*]

other end: *openstack::Project.provider [1]*

relation openstack::User `users` [0:*]

other end: *openstack::User.provider [1]*

relation openstack::Role `roles` [0:*]

other end: *openstack::Role.provider [1]*

relation openstack::Service `services` [0:*]

other end: *openstack::Service.provider [1]*

relation openstack::EndPoint `endpoints` [0:*]

other end: *openstack::EndPoint.provider [1]*

relation openstack::Network `networks` [0:*]

other end: *openstack::Network.provider [1]*

relation openstack::Port `ports` [0:*]

other end: *openstack::Port.provider [1]*

relation openstack::Subnet `subnets` [0:*]

other end: *openstack::Subnet.provider [1]*

relation openstack::Router `routers` [0:*]

other end: *openstack::Router.provider [1]*

relation openstack::SecurityGroup `security_groups` [0:*]

other end: *openstack::SecurityGroup.provider [1]*

relation openstack::FloatingIP `floating_ips` [0:*]

other end: *openstack::FloatingIP.provider [1]*

relation openstack::VirtualMachine `virtual_machines` [0:*]

other end: *openstack::VirtualMachine.provider [1]*

relation openstack::Flavor `flavors` [0:*]

other end: *openstack::Flavor.provider [1]*

relation openstack::Image `images` [0:*]

other end: *openstack::Image.provider [1]*

The following implementations are defined for this entity:

- *openstack::agentConfig*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`
- *openstack::agentConfig* constraint `auto_agent`

entity `openstack::Role`

Parents: *openstack::OpenStackResource*

A role in openstack. A role defines membership of a user in a project. This entity is used to connect users to projects. With this, it implicitly defines the role.

attribute string `role_id`

attribute string `role`

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.roles [0:*]*

relation openstack::Project `project` [1]

Each user can have multiple roles

other end: *openstack::Project.roles [0:*]*

relation openstack::User `user` [1]

other end: *openstack::User.roles [0:*]*

The following implementations are defined for this entity:

- *openstack::roleImpl*

The following implements statements select implementations for this entity:

- *openstack::roleImpl*, *openstack::providerRequire* constraint `true`

entity `openstack::Route`

Parents: *std::Entity*

A routing rule to add

attribute std::ipv4_network `destination`

attribute std::ipv4_address `nexthop`

relation openstack::Router `router` [0:1]

other end: *openstack::Router.routes [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `openstack::Router`

Parents: *`openstack::OpenStackResource`*

A router

attribute openstack::admin_state `admin_state`='up'

attribute string `name`

attribute bool `ha`=false

attribute bool `distributed`=false

relation openstack::Provider `provider` [1]

other end: *`openstack::Provider.routers [0:*]`*

relation openstack::Project `project` [1]

other end: *`openstack::Project.routers [0:*]`*

relation openstack::RouterPort `ports` [0:*]

other end: *`openstack::RouterPort.router [1]`*

relation openstack::Subnet `subnets` [0:*]

other end: *`openstack::Subnet.router [0:1]`*

relation openstack::Network `ext_gateway` [0:1]

other end: *`openstack::Network.routers [0:*]`*

relation openstack::Route `routes` [0:*]

other end: *`openstack::Route.router [0:1]`*

The following implements statements select implementations for this entity:

- *`openstack::providerRequire`* constraint `true`

entity `openstack::RouterPort`

Parents: *`openstack::Port`*

A port attached to a router

attribute string `name`

relation openstack::Subnet `subnet` [1]

other end: *`openstack::Subnet.routers [0:*]`*

relation openstack::Router `router` [1]

other end: *`openstack::Router.ports [0:*]`*

The following implements statements select implementations for this entity:

- *`openstack::providerRequire`* constraint `true`

entity `openstack::SecurityGroup`

Parents: *`openstack::OpenStackResource`*

attribute string `description`=''

attribute string `name`

attribute bool `manage_all`=true

attribute int `retries`=10

A security group can only be deleted when it is no longer in use. The API confirms the delete of a virtual machine for example, but it might still be in progress. This results in a failure to delete the security group. To speed up deployments, the handler can retry this number of times before skipping the resource.

attribute int `wait=5`

The number of seconds to wait between retries.

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.security_groups [0:*]*

relation openstack::Project `project` [1]

other end: *openstack::Project.security_groups [0:*]*

relation openstack::VirtualMachine `virtual_machines` [0:*]

other end: *openstack::VirtualMachine.security_groups [0:*]*

relation openstack::GroupRule `remote_group_rules` [0:*]

other end: *openstack::GroupRule.remote_group [1]*

relation openstack::SecurityRule `rules` [0:*]

other end: *openstack::SecurityRule.group [1]*

The following implementations are defined for this entity:

- *openstack::sg*

The following implements statements select implementations for this entity:

- *openstack::sg*, *openstack::providerRequire* constraint `true`

entity `openstack::SecurityRule`

Parents: *std::Entity*

A filter rule in the a security group

attribute openstack::protocol `ip_protocol`

The type of ip protocol to allow. Currently this support tcp/udp/icmp/sctp or all

attribute std::port `port_min=0`

attribute std::port `port_max=0`

attribute std::port `port=0`

attribute openstack::direction `direction`

relation openstack::SecurityGroup `group` [1]

other end: *openstack::SecurityGroup.rules [0:*]*

entity `openstack::Service`

Parents: *openstack::OpenStackResource*

attribute string `name`

attribute string `type`

attribute string `description`

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.services [0:*]*

relation openstack::EndPoint `endpoint` [0:1]

other end: *openstack::EndPoint.service [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity openstack::Subnet

Parents: *openstack::OpenStackResource*

A neutron network subnet

attribute std::ipv4_network network_address

attribute bool dhcp

attribute string name

attribute string allocation_start=''

attribute string allocation_end=''

attribute std::ipv4_address[] dns_servers=List()

attribute std::ipv4_address? gateway_ip=null

The gateway IP to set on this subnet. If set to null, the first IP in the subnet will be used as the gateway_ip. Example: 192.168.0.1 will be used for the network 192.168.0.0/24.

attribute bool disable_gateway_ip=false

When set to true, no gateway IP will be set for the subnet. As such, the gateway_ip parameter will be ignored.

relation openstack::RouterPort routers [0:*]

other end: *openstack::RouterPort.subnet [1]*

relation openstack::HostPort host_ports [0:*]

other end: *openstack::HostPort.subnet [1]*

relation openstack::Provider provider [1]

other end: *openstack::Provider.subnets [0:*]*

relation openstack::Project project [1]

other end: *openstack::Project.subnets [0:*]*

relation openstack::Network network [1]

other end: *openstack::Network.subnets [0:*]*

relation openstack::Router router [0:1]

other end: *openstack::Router.subnets [0:*]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint true

entity openstack::User

Parents: *openstack::OpenStackResource*

A user in openstack. A handler for this entity type is loaded by agents.

attribute string name

The name of the user. The name of the user has to be unique on a specific IaaS. The handler will use this name to query for the exact user and its ID.

attribute string email

The email address of the user to use.

attribute bool enabled=true

Enable or disable this user

attribute string password=''

The password for this user. The handler will always reset back to this password. The handler will ignore this attribute when an empty string is set.

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.users [0:*]*

relation openstack::Role `roles` [0:*]

other end: *openstack::Role.user [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

entity `openstack::VMAttributes`

Parents: *std::Entity*

Entity with vm attributes that can be used for a virtual machine and a host

attribute string `flavor`

The name of the flavor

attribute string `image`

The uuid of the image

attribute string `user_data`

The user_data script to pass

attribute dict `metadata`={}

A dict of metadata items

attribute dict `personality`={}

A dict of files (personality)

attribute bool `config_drive`=false

Attach a configuration drive to the vm

entity `openstack::VirtualMachine`

Parents: *openstack::OpenStackResource*, *openstack::VMAttributes*

attribute string `name`

relation openstack::HostPort `ports` [0:*]

other end: *openstack::HostPort.vm [1]*

relation openstack::SecurityGroup `security_groups` [0:*]

other end: *openstack::SecurityGroup.virtual_machines [0:*]*

relation openstack::HostPort `eth0_port` [1]

relation ssh::Key `key_pair` [1]

relation openstack::Project `project` [1]

relation openstack::Provider `provider` [1]

other end: *openstack::Provider.virtual_machines [0:*]*

relation openstack::Host `host` [0:1]

other end: *openstack::Host.vm [1]*

The following implements statements select implementations for this entity:

- *openstack::providerRequire* constraint `true`

Implementations

implementation openstack::agentConfig

implementation openstack::endPoint

implementation openstack::eth0Port

implementation openstack::fipAddr

implementation openstack::fipName

implementation openstack::openstackVM

implementation openstack::providerRequire

implementation openstack::roleImpl

implementation openstack::sg

Plugins

openstack.**find_flavor**(*provider: 'openstack::Provider'*, *vcpus: 'number'*, *ram: 'number'*, *pinned: 'bool' = False*) → 'string'

Find the flavor that matches the closest to the resources requested.

Parameters

- **vcpus** – The number of virtual cpus in the flavor
- **ram** – The amount of ram in gigabyte
- **pinned** – Wether the CPUs need to be pinned (#vcpu == #pcpu)

openstack.**find_image**(*provider: 'openstack::Provider'*, *os: 'std::OS'*, *name: 'string' = None*) → 'string'

Search for an image that matches the given operating system. This plugin uses the os_distro and os_version tags of an image and the name and version attributes of the OS parameter.

If multiple images match, the most recent image is returned.

Parameters

- **provider** – The provider to query for an image
- **os** – The operating system and version (using os_distro and os_version metadata)
- **name** – An optional string that the image name should contain

Resources

class openstack.**EndPoint**

An endpoint for a service

- Resource for entity *openstack::EndPoint*
- Id attribute service_id
- Agent name provider.name
- Handlers *openstack.EndpointHandler*

class openstack.**Flavor**

A flavor is an available hardware configuration for a server.

- Resource for entity *openstack::Flavor*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.FlavorHandler*

class openstack.**FloatingIP**

A floating ip

- Resource for entity *openstack::FloatingIP*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.FloatingIPHandler*

class openstack.**HostPort**

A port in a router

- Resource for entity *openstack::HostPort*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.HostPortHandler*

class openstack.**Image**

- Resource for entity *openstack::Image*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.ImageHandler*

class openstack.**Network**

This class represents a network in neutron

- Resource for entity *openstack::Network*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.NetworkHandler*

class openstack.**Project**

This class represents a project in keystone

- Resource for entity *openstack::Project*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.ProjectHandler*

`class openstack.`**`Role`**

A role that adds a user to a project

- Resource for entity *`openstack::Role`*
- Id attribute `role_id`
- Agent name `provider.name`
- Handlers *`openstack.RoleHandler`*

`class openstack.`**`Router`**

This class represent a router in neutron

- Resource for entity *`openstack::Router`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`openstack.RouterHandler`*

`class openstack.`**`RouterPort`**

A port in a router

- Resource for entity *`openstack::RouterPort`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`openstack.RouterPortHandler`*

`class openstack.`**`SecurityGroup`**

A security group in an OpenStack tenant

- Resource for entity *`openstack::SecurityGroup`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`openstack.SecurityGroupHandler`*

`class openstack.`**`Service`**

A service for which endpoints can be registered

- Resource for entity *`openstack::Service`*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *`openstack.ServiceHandler`*

`class openstack.`**`Subnet`**

This class represent a subnet in neutron

- Resource for entity *`openstack::Subnet`*
- Id attribute `name`

- Agent name `provider.name`
- Handlers *openstack.SubnetHandler*

class openstack.**User**

A user in keystone

- Resource for entity *openstack::User*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.UserHandler*

class openstack.**VirtualMachine**

A virtual machine managed by a hypervisor or IaaS

- Resource for entity *openstack::VirtualMachine*
- Id attribute `name`
- Agent name `provider.name`
- Handlers *openstack.VirtualMachineHandler*

Handlers

class openstack.**FlavorHandler**

- Handler name `openstack`
- Handler for entity *openstack::Flavor*

class openstack.**ImageHandler**

- Handler name `openstack`
- Handler for entity *openstack::Image*

class openstack.**VirtualMachineHandler**

- Handler name `openstack`
- Handler for entity *openstack::VirtualMachine*

class openstack.**NetworkHandler**

- Handler name `openstack`
- Handler for entity *openstack::Network*

class openstack.**RouterHandler**

- Handler name `openstack`
- Handler for entity *openstack::Router*

class openstack.**SubnetHandler**

- Handler name `openstack`
- Handler for entity *openstack::Subnet*

class openstack.**RouterPortHandler**

- Handler name `openstack`
- Handler for entity *openstack::RouterPort*

class openstack.**HostPortHandler**

- Handler name `openstack`
- Handler for entity *openstack::HostPort*

class openstack.**SecurityGroupHandler**

- Handler name `openstack`
- Handler for entity *openstack::SecurityGroup*

class openstack.**FloatingIPHandler**

- Handler name `openstack`
- Handler for entity *openstack::FloatingIP*

class openstack.**ProjectHandler**

- Handler name `openstack`
- Handler for entity *openstack::Project*

class openstack.**UserHandler**

- Handler name `openstack`
- Handler for entity *openstack::User*

class openstack.**RoleHandler**

creates roles and user, project, role assocations

- Handler name `openstack`
- Handler for entity *openstack::Role*

class openstack.**ServiceHandler**

- Handler name `openstack`
- Handler for entity *openstack::Service*

class openstack.**EndpointHandler**

- Handler name `openstack`
- Handler for entity *openstack::EndPoint*

11.7.6 Module redhat

- License: Apache 2.0
- Version: 1.0.9
- Upstream project: https://github.com/inmanta/redhat.git

11.7.7 Module rest

- License: Apache 2.0
- Version: 0.2.21
- This module requires compiler version 2018.1 or higher
- Upstream project: https://github.com/inmanta/rest.git

Entities

entity `rest::RESTCall`

Parents: *`std::Resource`*

This resource executes a restcall during the execute phase of the handler

attribute string `url_id`

attribute string `url`

The url to call

attribute string `method`='GET'

The HTTP method to use

attribute dict `body`

The body of the the http call. By default this body is sent as a json body

attribute dict `headers`={}

Additional headers to pass to the server.

attribute bool `form_encoded`=false

Use form encoding for the body

attribute bool `ssl_verify`=true

Verify the ssl cert of the server

attribute string? `auth_user`=null

The user to authenticate with

attribute string? `auth_password`=null

The password to authenticate with

attribute number[] `return_codes`=List()

Returns code that indicate that the call was successfull

attribute string? `validate_return`=null

An JQ expression to validate the return result of the call. The result of this JQ expression evaluates to a python true or false.

attribute bool `skip_on_fail`=false

Report this resource as skipped instead of failed.

attribute string `agent`='internal'

The agent to initiate the REST call from

The following implementations are defined for this entity:

- *`rest::restCallID`*

The following implements statements select implementations for this entity:

- *`rest::restCallID`* constraint `true`

Implementations

implementation rest::restCallID

Resources

class rest.**RESTCall**

A Call to a rest endpoint

- Resource for entity *rest::RESTCall*
- Id attribute url_id
- Agent name agent
- Handlers *rest.RESTHandler*

Handlers

class rest.**RESTHandler**

- Handler name requests
- Handler for entity *rest::RESTCall*

11.7.8 Module ssh

- License: Apache 2.0
- Version: 1.0.1
- Upstream project: https://github.com/inmanta/ssh.git

Entities

entity ssh::Key

Parents: *std::Entity*

A public ssh-key used to access virtual machine

attribute string public_key

The actual public key that needs to be deployed

attribute string name

An identifier for the public key

attribute string command=''

The command that can be executed with this public key

attribute string options=''

SSH options associated with this public key

relation ssh::SSHUser ssh_users [0:*]

other end: *ssh::SSHUser.ssh_keys [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity `ssh::SSHUser`

Parents: *`std::Entity`*

An ssh users allows authorized keys to be installed

attribute string `home_dir`

attribute string `user`

attribute string `group`

relation ssh::Key `ssh_keys` [0:*]

other end: *`ssh::Key.ssh_users [0:*]`*

relation std::Host `host` [1]

The following implementations are defined for this entity:

- *`ssh::sshUser`*

The following implements statements select implementations for this entity:

- *`ssh::sshUser`* constraint `true`

Implementations

implementation `ssh::sshUser`

Plugins

`ssh.`**`get_private_key`**(*name: 'string'*) → 'string'

Create or return if it already exists a key with the given name. The private key is returned.

`ssh.`**`get_public_key`**(*name: 'string'*) → 'string'

See get_private_key

`ssh.`**`get_putty_key`**(*name: 'string'*) → 'string'

11.7.9 Module std

- License: Apache 2.0
- Version: 5.2.1
- This module requires compiler version 2023.6 or higher
- Upstream project: https://github.com/inmanta/std.git

Typedefs

typedef `std::alfanum`

- Base type `string`
- Type constraint `std::validate_type('pydantic.constr',self,{'regex': '^[a-zA-Z0-9]*$', 'strict': True})`

typedef `std::any_http_url`

- Base type `string`
- Type constraint `std::validate_type('pydantic.AnyHttpUrl',self)`

typedef `std::any_url`

- Base type `string`
- Type constraint `std::validate_type('pydantic.AnyUrl',self)`

typedef `std::ascii_word`

- Base type `string`
- Type constraint `std::validate_type('pydantic.constr',self,{'regex': '^[!-~]*$', 'strict': True})`

typedef `std::base64`

- Base type `string`
- Type constraint `std::is_base64_encoded(self)`

typedef `std::config_agent`

- Base type `string`
- Type constraint `(self != 'internal')`

typedef `std::date`

- Base type `string`
- Type constraint `std::validate_type('datetime.date',self)`

typedef `std::datetime`

- Base type `string`
- Type constraint `std::validate_type('datetime.datetime',self)`

typedef `std::email_str`

- Base type `string`
- Type constraint `std::validate_type('pydantic.EmailStr',self)`

typedef `std::hoststring`

- Base type `string`
- Type constraint `/^[A-Za-z0-9-]+(\.[A-Za-z0-9-]+)*$/`

typedef `std::http_url`

- Base type `string`
- Type constraint `std::validate_type('pydantic.HttpUrl',self)`

typedef `std::ipv4_address`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv4Address',self)`

typedef `std::ipv4_interface`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv4Interface',self)`

typedef `std::ipv4_network`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv4Network',self)`

typedef `std::ipv6_address`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv6Address',self)`

typedef `std::ipv6_interface`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv6Interface',self)`

typedef `std::ipv6_network`

- Base type `string`
- Type constraint `std::validate_type('ipaddress.IPv6Network',self)`

typedef `std::ipv_any_address`

- Base type `string`
- Type constraint `std::validate_type('pydantic.IPvAnyAddress',self)`

typedef `std::ipv_any_interface`

- Base type `string`
- Type constraint `std::validate_type('pydantic.IPvAnyInterface',self)`

typedef `std::ipv_any_network`

- Base type `string`
- Type constraint `std::validate_type('pydantic.IPvAnyNetwork',self)`

typedef `std::name_email`

- Base type `string`
- Type constraint `std::validate_type('pydantic.NameEmail',self)`

typedef `std::negative_float`

- Base type `number`
- Type constraint `std::validate_type('pydantic.NegativeFloat',self)`

typedef `std::negative_int`

- Base type `int`
- Type constraint `std::validate_type('pydantic.NegativeInt',self)`

typedef `std::non_empty_string`

- Base type `string`
- Type constraint `/^(.*\S.*)$/`

typedef `std::package_state`

- Base type `string`
- Type constraint `(((self == 'installed') or (self == 'removed')) or (self == 'latest'))`

typedef `std::port`

- Base type `int`
- Type constraint `((self >= 0) and (self < 65536))`

typedef `std::positive_float`

- Base type `number`
- Type constraint `std::validate_type('pydantic.PositiveFloat',self)`

typedef `std::positive_int`

- Base type `int`
- Type constraint `std::validate_type('pydantic.PositiveInt',self)`

typedef `std::printable_ascii`

- Base type `string`
- Type constraint `std::validate_type('pydantic.constr',self,{'regex': '^[ -~]*$', 'strict': True})`

typedef `std::service_state`

- Base type `string`
- Type constraint `((self == 'running') or (self == 'stopped'))`

typedef `std::time`

- Base type `string`
- Type constraint `std::validate_type('datetime.time',self)`

typedef `std::uuid`

- Base type `string`
- Type constraint `std::validate_type('uuid.UUID',self)`

Entities

entity `std::AgentConfig`

Parents: *std::PurgeableResource*

Control agent settings. Currently these settings are only applied to autostarted agents

attribute bool `autostart`=false

When this flag is set to true, the resource will be exported and set the agent map on the orchestrator. When false, this instance is ignored but can be used to generate agent configuration files.

attribute std::config_agent `agentname`

The name of the agent to which this config applies.

attribute string `agent`='internal'

If a resource is exported, agent manages the resource.

attribute string `uri`='local:'

The uri that indicates how the agent should execute. Currently the following uri are supported: * "" An empty string. This is the same as running it locally * local: Manage resource locally * ssh://[user@]hostname[:port] Login using ssh. When user is left out, root is assumed. For port, the system default is used. * host The actual hostname or ip to use. Although this is not a valid host in uri form it is supported. * A query string can be used to set the properties: * python: The python interpreter to use. The default value is python * retries: The number of retries before giving up. The default number of retries 10 * retry_wait: The time to wait between retries for the remote target to become available. The default wait is 30s. Example: ssh://centos@centos-machine/?python=python3 (This would connect to a the centos machine and use python3 as it's interpreter)

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::ConfigFile`

Parents: *std::File*

A file with often used defaults for configuration files.

attribute int `mode`=644

attribute string `owner`='root'

attribute string `group`='root'

The following implements statements select implementations for this entity:

- *std::reload*, *std::fileHost* constraint `true`

entity `std::Content`

Parents: *std::Entity*

A content block as a prefix or suffix to a file. This blocks are only merged with the content at export time. This is an advanced pattern that can be used to speed up the compilation in very specific use cases.

attribute string? `sorting_key`=null

The key to use to sort the content blocks in the same list. When this attribute is not set value is used as sorting key.

attribute string `value`

The value to prepend or append

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::DefaultDirectory`

Parents: *std::Directory*

A directory that is world readable. It is also writable for its owner root.

attribute int `mode`=755

attribute string `owner`='root'

attribute string `group`='root'

The following implements statements select implementations for this entity:

- *std::reload*, *std::dirHost* constraint `true`

entity `std::Directory`

Parents: *std::Reload*, *std::PurgeableResource*

A directory on the filesystem

attribute string `path`

attribute int `mode`

attribute string `owner`

attribute string `group`

attribute bool `purge_on_delete`=false

relation std::Host `host` [1]

other end: *std::Host.directories [0:*]*

The following implementations are defined for this entity:

- *std::dirHost*

The following implements statements select implementations for this entity:

- *std::reload*, *std::dirHost* constraint `true`

entity `std::DiscoveryResource`

Parents: *std::Resource*

A resource that scans the infrastructure for a certain type of deployment. This resource can be used to facilitate the onboarding of resources that are not yet managed by the orchestrator.

entity `std::Entity`

The entity all other entities inherit from.

relation std::Entity `requires` [0:*]

other end: *std::Entity.provides [0:*]*

relation std::Entity `provides` [0:*]

other end: *std::Entity.requires [0:*]*

The following implementations are defined for this entity:

- *std::none*

entity `std::File`

Parents: *std::Reload*, *std::PurgeableResource*

This represents a file on the filesystem

attribute string `path`

The path of the file

attribute int `mode`

The permissions of the file

attribute string `owner`

The owner of the file

attribute string `group`

The group of the file

attribute string `content`

The file contents

attribute bool `purge_on_delete`=false

attribute bool `send_event`

attribute string `content_seperator`='\n'

relation std::Content `prefix_content` [0:*]

relation std::Content `suffix_content` [0:*]

relation std::Host `host` [1]

other end: *std::Host.files [0:*]*

The following implementations are defined for this entity:

- *std::fileHost*

The following implements statements select implementations for this entity:

- *std::reload*, *std::fileHost* constraint `true`

entity `std::Host`

Parents: *std::ManagedDevice*

A host models a server or computer in the managed infrastructure that has an ip address.

attribute std::ipv_any_address? `ip`=null

The ipaddress of this node.

attribute bool `remote_agent`=false

Start the mgmt agent for this node on the server and use remote io (ssh).

attribute string `remote_user`='root'

The remote user for the remote agent to login with.

attribute std::port `remote_port`=22

The remote port for this remote agent to use.

relation apt::Repository `repository` [0:*]

other end: *apt::Repository.host [1]*

relation std::File `files` [0:*]

other end: *std::File.host [1]*

relation std::Service `services` [0:*]

other end: *std::Service.host [1]*

relation std::Package `packages` [0:*]

other end: *std::Package.host [1]*

relation std::Directory `directories` [0:*]

other end: *std::Directory.host [1]*

relation std::Symlink `symlinks` [0:*]

other end: *std::Symlink.host [1]*

relation std::OS `os` [1]

Each host has an OS defined. This values is mostly used to select implementation in the where clause of an *implement* statement. The `familyof()` plugin can be used for this.

relation std::HostConfig `host_config` [1]

other end: *std::HostConfig.host [1]*

relation std::HostGroup `host_groups` [0:*]

other end: *std::HostGroup.hosts [0:*]*

The following implementations are defined for this entity:

- *std::hostDefaults*

The following implements statements select implementations for this entity:

- *std::hostDefaults* constraint `true`

entity `std::HostConfig`

Parents: *std::Entity*

This represents generic configuration for a host. This entity is used by other modules to include their host specific configuration. This should be instantiated in the implementation of std::Host or subclasses. This host specific configuration cannot be included by just implementing std::Host because possibly subclasses of std::Host are instantiated and implementations are not inherited.

relation std::Host `host` [1]

other end: *std::Host.host_config [1]*

The following implementations are defined for this entity:

- *std::agentConfig*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`
- *std::agentConfig* constraint `((host.ip != null) and host.remote_agent)`

entity `std::HostGroup`

Parents: *std::Entity*

This entity represents a group of hosts. For example a cluster of machines.

attribute string `name`

relation std::Host `hosts` [0:*]

other end: *std::Host.host_groups [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::ManagedDevice`

Parents: *std::Entity*

This interface represents all devices that can be managed

attribute std::hoststring `name`

entity `std::ManagedResource`

Parents: *std::Resource*

A base class for a resource that can be ignored/unmanaged by Inmanta.

attribute bool `managed=true`

This determines whether this resource is managed by Inmanta or not.

entity `std::MutableBool`

Parents: *std::Entity*

Wrapper for boolean values, used to pass a boolean out of an if statement.

Example

```
attr_a = std::MutableBool()
if some_condition:
    attr_a.value = True
else:
    attr_a.value = Null
end
```

attribute bool? `value`

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::MutableNumber`

Parents: *std::Entity*

Wrapper for number values, used to pass a number out of an if statement or to use relations to create a mutuable set of numbers.

Example

```
attr_a = std::MutableNumber()
if some_condition:
    attr_a.value = 3
else:
    attr_a.value = 4
end
```

Example

```
entity Test:
end

Test.string_list [0:] -- std::MutableNumber

a = Test()
a.string_list += std::MutableNumber(value=3)
a.string_list += std::MutableNumber(value=7)
```

attribute number? `value`

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::MutableString`

Parents: *std::Entity*

Wrapper for string values. It can be used to pass a string out of an if statement, or to use relations to create a mutable set of strings.

Example

```
attr_a = std::MutableString()
if some_condition:
    attr_a.value = "a"
else:
    attr_a.value = "b"
end
```

Example

```
entity Test:
end

Test.string_list [0:] -- std::MutableString

a = Test()
a.string_list += std::MutableString(value="value1")
a.string_list += std::MutableString(value="value2")
```

attribute string? `value`

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `std::OS`

Parents: *`std::Entity`*

Defines an operating system

attribute string `name`

The name of the operating system or family of operating systems

attribute number `version`=0.0

A specific version

attribute string? `python_cmd`='python'

Specifies what command should be used to launch the python interpreter on the other end

relation std::OS `member` [0:*]

other end: *`std::OS.family [0:1]`*

relation std::OS `family` [0:1]

other end: *`std::OS.member [0:*]`*

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `std::Package`

Parents: *`std::Reload`*

A software package installed on a managed device.

attribute string `name`

The name of the package to manage

attribute std::package_state `state`

The state of the package. Valid values are 'installed', 'removed' or 'latest'. latest will upgrade the package when an update is available.

relation std::Host `host` [1]

other end: *`std::Host.packages [0:*]`*

The following implementations are defined for this entity:

- *`std::pkgHost`*

The following implements statements select implementations for this entity:

- *`std::reload`*, *`std::pkgHost`* constraint `true`

entity `std::Packages`

Parents: *`std::Entity`*

Defined the state for multiple packages at once

attribute string[] `name`

A list of package names

attribute std::package_state `state`='installed'

The state of the package

relation std::Host `host` [1]

The following implementations are defined for this entity:

- *`std::pkgs`*

The following implements statements select implementations for this entity:

- *`std::pkgs`* constraint `true`

entity `std::PurgeableResource`

Parents: *`std::Resource`*

A base class for a resource that can be purged and can be purged by Inmanta whenever the resource is no longer managed.

attribute bool `purged`=false

Set whether this resource should exist or not.

attribute bool `purge_on_delete`=false

Purge the resource when it is deleted from the configuration model. When this attribute is true, the server will include a resource with purged=true when this resource is no longer included in the configuration model.

entity `std::Reload`

Parents: *`std::Resource`*

An entity to make the (old) reload mechanism compatible with the event mechanism

attribute bool `reload`=false

If a service requires this file, reload or restart the service when this file changes.

attribute bool `send_event`

The following implementations are defined for this entity:

- *`std::reload`*

entity `std::Resource`

Parents: *`std::Entity`*

A base entity for resources that can be exported. This type add specific attributes that are common for most handlers. It is not required to inherit from this entity at the moment but highly recommended for documentation purposes.

attribute bool `send_event`=false

This controls whether a resource should send its deploy state to the resources in its provides.

entity `std::ResourceSet`

Parents: *`std::Entity`*

A ResourceSet describes resources that logically belong together, and can be manipulated independently from other managed resources.

attribute std::non_empty_string `name`

The name of the resource set.

relation std::Resource `resources` [0:*]

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

entity `std::Service`

Parents: *`std::Reload`*

Manage a service on a host.

attribute string `name`

The name of the service to manage

attribute std::service_state `state`

The desired state of the service. Valid values are 'running' or 'stopped'

attribute bool `onboot`

Should the service start on boot.

relation std::Host `host` [1]

other end: *`std::Host.services [0:*]`*

The following implementations are defined for this entity:

- *`std::serviceHost`*

The following implements statements select implementations for this entity:

- *`std::reload`*, *`std::serviceHost`* constraint `true`

entity `std::Symlink`

Parents: *`std::Reload`*, *`std::PurgeableResource`*

A symbolic link on the filesystem

attribute string `source`

attribute string `target`

attribute bool `purge_on_delete`=false

attribute bool `send_event`

relation std::Host `host` [1]

other end: *`std::Host.symlinks [0:*]`*

The following implementations are defined for this entity:

- *`std::symHost`*

The following implements statements select implementations for this entity:

- *`std::reload`*, *`std::symHost`* constraint `true`

entity `std::testing::NullResource`

Parents: *`std::ManagedResource`*, *`std::PurgeableResource`*

A resource that does nothing, for use in tests and examples

attribute string `name`='null'

the name of this resource

attribute string `agentname`='internal'

the name of the agent to deploy this resource on

attribute bool `send_event`=true

attribute bool `fail`=false

when true, this resource will always fail on both dryrun and deploy

The following implements statements select implementations for this entity:

- *`std::none`* constraint `true`

Implementations

implementation `std::agentConfig`

implementation `std::dirHost`

implementation `std::fileHost`

implementation `std::hostDefaults`

implementation `std::none`

An empty implementation that can be used as a safe default.

implementation `std::pkgHost`

implementation `std::pkgs`

implementation `std::reload`

implementation `std::serviceHost`

implementation `std::symHost`

Plugins

`std.`**`add_to_ip`**(*addr: 'std::ipv_any_address'*, *n: 'int'*) → 'std::ipv_any_address'

Add a number to the given ip.

`std.`**`assert`**(*expression: 'bool'*, *message: 'string'* =) → 'any'

Raise assertion error if expression is false

`std.`**`at`**(*objects: 'list'*, *index: 'int'*) → 'any'

Get the item at index

`std.`**`attr`**(*obj: 'any'*, *attr: 'string'*) → 'any'

`std.`**`capitalize`**(*string: 'string'*) → 'string'

Capitalize the given string

`std.`**`contains`**(*dct: 'dict'*, *key: 'string'*) → 'bool'

Check if key exists in dct.

`std.`**`count`**(*item_list: 'list'*) → 'int'

Returns the number of elements in this list.

If any unknowns are present in the list, counts them like any other value. Depending on the unknown semantics in your model this may produce an inaccurate count. For a count that is conservative with respect to unknowns, see *len*.

`std.`**`dict_get`**(*dct: 'dict'*, *key: 'string'*) → 'string'

Get an element from the dict. Raises an exception when the key is not found in the dict

`std.`**`environment`**() → 'string'

Return the environment id

`std.`**`environment_name`**() → 'string'

Return the name of the environment (as defined on the server)

`std.`**`environment_server`**() → 'string'

Return the address of the management server

`std.`**`equals`**(*arg1: 'any'*, *arg2: 'any'*, *desc: 'string' = None*) → 'any'

Compare arg1 and arg2

`std.`**`familyof`**(*member: 'std::OS'*, *family: 'string'*) → 'bool'

Determine if member is a member of the given operating system family

`std.`**`file`**(*path: 'string'*) → 'string'

Return the textual contents of the given file

`std.`**`filter`**(*values: 'list'*, *not_item: 'std::Entity'*) → 'list'

Filter not_item from values

`std.`**`flatten`**(*item_list: 'list'*) → 'list'

Flatten this list

`std.`**`generate_password`**(*pw_id: 'string'*, *length: 'int' = 20*) → 'string'

Generate a new random password and store it in the data directory of the project. On next invocations the stored password will be used.

Parameters

- **`pw_id`** – The id of the password to identify it.
- **`length`** – The length of the password, default length is 20

`std.`**`get_env`**(*name: 'string'*, *default_value: 'string' = None*) → 'string'

`std.`**`get_env_int`**(*name: 'string'*, *default_value: 'int' = None*) → 'int'

`std.`**`getattr`**(*entity: 'std::Entity'*, *attribute_name: 'string'*, *default_value: 'any' = None*, *no_unknown: 'bool' = True*) → 'any'

Return the value of the given attribute. If the attribute does not exist, return the default value.

Attr no_unknown

When this argument is set to true, this method will return the default value when the attribute is unknown.

`std.`**`getfact`**(*resource: 'any'*, *fact_name: 'string'*, *default_value: 'any' = None*) → 'any'

Retrieve a fact of the given resource

`std.`**`hostname`**(*fqdn: 'string'*) → 'string'

Return the hostname part of the fqdn

`std.`**`inlineif`**(*conditional: 'bool'*, *a: 'any'*, *b: 'any'*) → 'any'

An inline if

`std.`**`invert`**(*value: 'bool'*) → 'bool'

Invert a boolean value

`std.`**`ip_address_from_interface`**(*ip_interface: 'std::ipv_any_interface'*) → 'std::ipv_any_address'

Take an ip address with network prefix and only return the ip address

Parameters

`ip_interface` – The interface from where we will extract the ip address

`std.`**`ipindex`**(*addr: 'std::ipv_any_network'*, *position: 'int'*, *keep_prefix: 'bool' = False*) → 'string'

Return the address at position in the network.

Parameters

- **`addr`** – The network address
- **`position`** – The desired position of the address
- **`keep_prefix`** – If the prefix should be included in the result

`std.`**`is_base64_encoded`**(*s: 'string'*) → 'bool'

Check whether the given string is base64 encoded.

`std.`**`is_instance`**(*obj: 'any'*, *cls: 'string'*) → 'bool'

`std.`**`is_unknown`**(*value: 'any'*) → 'bool'

`std.`**`isset`**(*value: 'any'*) → 'bool'

Returns true if a value has been set

`std.`**`item`**(*objects: 'list'*, *index: 'int'*) → 'list'

Return a list that selects the item at index from each of the sublists std.**key_sort**(*items: 'list'*, *key: 'any'*) → 'list'

Sort an array of object on key

std.**len**(*item_list: 'list'*) → 'int'

Returns the number of elements in this list. Unlike *count*, this plugin is conservative when it comes to unknown values. If any unknown is present in the list, the result is also unknown.

std.**length**(*value: 'string'*) → 'int'

Return the length of the string

std.**limit**(*string: 'string'*, *length: 'int'*) → 'string'

Limit the length for the string

Parameters

- **string** – The string to limit the length off
- **length** – The max length of the string

std.**list_files**(*path: 'string'*) → 'list'

List files in a directory

std.**lower**(*string: 'string'*) → 'string'

Return a copy of the string with all the cased characters converted to lowercase.

std.**netmask**(*addr: 'std::ipv_any_interface'*) → 'std::ipv_any_address'

Return the netmask of the CIDR

For instance:

std::print(netmask("192.168.1.100/24")) –> 255.255.255.0

std.**network_address**(*addr: 'std::ipv_any_interface'*) → 'std::ipv_any_address'

Return the network address of the CIDR

For instance:

std::print(network_address("192.168.1.100/24")) –> 192.168.1.0

std.**objid**(*value: 'any'*) → 'string'

std.**password**(*pw_id: 'string'*) → 'string'

Retrieve the given password from a password file. It raises an exception when a password is not found

Parameters

pw_id – The id of the password to identify it.

std.**prefixlen**(*addr: 'std::ipv_any_interface'*) → 'int'

Return the prefixlen of the CIDR

For instance:

std::print(prefixlen("192.168.1.100/24")) –> 24

std.**prefixlength_to_netmask**(*prefixlen: 'int'*) → 'std::ipv4_address'

Given the prefixlength, return the netmask

std.**print**(*message: 'any'*) → 'any'

Print the given message to stdout

std.**replace**(*string: 'string'*, *old: 'string'*, *new: 'string'*) → 'string'

std.**select**(*objects: 'list'*, *attr: 'string'*) → 'list'

Return a list with the select attributes `std.`**`sequence`**(*i: 'int'*, *start: 'int' = 0*, *offset: 'int' = 0*) → 'list'

Return a sequence of i numbers, starting from zero or start if supplied.

Parameters

- **`i`** – The number of elements in the sequence.
- **`start`** – The starting value for the sequence.
- **`offset`** – [Deprecated] An offset value (this parameter will be removed in the future).

Returns

A list containing the sequence of ints.

`std.`**`server_ca`**() → 'string'

`std.`**`server_port`**() → 'int'

`std.`**`server_ssl`**() → 'bool'

`std.`**`server_token`**(*client_types: 'string[]' = ['agent']*) → 'string'

`std.`**`source`**(*path: 'string'*) → 'string'

Return the textual contents of the given file

`std.`**`split`**(*string_list: 'string'*, *delim: 'string'*) → 'list'

Split the given string into a list

Parameters

- **`string_list`** – The list to split into parts
- **`delim`** – The delimeter to split the text by

`std.`**`template`**(*path: 'string'*, ***kwargs: 'any'*) → 'string'

Execute the template in path in the current context. This function will generate a new statement that has dependencies on the used variables.

Parameters

- **`path`** – The path to the jinja2 template that should be resolved.
- **`**kwargs`** – A set of variables that should overwrite the context accessible to the template.

`std.`**`timestamp`**(*dummy: 'any' = None*) → 'int'

Return an integer with the current unix timestamp

Parameters

any – A dummy argument to be able to use this function as a filter

`std.`**`to_number`**(*value: 'any'*) → 'number'

Convert a value to a number

`std.`**`type`**(*obj: 'any'*) → 'any'

`std.`**`unique`**(*item_list: 'list'*) → 'bool'

Returns true if all items in this sequence are unique

`std.`**`unique_file`**(*prefix: 'string'*, *seed: 'string'*, *suffix: 'string'*, *length: 'int' = 20*) → 'string'

`std.`**`upper`**(*string: 'string'*) → 'string'

Return a copy of the string with all the cased characters converted to uppercase.

`std.validate_type`(*fq_type_name: 'string'*, *value: 'any'*, *validation_parameters: 'dict' = None*) → 'bool'

Check whether *value* satisfies the constraints of type *fq_type_name*. When the given type (fq_type_name) requires validation_parameters, they can be provided using the optional *validation_parameters* argument.

The following types require validation_parameters:

- **pydantic.condecimal:**
  gt: Decimal = None ge: Decimal = None lt: Decimal = None le: Decimal = None max_digits: int = None decimal_places: int = None multiple_of: Decimal = None
- **pydantic.confloat and pydantic.conint:**
  gt: float = None ge: float = None lt: float = None le: float = None multiple_of: float = None,
- **pydantic.constr:**
  min_length: int = None max_length: int = None curtail_length: int = None (Only verify the regex on the first curtail_length characters) regex: str = None (The regex is verified via Pattern.match())

Example usage:

- Define a vlan_id type which represent a valid vlan ID (0-4,095):

  typedef vlan_id as number matching std::validate_type("pydantic.conint", self, {"ge": 0, "le": 4095})

Resources

class `std.resources.AgentConfig`

A resource that can modify the agentmap for autostarted agents

- Resource for entity *std::AgentConfig*
- Id attribute `agentname`
- Agent name `agent`
- Handlers *std.resources.AgentConfigHandler*

class `std.resources.Directory`

A directory on a filesystem

- Resource for entity *std::Directory*
- Id attribute `path`
- Agent name `host.name`
- Handlers *std.resources.DirectoryHandler*

class `std.resources.File`

A file on a filesystem

- Resource for entity *std::File*
- Id attribute `path`
- Agent name `host.name`
- Handlers *std.resources.PosixFileProvider*

class `std.resources.Package`

A software package installed on an operating system.

- Resource for entity *std::Package*

- Id attribute `name`
- Agent name `host.name`
- Handlers *apt.AptPackage*, *std.resources.YumPackage*

class `std.resources.`**`Service`**

This class represents a service on a system.

- Resource for entity *std::Service*
- Id attribute `name`
- Agent name `host.name`
- Handlers *std.resources.SystemdService*, *std.resources.ServiceService*, *ubuntu.UbuntuService*

class `std.resources.`**`Symlink`**

A symbolic link on the filesystem

- Resource for entity *std::Symlink*
- Id attribute `target`
- Agent name `host.name`
- Handlers *std.resources.SymlinkProvider*

class `std.resources.`**`Null`**

- Resource for entity *std::testing::NullResource*
- Id attribute `name`
- Agent name `agentname`
- Handlers *std.resources.NullProvider*

Handlers

class `std.resources.`**`YumPackage`**

A Package handler that uses yum

- Handler name `yum`
- Handler for entity *std::Package*

class `std.resources.`**`NullProvider`**

Does nothing at all

- Handler name `null`
- Handler for entity *std::testing::NullResource*

class `std.resources.`**`PosixFileProvider`**

This handler can deploy files on a unix system

- Handler name `posix_file`
- Handler for entity *std::File*

class std.resources.**SystemdService**

A handler for services on systems that use systemd

- Handler name `systemd`
- Handler for entity *std::Service*

class std.resources.**ServiceService**

A handler for services on systems that use service

- Handler name `redhat_service`
- Handler for entity *std::Service*

class std.resources.**DirectoryHandler**

A handler for creating directories

- Handler name `posix_directory`
- Handler for entity *std::Directory*

class std.resources.**SymlinkProvider**

This handler can deploy symlinks on unix systems

- Handler name `posix_symlink`
- Handler for entity *std::Symlink*

class std.resources.**AgentConfigHandler**

- Handler name `agentrest`
- Handler for entity *std::AgentConfig*

11.7.10 Module ubuntu

- License: Apache 2.0
- Version: 0.4.20
- Upstream project: https://github.com/inmanta/ubuntu.git

Handlers

class ubuntu.**UbuntuService**

A handler for services on systems that use upstart

- Handler name `ubuntu_service`
- Handler for entity *std::Service*

11.7.11 Module user

- License: ASL 2
- Version: 0.1.22
- Upstream project: https://github.com/inmanta/user.git

Entities

entity user::Group

Parents: *std::ManagedResource*, *std::PurgeableResource*

attribute string name

attribute bool system=false

relation std::Host host [1]

The following implementations are defined for this entity:

- *user::execGroup*

The following implements statements select implementations for this entity:

- *user::execGroup* constraint true

entity user::User

Parents: *std::ManagedResource*, *std::PurgeableResource*

attribute string name

attribute string group

attribute string[] groups=List()

attribute bool system=false

attribute string shell='/bin/bash'

attribute string homedir

relation std::Host host [1]

The following implementations are defined for this entity:

- *user::execUser*

The following implements statements select implementations for this entity:

- *user::execUser* constraint true

Implementations

implementation user::execGroup

Exec based implementation until a handler is available

implementation user::execUser

Exec based implementation until a handler is available

11.7.12 Module yum

- License: Apache 2.0
- Version: 0.7.8
- Upstream project: https://github.com/inmanta/yum.git

Entities

entity `yum::Repository`

Parents: *`std::Entity`*

A yum repository.

Constraint: The attributes baseurl, mirrorlist and metalink cannot be null at the same time.

attribute string `name`

attribute bool `gpgcheck`=false

attribute bool `enabled`=true

attribute string? `baseurl`=null

attribute string? `mirrorlist`=null

attribute string? `metalink`=null

attribute string `gpgkey`=''

attribute number `metadata_expire`=7200

attribute bool `skip_if_unavailable`=false

relation std::Host `host` [1]

other end: `std::Host.repos [0:*]`

The following implementations are defined for this entity:

- *`yum::validateInput`*
- *`yum::redhatRepo`*

The following implements statements select implementations for this entity:

- *`yum::validateInput`* constraint `true`
- *`yum::redhatRepo`* constraint `std::familyof(host.os,'redhat')`

Implementations

implementation `yum::redhatRepo`

implementation `yum::validateInput`

11.8 REST API reference

11.9 Compatibility

11.9.1 System requirements

The table below shows the system requirements of version 8 of the Inmanta Service Orchestrator.

Table 1: System requirements

Component	Required version
	3.11

Note: This information is also available in a machine-consumable format in the compatibility.json file.

CHAPTER

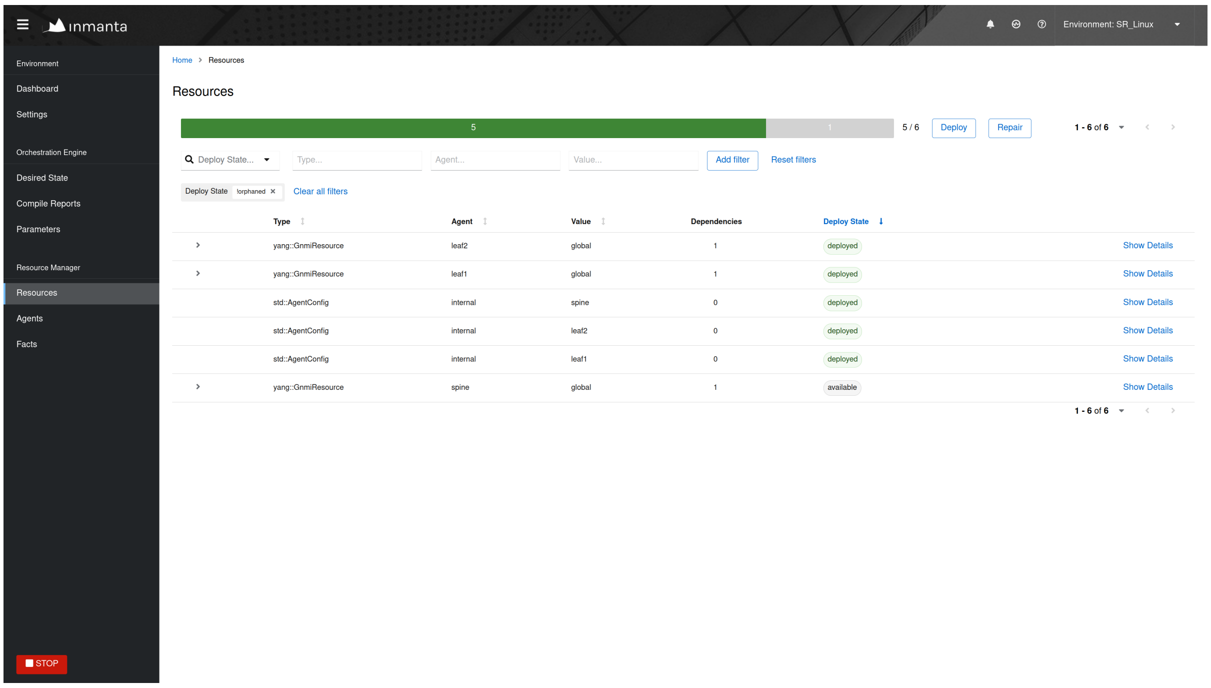
TWELVE

TROUBLESHOOTING

This page describes typical failure scenario's and provides a guideline on how to troubleshoot them.

12.1 A resource is stuck in the state available

When a resource is stuck in the available state, it usually means that the agent, which should deploy the resource, is currently down or paused. Click on the `Resources` tab of the web-console, to get an overview of the different resources in the model. This overview shows the state of each resource and the name of its agent. Filter on resources in the available state and check which resource are ready to be deployed (i.e. a resource without dependencies or a resource for which all dependencies were deployed successfully). The agent of that resource, is the agent that causes the problem. In the figure below, the `global` GnmiResource should be ready to deploy on the `spine` agent.



	Type	Agent	Value	Dependencies	Deploy State	
›	yang::GnmiResource	leaf2	global	1	deployed	Show Details
›	yang::GnmiResource	leaf1	global	1	deployed	Show Details
	std::AgentConfig	internal	spine	0	deployed	Show Details
	std::AgentConfig	internal	leaf2	0	deployed	Show Details
	std::AgentConfig	internal	leaf1	0	deployed	Show Details
›	yang::GnmiResource	spine	global	1	available	Show Details

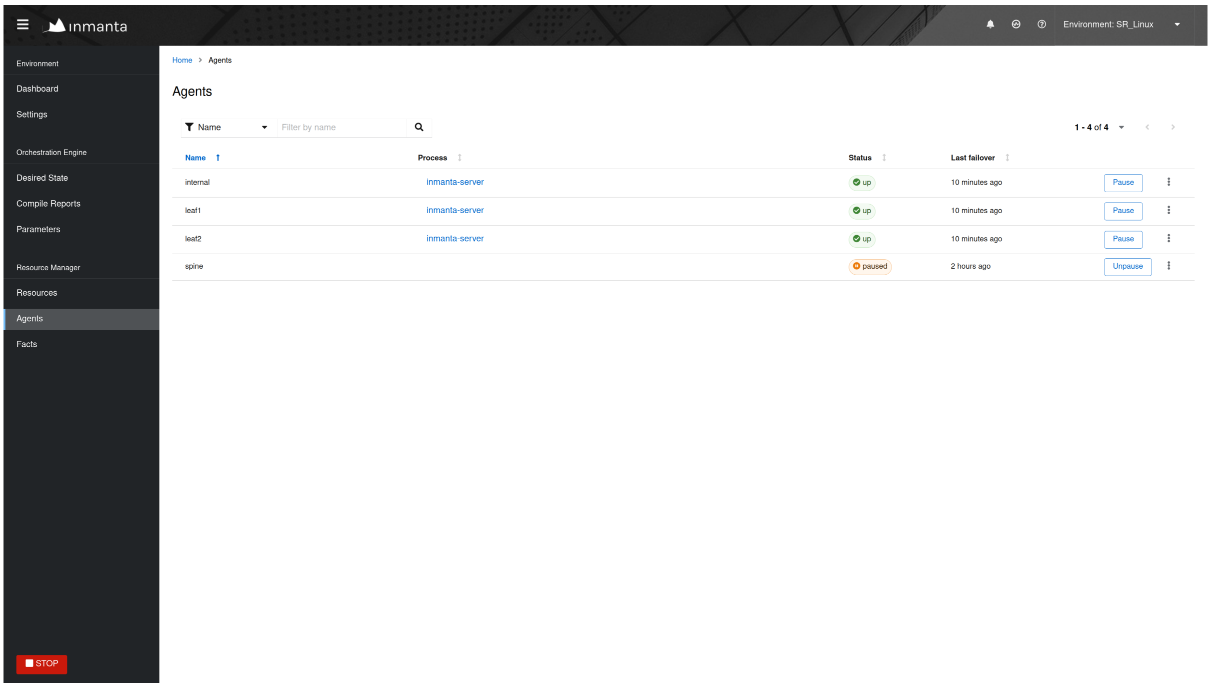
1 - 6 of 6

Next, go to the `Agents` tab of the web-console to verify the state of that agent.

An agent can be in one of the following states:

- Down
- Paused
- Up

Each of the following subsections describes what should be done when the agent is in each of the different states.



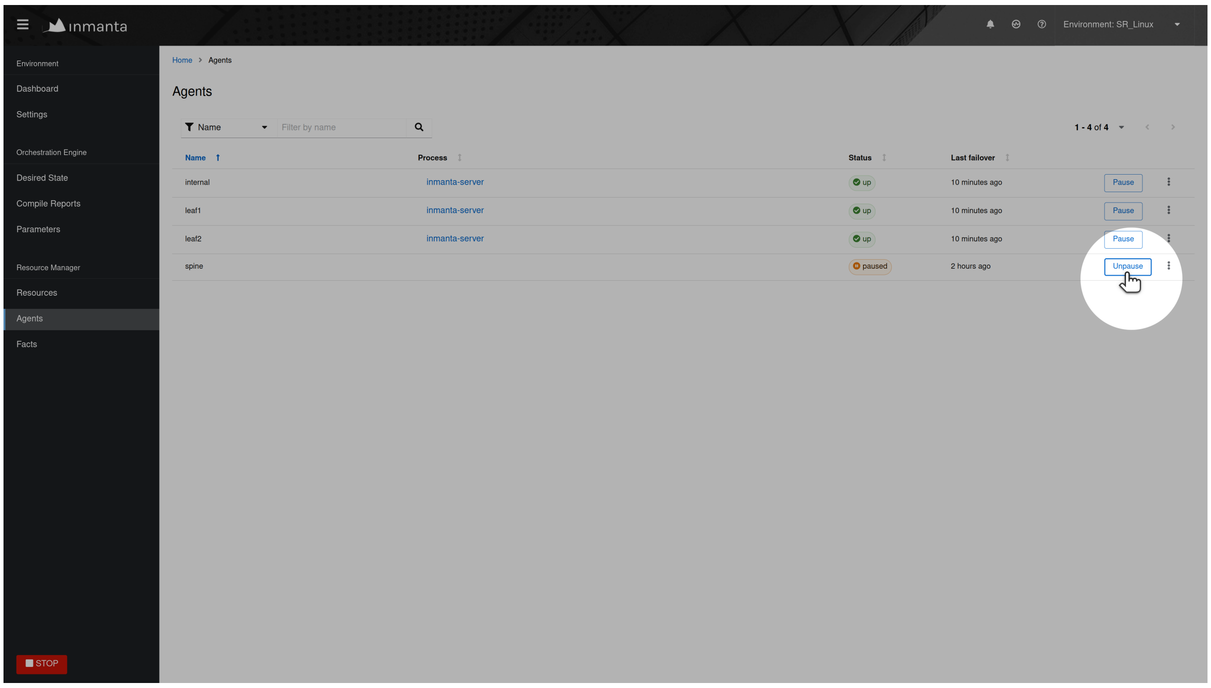
Name	Process	Status	Last failover		
internal	inmanta-server	up	10 minutes ago	Pause	⋮
leaf1	inmanta-server	up	10 minutes ago	Pause	⋮
leaf2	inmanta-server	up	10 minutes ago	Pause	⋮
spine		paused	2 hours ago	Unpause	⋮

12.1.1 The agent is down

The Section *Agent doesn't come up* provides information on how to troubleshoot the scenario where an agent that shouldn't be down is down.

12.1.2 The agent is paused

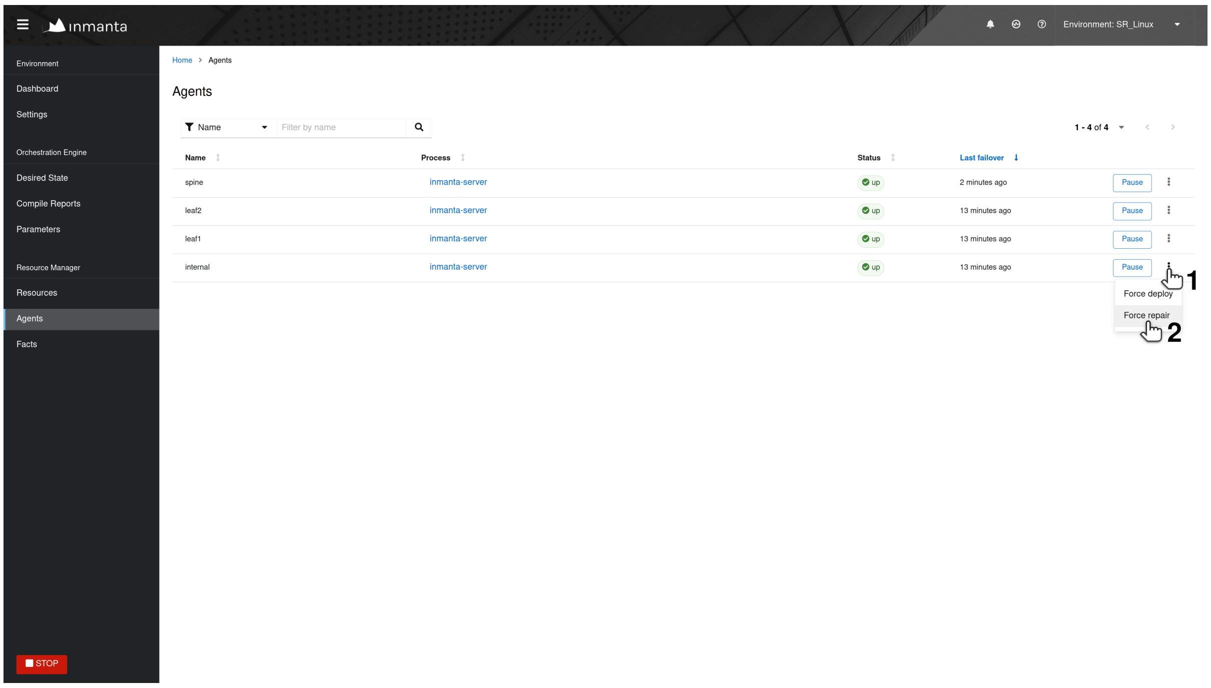
Unpause the agent by clicking the `Unpause` button in the `Agents` tab of the web-console.



Name	Process	Status	Last failover		
internal	inmanta-server	up	10 minutes ago	Pause	⋮
leaf1	inmanta-server	up	10 minutes ago	Pause	⋮
leaf2	inmanta-server	up	10 minutes ago	Pause	⋮
spine		paused	2 hours ago	Unpause	⋮

12.1.3 The agent is up

When the agent is in the up state, it should be ready to deploy resources. Read the agent log to verify it doesn't contain error or warning messages that would explain why the agent is not deploying any resources. For auto-started agents, three different log files exist. The log files are present in `<config.log-dir>/agent-<environment-id>.[log|out|err]`. The environment ID can be found in the URL of the web-console, or in the `Settings` tab. More information about the different log files can be found *here*. For manually started agents the log file is present in `/var/log/inmanta/agent.log`. If the log file doesn't provide any more information, trigger the agent to execute a deployment by clicking on the `Force repair` button in the `Agents` tab of the web-console, as shown in the figure below:



Name	Process	Status	Last failover
spine	inmanta-server	up	2 minutes ago
leaf2	inmanta-server	up	13 minutes ago
leaf1	inmanta-server	up	13 minutes ago
internal	inmanta-server	up	13 minutes ago

When the agent receives the notification from the server, it writes the following log message in its log:

```
INFO     inmanta.agent.agent Agent <agent-name> got a trigger to update in␣
↪environment <environment ID>
```

If the notification from the server doesn't appear in the log file of the agent after clicking the `Force repair` button, the problem is situated on the server side. Check if the server log contains any error messages or warning that could explain the reason why the agent didn't get a notification from the server. The server log file is situated at `<config.log-dir>/server.log`.

12.2 The deployment of a resource fails

When a resource cannot be deployed, it ends up in one of the following deployment states:

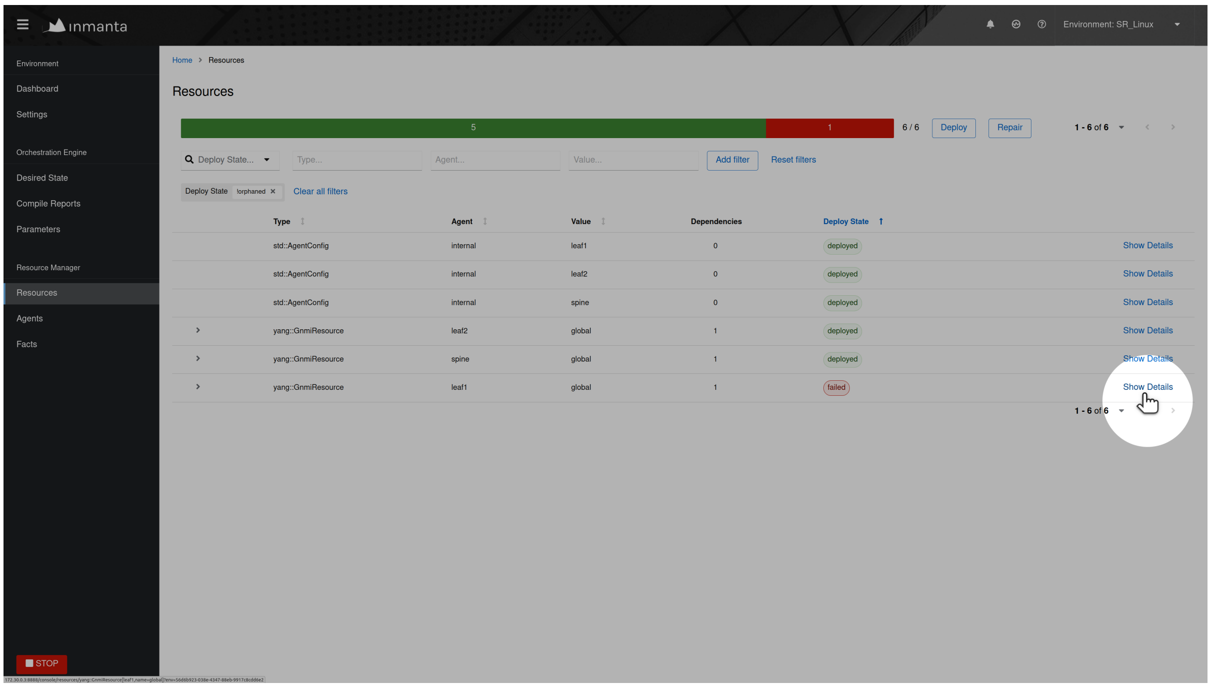
- **failed:** A resource ends up in the `failed` state when the handler of that resource raises an uncaught exception. *Check the log of the resource* to get more details about the issue.
- **unavailable:** A resource ends up in the `unavailable` state when no handler could be found to deploy that resource. *Check the log of the resource* to get more details about the issue.
- **undefined:** A resource ends up in the `undefined` state when an attribute required by that resource, didn't yet resolve to a definite value. Read Section *Check which attributes are undefined* to find out which attributes are undefined.
- **skipped:** When a resource is in the `skipped` state, it can mean two different things. Either the resource cannot be deployed because one of its dependencies ended up in the failed state or the handler itself raised

a SkipResource exception to indicate that the resource in not yet ready to be deployed. The latter case can occur when a VM is still booting for example. *Check the log of the resource* to get more information about actual root cause.

- **skipped_for_undefined:** The `skipped_for_undefined` state indicates that the resource cannot be deployed because one of its dependencies cannot be deployed. *Check the log of the resource* to get information about the actual dependency that cannot be deployed.

12.2.1 Read the logs of a resource

This section describes how to obtain the logs for a specific resource. In the `Resources` tab of the web-console, click on `Show Details` for the desired resource.



	Type	Agent	Value	Dependencies	Deploy State	
	std::AgentConfig	internal	leaf1	0	deployed	Show Details
	std::AgentConfig	internal	leaf2	0	deployed	Show Details
	std::AgentConfig	internal	spine	0	deployed	Show Details
›	yang::GnmiResource	leaf2	global	1	deployed	Show Details
›	yang::GnmiResource	spine	global	1	deployed	Show Details
›	yang::GnmiResource	leaf1	global	1	failed	Show Details

1 - 6 of 6

Next, in the `Logs` tab of this view, the logs can be sorted and filtered. Click on the chevron for a specific log line to display more information, such as the traceback.

12.2.2 Check which attributes are undefined

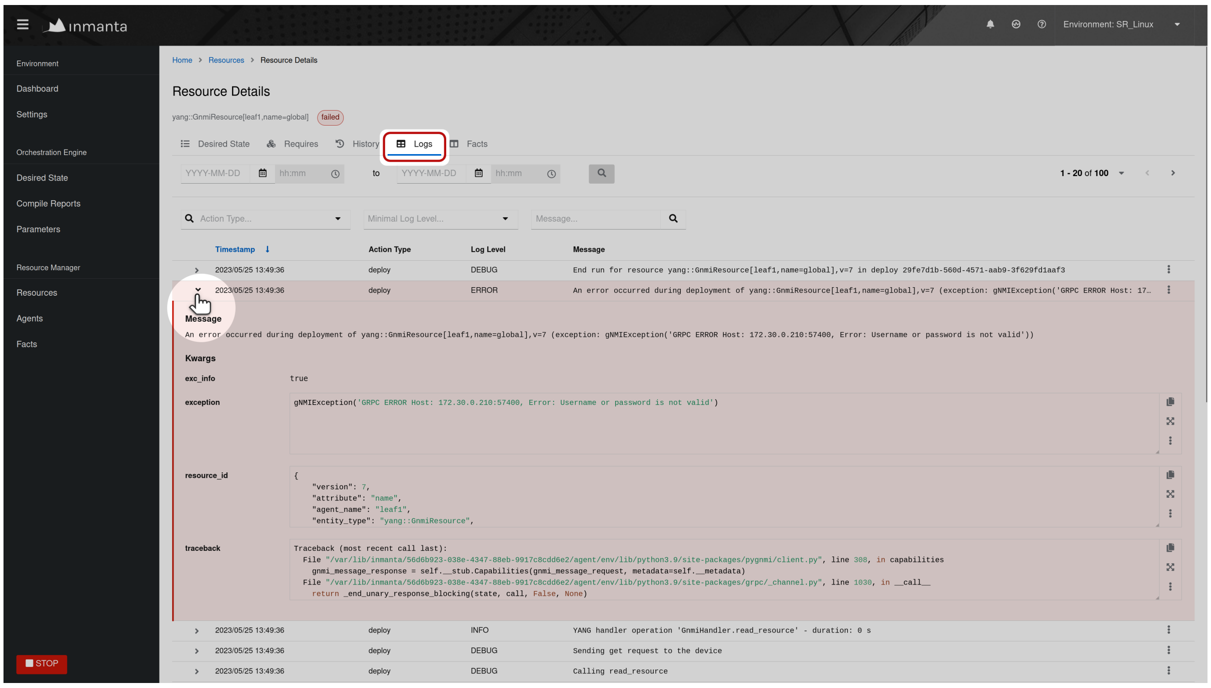
To find out undefined attributes of a resource, click on `Show Details` for the resource in the `undefined` state, as shown in the figure below.

Look for attributes marked as undefined in the list of attributes of that resource (See figure below). Track the source of this attribute down within the configuration model to find out why this attribute is undefined.

12.3 Agent doesn't come up

This section explains how to troubleshoot the problem where an agent is in the down state while it should be up. In the figure shown below, the four agents are down.

Agents can be started in two different ways, either automatically by the inmanta server (auto-started agents) or manually (manually-started) agents. More information about the configuration of both types of agent can be found on *this page*. The Section *Auto-started agents* describes how to troubleshoot this issue for agents started by the Inmanta server. The Section *Manually-started agents* describes how to troubleshoot this issue for agents that were started manually.



Resource Details

yang::GnmiResource[leaf1,name=global] failed

	Timestamp	Action Type	Log Level	Message
›	2023/05/25 13:49:36	deploy	DEBUG	End run for resource yang::GnmiResource[leaf1,name=global],v=7 in deploy 29fe7d1b-560d-4571-aab9-3f629fd1aaf3
˅	2023/05/25 13:49:36	deploy	ERROR	An error occurred during deployment of yang::GnmiResource[leaf1,name=global],v=7 (exception: gNMIException('GRPC ERROR Host: 17…

Message

```
An error occurred during deployment of yang::GnmiResource[leaf1,name=global],v=7 (exception: gNMIException('GRPC ERROR Host: 172.30.0.210:57400, Error: Username or password is not valid'))
```

Kwargs

exc_info `true`

exception

```
gNMIException('GRPC ERROR Host: 172.30.0.210:57400, Error: Username or password is not valid')
```

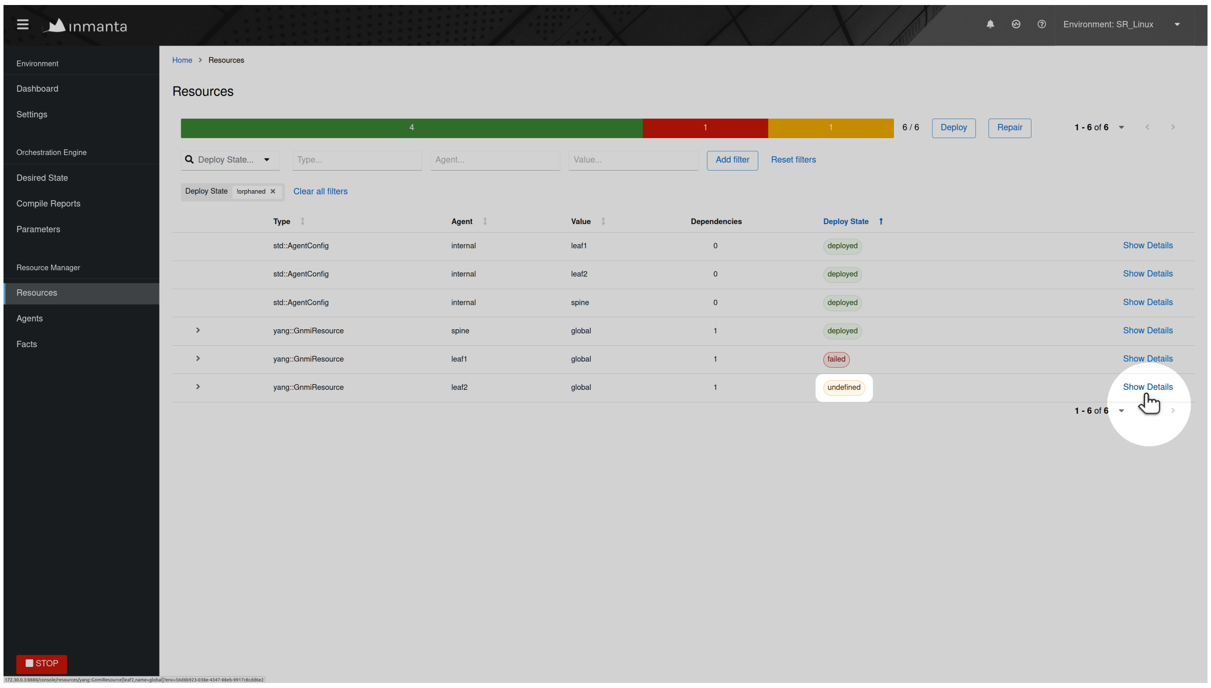
resource_id

```
{
    "version": 7,
    "attribute": "name",
    "agent_name": "leaf1",
    "entity_type": "yang::GnmiResource",
```

traceback

```
Traceback (most recent call last):
  File "/var/lib/inmanta/56d6b923-038e-4347-88eb-9917c8cdd6e2/agent/env/lib/python3.9/site-packages/pygnmi/client.py", line 308, in capabilities
    gnmi_message_response = self.__stub.Capabilities(gnmi_message_request, metadata=self.__metadata)
  File "/var/lib/inmanta/56d6b923-038e-4347-88eb-9917c8cdd6e2/agent/env/lib/python3.9/site-packages/grpc/_channel.py", line 1030, in __call__
    return _end_unary_response_blocking(state, call, False, None)
```

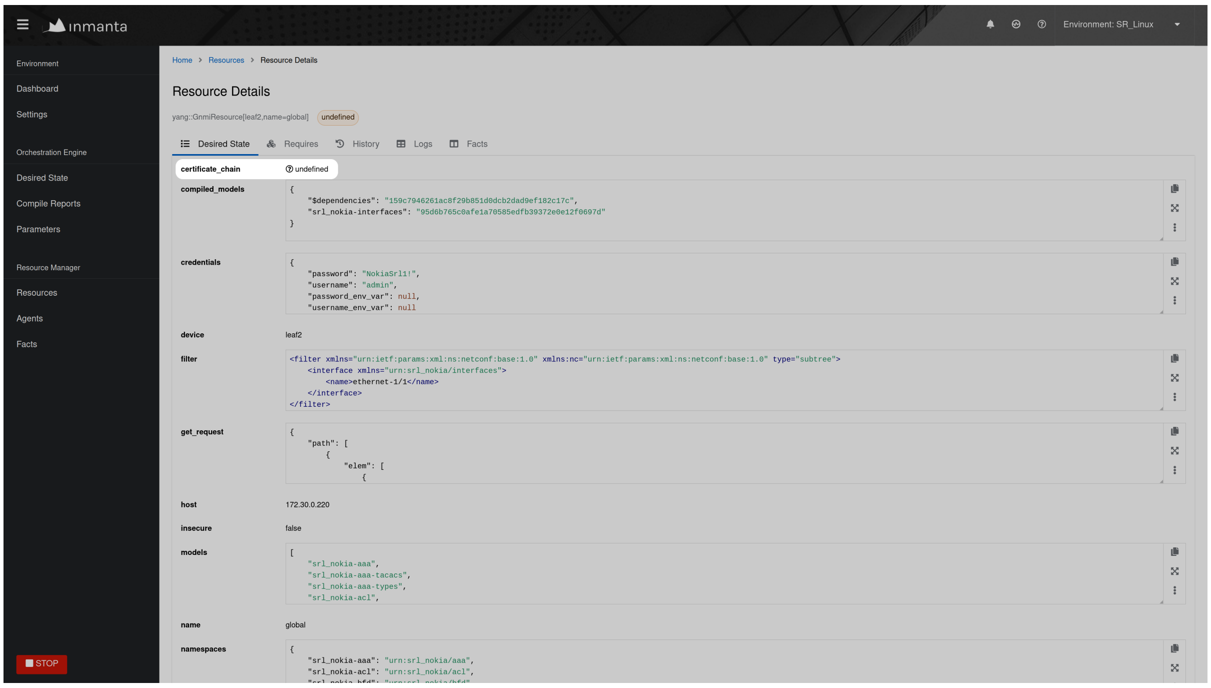
	Timestamp	Action Type	Log Level	Message
›	2023/05/25 13:49:36	deploy	INFO	YANG handler operation 'GnmiHandler.read_resource' - duration: 0 s
›	2023/05/25 13:49:36	deploy	DEBUG	Sending get request to the device
›	2023/05/25 13:49:36	deploy	DEBUG	Calling read_resource



Resources

	Type	Agent	Value	Dependencies	Deploy State	
	std::AgentConfig	internal	leaf1	0	deployed	Show Details
	std::AgentConfig	internal	leaf2	0	deployed	Show Details
	std::AgentConfig	internal	spine	0	deployed	Show Details
›	yang::GnmiResource	spine	global	1	deployed	Show Details
›	yang::GnmiResource	leaf1	global	1	failed	Show Details
›	yang::GnmiResource	leaf2	global	1	undefined	Show Details

1 - 6 of 6



Resource Details

yang::GnmiResource[leaf2,name=global] undefined

certificate_chain: undefined

compiled_models

```
{
    "$dependencies": "159c7946261ac8f29b851d0dcb2dad9ef182c17c",
    "srl_nokia-interfaces": "95d6b765c0afe1a70585edfb39372e0e12f0697d"
}
```

credentials

```
{
    "password": "NokiaSrl1!",
    "username": "admin",
    "password_env_var": null,
    "username_env_var": null
```

device: leaf2

filter

```
<filter xmlns="urn:ietf:params:xml:ns:netconf:base:1.0" xmlns:nc="urn:ietf:params:xml:ns:netconf:base:1.0" type="subtree">
    <interface xmlns="urn:srl_nokia/interfaces">
        <name>ethernet-1/1</name>
    </interface>
</filter>
```

get_request

```
{
    "path": [
        {
            "elem": [
                {
```

host: 172.30.0.220

insecure: false

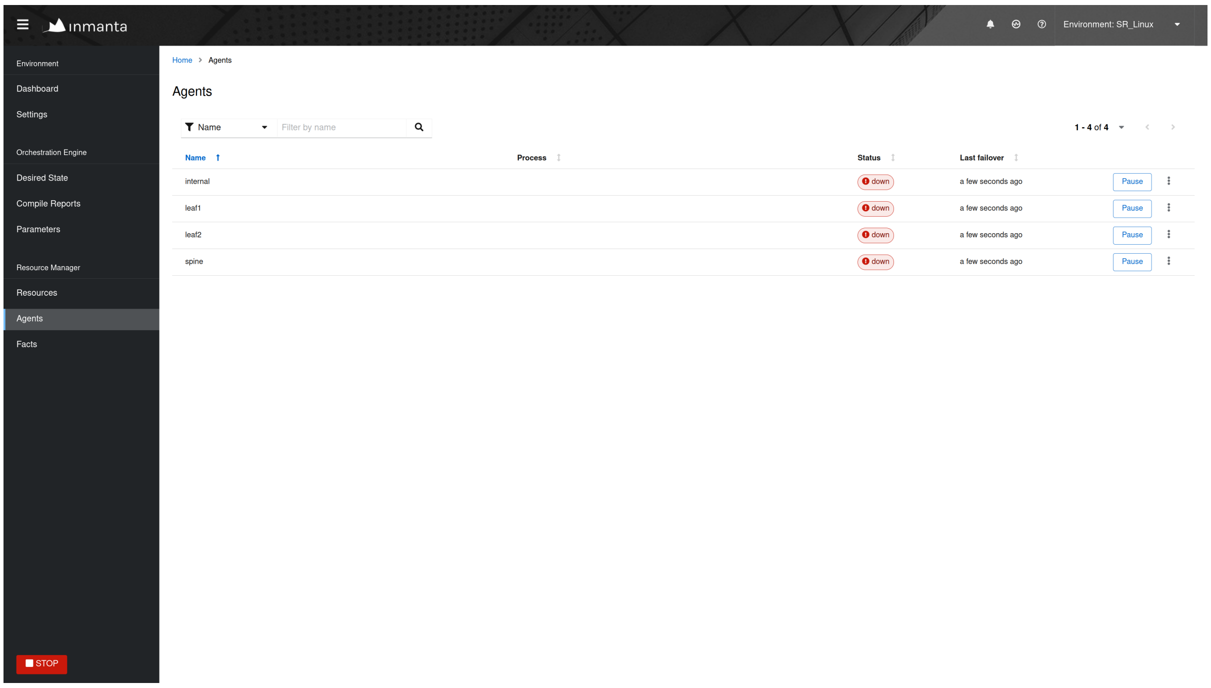
models

```
[
    "srl_nokia-aaa",
    "srl_nokia-aaa-tacacs",
    "srl_nokia-aaa-types",
    "srl_nokia-acl",
```

name: global

namespaces

```
{
    "srl_nokia-aaa": "urn:srl_nokia/aaa",
    "srl_nokia-acl": "urn:srl_nokia/acl",
```

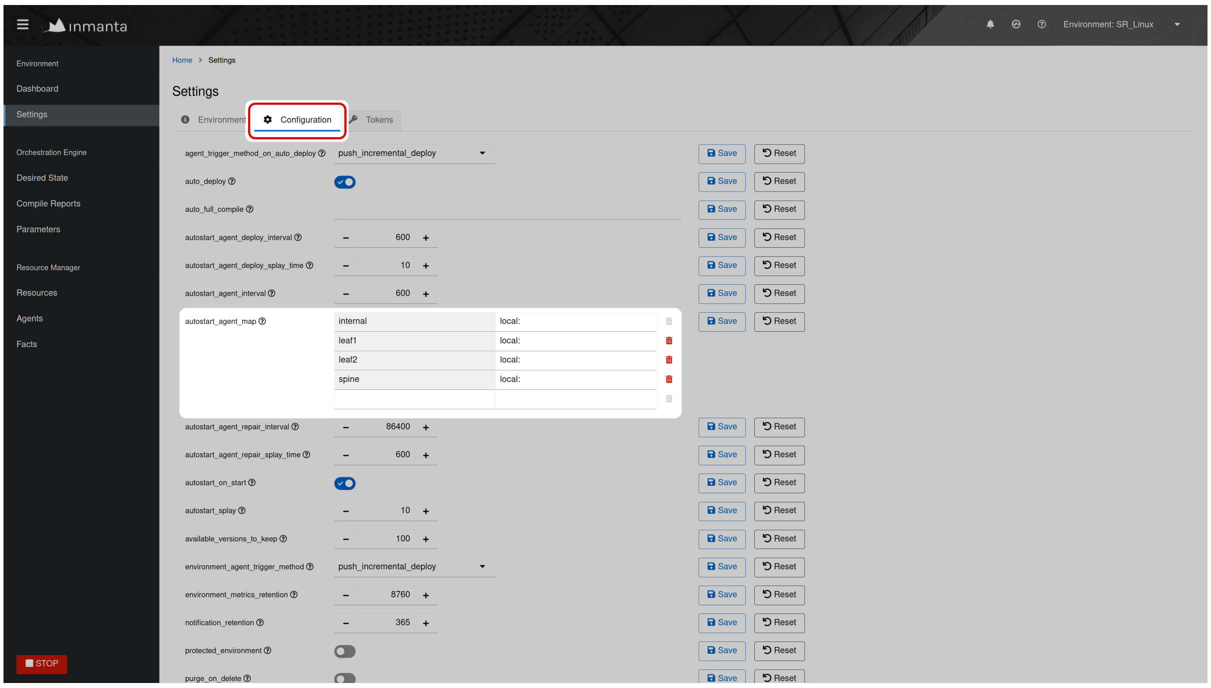


Agents

Name	Process	Status	Last failover		
internal		down	a few seconds ago	Pause	⋮
leaf1		down	a few seconds ago	Pause	⋮
leaf2		down	a few seconds ago	Pause	⋮
spine		down	a few seconds ago	Pause	⋮

12.3.1 Auto-started agents

An auto-started agent is only started when that agent is present in the `autostart_agent_map` environment setting. Verify that requirement in the `Configuration` panel of the `Settings` tab, as shown in the figure below.



agent_trigger_method_on_auto_deploy	push_incremental_deploy
auto_deploy	✓
auto_full_compile	
autostart_agent_deploy_interval	600
autostart_agent_deploy_splay_time	10
autostart_agent_interval	600
autostart_agent_map	internal: local: leaf1: local: leaf2: local: spine: local:
autostart_agent_repair_interval	86400
autostart_agent_repair_splay_time	600
autostart_on_start	✓
autostart_splay	10
available_versions_to_keep	100
environment_agent_trigger_method	push_incremental_deploy
environment_metrics_retention	8760
notification_retention	365
protected_environment	
purge_on_delete	

When the `autostart_agent_map` is configured correctly, but the agent is still not up, read the logs of the auto-started agent . These logs can be found at the following location: `<config.log-dir>/agent-<environment-id>.[log|out|err]`. The environment ID is present in the URL of the web-console. More information about the different log files can be found *here*. When reading those log files, pay specific attention to error messages and warnings that could explain why the agent is marked as down. Also, ensure that the name of the agent under consideration is added as an endpoint to the agent process. The log file should contain the following message when a certain agent is added as an endpoint to the process:

```
inmanta.agent.agent Adding endpoint <agent-name>
```

When the agent is not added as an endpoint, log an issue on https://github.com/inmanta/inmanta-core/issues.

An autostarted-agent connects to the Inmanta server via the address configured in the *`server.server-address`* config option. If this option is set incorrectly, the agent will not be able to connect to the server.

12.3.2 Manually started agents

When a manually-started agent doesn't come up, verify whether the agent process is still running via the following command:

```
$ systemctl status inmanta-agent
```

If the agent process is down, start and enable it via the following command:

```
$ systemctl enable --now inmanta-agent
```

Also check the log file of the manually-started agent. This log file is located at `/var/log/inmanta/agent.log`. The standard output and the standard error streams produced by the agent, can be obtained via journalctl:

```
$ journalctl -u inmanta-agent
```

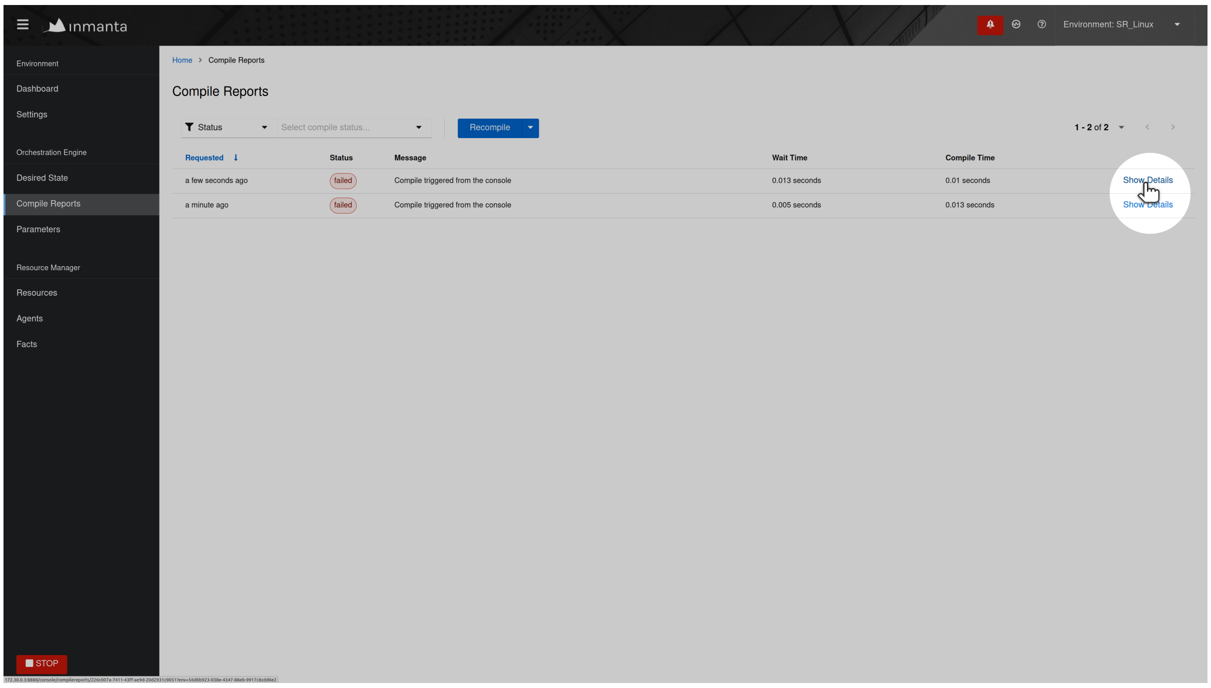
12.3.3 Potential reasons why an agent doesn't start

This section provides a list of potential reasons why an agent wouldn't start:

- **bind-address set incorrectly:** The Inmanta server listens on all the interfaces configured via the *`server.bind-address`* option. If the server doesn't listen on an interface used by a remote agent, the agent will not be able to connect to the server.
- **Authentication issue:** If the Inmanta server has been setup with authentication, a misconfiguration may deny an agent access to the Inmanta API. For example, not configuring a token provider (issuer) with `sign=true` in the `auth_jwt_<ID>` section of the Inmanta configuration file. Documentation on how to configure authentication correctly can be found *here*.
- **SSL problems:** If the Inmanta server is configured to use SSL, the Agent should be configured to use SSL as well (See the SSL-related configuration options in the *`server`* and *`agent_rest_transport`* section of the Inmanta configuration reference)
- **Network issue:** Many network-related issue may exist which don't allow the agent to establish a connection with the Inmanta server. A firewall may blocks traffic between the Inmanta agent and the server, no network route may exist towards the Inmanta server, etc.

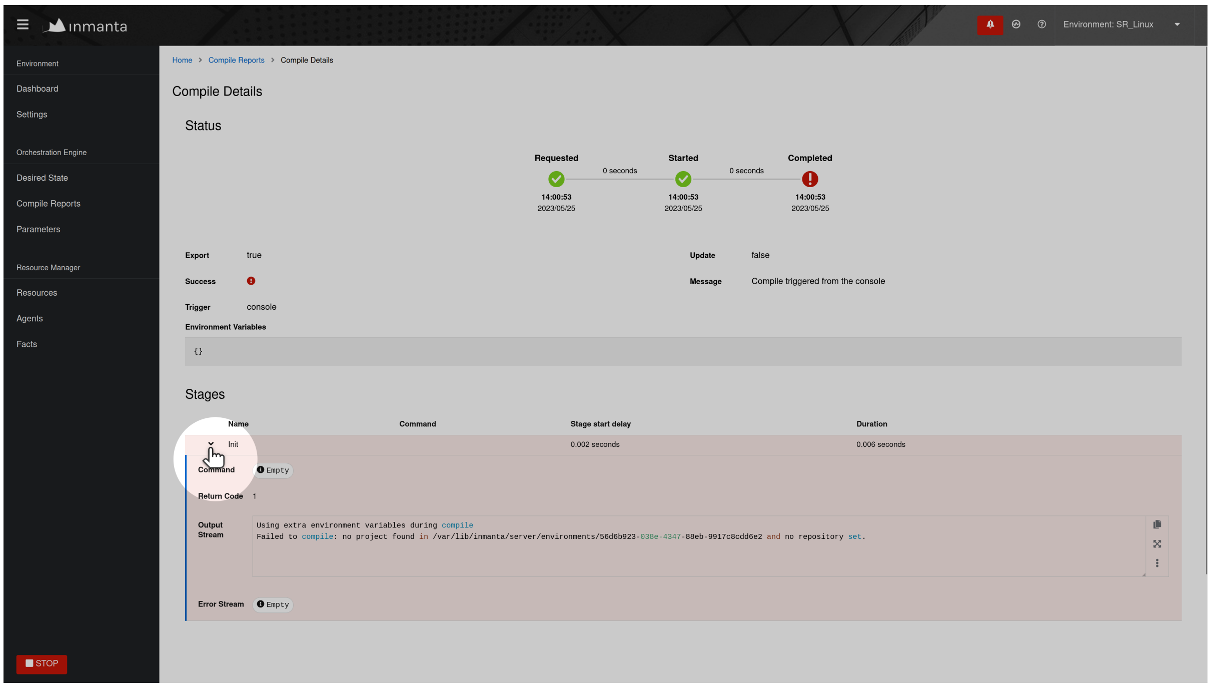
12.4 Recompilation failed

You can trigger a recompilation from the `Compile Reports` tab. It shows a list of compile reports for the latest compilations. Click on `Show Details` to see more information about a given report.



Requested	Status	Message	Wait Time	Compile Time	
a few seconds ago	failed	Compile triggered from the console	0.013 seconds	0.01 seconds	Show Details
a minute ago	failed	Compile triggered from the console	0.005 seconds	0.013 seconds	Show Details

Each step of the compile process is shown. Click on the chevron, as shown below, for a specific step, to display more information such as the output produced by that step and the return code. Verify that the timestamp of the compile report corresponds to the time the compilation was triggered in the web-console. If no compile report was generated or the compile report doesn't show any errors, check the server logs as well. By default the server log is present in `<config.log-dir>/server.log`.



Compile Details

Status

Requested		Started		Completed
	0 seconds		0 seconds	
14:00:53 2023/05/25		14:00:53 2023/05/25		14:00:53 2023/05/25

Export	true	**Update**	false
Success	!	**Message**	Compile triggered from the console
Trigger	console		

Environment Variables

```
{}
```

Stages

Name	Command	Stage start delay	Duration
Init		0.002 seconds	0.006 seconds

Command Empty

Return Code 1

Output Stream

```
Using extra environment variables during compile
Failed to compile: no project found in /var/lib/inmanta/server/environments/56d6b923-038e-4347-88eb-9917c8cdd6e2 and no repository set.
```

Error Stream Empty

12.5 Logs show "empty model" after export

This log message indicates that something went wrong during the compilation or the export of the model to the server. To get more information about the problem, rerun the command with the `-vvv` and the `-X` options. The `-vvv` option increases the log level of the command to the DEBUG level and the `-X` option shows stack traces and errors.

```
$ inmanta -vvv export -X
```

12.6 Compilation fails

In rare situations, the compiler might fail with a `List modified after freeze` or an `Optional variable accessed that has no value` error, even though the model is syntactically correct. The following sections describe why this error occurs and what can be done to make the compilation succeed.

12.6.1 Reason for compilation failure

When the compiler runs, it cannot know upfront how many elements will be added to a relationship. At some stages of the compilation process the compiler has to guess which relations are completely populated in order to be able to continue the compilation process. Heuristics are being used to determine the correct order in which relationships can be considered completely populated. In most situation these heuristics work well, but in rare situations the compiler makes an incorrect decision and considers a relationship to be complete while it isn't. In those situation the compiler crashes with one of the following exception:

- `List modified after freeze`: This error occurs when a relationship with an upper arity larger than one was considered complete too soon.
- `Optional variable accessed that has no value`: This error occurs when a `[0:1]` relationship was considered complete too soon.

The following sections provide information on how this issue can be resolved.

12.6.2 Relationship precedence policy

Warning: The inmanta compiler is very good at determining in which order it should evaluate the orchestration model. Unfortunately in very complex models it might not be able to do this. In that case you can give the compiler some instruction by providing it with relationship precedence rules.

This is a very powerful tool because you can override all the intelligence in the compiler. This means that if you provide the correct rule it will fix the compilation. If you provide a wrong rule it can make this even worse. However, it can never make the orchestrator compile incorrect results.

The above-mentioned problem can be resolved by defining a *relation precedence policy* in the `project.yml` file of an Inmanta project. This policy consists of a list of rules. Each rule defining the order in which two relationships should be considered complete with respect to each other. By providing this policy, it's possible to guide the compiler in making the correct decisions that lead to a successful compilation.

Example: Consider the following `project.yml` file.

```
name: quickstart
modulepath: libs
downloadpath: libs
repo: https://github.com/inmanta/
description: A quickstart project that installs a drupal website.
relation_precedence_policy:
  - "a::EntityA.relation before b::EntityB.other_relation"
```

The last two lines of this file define the relation precedence policy of the project. The policy contains one rule saying that the relationship `relation` of entity `a::EntityA` should be considered completely populated before the relation `other_relation` of entity `b::EntityB` can be considered complete.

Each rule in a relation precedence policy should have the following syntax:

```
<first-type>.<first-relation-name> before <then-type>.<then-relation-name>
```

12.6.3 Compose a relationship precedence policy

Depending on the complexity of your model, it might be difficult to determine the rule(s) that should be added to the relation precedence policy to make the compile succeed. In this section we will provide some guidelines to compose the correct set of rules.

When the compilation of a model fails with a `List modified after freeze` or an `Optional variable accessed that has no value` error, the output from the compiler will contain information regarding which relationship was frozen too soon.

For example, consider the following compiler output:

```
...
Exception explanation
=====================
The compiler could not figure out how to execute this model.

During compilation, the compiler has to decide when it expects a relation to have all␣
↪its elements.
In this compiler run, it guessed that the relation 'finds' on the instance␣
↪maze::ServiceA
(instantiated at /home/centos/maze_project/libs/maze/model/_init.cf:43) would be␣
↪complete with the values [], but the
value maze::SubB (instantiated at /home/centos/maze_project/libs/maze/model/_init.
```

(continues on next page)

(continued from previous page)

```
↪cf:62) was added at
/home/centos/maze_project/libs/maze/model/_init.cf:75
...
```

In the above-mentioned example, the relationship `maze::ServiceA.finds` was incorrectly considered complete. To find the other relation in the ordering conflict, compile the model once more with the log level set to DEBUG by passing the `-vvv` option and grep for the log lines that contain the word `freezing`. The output will contains a log line for each relationship that is considered complete. This way you get an overview regarding the order in which the compiler considers the different relations to be complete.

```
$ inmanta -vvv compile|grep -i freezing
...
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::ServiceA (instantiated␣
↪at /home/centos/maze_project/libs/maze/model/_init.cf:43) maze::ServiceA.finds = []
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::ServiceA (instantiated␣
↪at /home/centos/maze_project/libs/maze/model/_init.cf:43) maze::ServiceA.finds = []
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::ServiceA (instantiated␣
↪at /home/centos/maze_project/libs/maze/model/_init.cf:43) maze::ServiceA.finds = []
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::ServiceA (instantiated␣
↪at /home/centos/maze_project/libs/maze/model/_init.cf:43) maze::ServiceA.finds = []
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::ServiceA (instantiated␣
↪at /home/centos/maze_project/libs/maze/model/_init.cf:43) maze::ServiceA.finds = []
inmanta.execute.schedulerLevel 3 Freezing ListVariable maze::World (instantiated at /
↪home/centos/maze_project/libs/maze/model/_init.cf:10) maze::World.services =␣
↪[maze::ServiceA 7f8feb20f700, maze::ServiceA 7f8feb20faf0, maze::ServiceA␣
↪7f8feb20fee0, maze::ServiceA 7f8feb1e7310, maze::ServiceA 7f8feb1e7700]
Could not set attribute `finds` on instance `maze::ServiceA
...
```

All the relationships frozen after the freeze of the `maze::ServiceA.finds` relationship are potentially causing the compilation problem. In the above-mention example, there is only one, namely the `maze::World.services` relationship.

As such the following rule should be added to the relation precedence policy to resolve this specific conflict:

```
maze::World.services before maze::ServiceA.finds
```

When you compile the model once more with the relation precedence policy in-place, the compilation can either succeed or fail with another `List modified after freeze` or an `Optional variable accessed that has no value` error. The latter case indicates that a second rule should be added to the relation precedence policy.

12.7 Debugging

Debugging the server is possible in case the rpdb package is installed. Installing the `rpdb` package to the virtual environment used by Inmanta by default can be done the following way:

```
$ /opt/inmanta/bin/python3 -m pip install rpdb
```

Rpdb can be triggered by sending a TRAP signal to the inmanta server process.

```
$ kill -5 <PID>
```

After receiving the signal, the process hangs, and it's possible to attach a pdb debugger by connecting to 127.0.0.1, on port 4444 (for example using telnet).

CHAPTER

THIRTEEN

CHANGELOG

13.1 Unreleased changes (2024-05-05)

13.1.1 General changes

Upgrade notes

- Ensure the database is backed up before executing an upgrade.

Bug fixes

- Fix bug where the server restarts while the server venv is in an inconsistent state when the server is upgraded.

Other notes

- The `net` module and the `ip` modules are no longer supported. (#448)

13.1.2 Inmanta-core: unreleased changes (2024-05-05)

New features

- Added support for forking agent executor (#7524)

Upgrade notes

- Handlers can now access the agent via 'inmanta.agent.executor.AgentInstance' instead of `inmanta.agent.agent.AgentInstance`. If you have developed a handler using `self._agent` to access agent internals, it may break.

13.1.3 Inmanta-core: release 12.0.0 (2024-05-05)

No changelog entries.

13.1.4 Inmanta-lsm: unreleased changes (2024-05-05)

Bug fixes

- Fix race condition: Requesting a state transition with the `lsm_services_resources_set_state` from a non-exporting state to an exporting state was sometimes causing compiles scheduled earlier to fail. (#1638)

CHAPTER FOURTEEN

PDF VERSION

Download: `inmanta.pdf`

PYTHON MODULE INDEX

i

inmanta.model, 115
inmanta.protocol.methods, 318
inmanta.protocol.methods_v2, 330

INDEX

Symbols

_abc_impl (*inmanta.agent.handler.ResourceHandler attribute*), 298
_diff() (*inmanta.agent.handler.ResourceHandler method*), 299
--action
    inmanta-cli-action-log-list command line option, 253
--action-id
    inmanta-cli-action-log-show-messages command line option, 254
--agent
    inmanta-cli-agent-pause command line option, 254
    inmanta-cli-agent-unpause command line option, 255
    inmanta-cli-token-create command line option, 264
--all
    inmanta-cli-agent-pause command line option, 254
    inmanta-cli-agent-unpause command line option, 255
--api
    inmanta-cli-token-create command line option, 264
--branch
    inmanta-cli-environment-create command line option, 255
    inmanta-cli-environment-modify command line option, 256
--compiler
    inmanta-cli-token-create command line option, 264
--environment
    inmanta-cli-action-log-list command line option, 253
    inmanta-cli-action-log-show-messages command line option, 254
    inmanta-cli-agent-list command line option, 254
    inmanta-cli-agent-pause command line option, 254
    inmanta-cli-agent-unpause command line option, 255
    inmanta-cli-environment-setting-delete command line option, 258
    inmanta-cli-environment-setting-get command line option, 258
    inmanta-cli-environment-setting-list command line option, 258
    inmanta-cli-environment-setting-set command line option, 258
    inmanta-cli-lsm-resources-list command line option, 260
    inmanta-cli-lsm-service-entities-list command line option, 260
    inmanta-cli-lsm-service-instances-list command line option, 261
    inmanta-cli-monitor command line option, 261
    inmanta-cli-param-get command line option, 261
    inmanta-cli-param-list command line option, 262
    inmanta-cli-param-set command line option, 262
    inmanta-cli-token-create command line option, 264
    inmanta-cli-version-list command line option, 265
    inmanta-cli-version-release command line option, 265
    inmanta-cli-version-report command line option, 265
--format
    inmanta-cli-environment-show command line option, 259
--full
    inmanta-cli-version-release command line option, 265
--host
    inmanta-cli command line option, 253
    inmanta-cli-lsm command line option, 259
--instance
    inmanta-cli-lsm-resources-list command line option, 260
--key
    inmanta-cli-environment-setting-delete command line option, 258
    inmanta-cli-environment-setting-get

command line option, 258
inmanta-cli-environment-setting-set command line option, 258
--name
inmanta-cli-environment-create command line option, 255
inmanta-cli-environment-modify command line option, 256
inmanta-cli-param-get command line option, 261
inmanta-cli-param-set command line option, 262
inmanta-cli-project-create command line option, 262
inmanta-cli-project-modify command line option, 263
--port
inmanta-cli command line option, 253
inmanta-cli-lsm command line option, 259
--project
inmanta-cli-environment-create command line option, 255
--push
inmanta-cli-version-release command line option, 265
--repo-url
inmanta-cli-environment-create command line option, 255
inmanta-cli-environment-modify command line option, 256
--resource
inmanta-cli-param-get command line option, 261
--rvid
inmanta-cli-action-log-list command line option, 253
inmanta-cli-action-log-show-messages command line option, 254
--save
inmanta-cli-environment-create command line option, 255
--service-entity
inmanta-cli-lsm-resources-list command line option, 260
inmanta-cli-lsm-service-instances-list command line option, 261
--update
inmanta-cli-environment-recompile command line option, 257
--value
inmanta-cli-environment-setting-set command line option, 258
inmanta-cli-param-set command line option, 262
--version
inmanta-cli-lsm-resources-list command line option, 260
inmanta-cli-version-report command line option, 265
-b
inmanta-cli-environment-create command line option, 255
inmanta-cli-environment-modify command line option, 256
-e
inmanta-cli-action-log-list command line option, 253
inmanta-cli-action-log-show-messages command line option, 254
inmanta-cli-agent-list command line option, 254
inmanta-cli-agent-pause command line option, 254
inmanta-cli-agent-unpause command line option, 255
inmanta-cli-environment-setting-delete command line option, 258
inmanta-cli-environment-setting-get command line option, 258
inmanta-cli-environment-setting-list command line option, 258
inmanta-cli-environment-setting-set command line option, 258
inmanta-cli-lsm-resources-list command line option, 260
inmanta-cli-lsm-service-entities-list command line option, 260
inmanta-cli-lsm-service-instances-list command line option, 261
inmanta-cli-monitor command line option, 261
inmanta-cli-param-get command line option, 261
inmanta-cli-param-list command line option, 262
inmanta-cli-param-set command line option, 262
inmanta-cli-token-create command line option, 264
inmanta-cli-version-list command line option, 265
inmanta-cli-version-release command line option, 265
inmanta-cli-version-report command line option, 265
-i
inmanta-cli-lsm-resources-list command line option, 260
inmanta-cli-version-report command line option, 265
-k
inmanta-cli-environment-setting-delete command line option, 258
inmanta-cli-environment-setting-get command line option, 258

inmanta-cli-environment-setting-set
command line option, 258
-l
inmanta-cli-version-report command
line option, 265
-n
inmanta-cli-environment-create command
line option, 255
inmanta-cli-environment-modify command
line option, 256
inmanta-cli-project-create command
line option, 262
inmanta-cli-project-modify command
line option, 263
-o
inmanta-cli-environment-setting-set
command line option, 258
-p
inmanta-cli-environment-create command
line option, 255
inmanta-cli-version-release command
line option, 265
-r
inmanta-cli-environment-create command
line option, 255
inmanta-cli-environment-modify command
line option, 256
-s
inmanta-cli-environment-create command
line option, 255
inmanta-cli-lsm-resources-list command
line option, 260
inmanta-cli-lsm-service-instances-list
command line option, 261
-u
inmanta-cli-environment-recompile
command line option, 257
-v
inmanta-cli-lsm-resources-list command
line option, 260

A

add_change() (*inmanta.agent.handler.HandlerContext method*), 298
add_changes() (*inmanta.agent.handler.HandlerContext method*), 298
add_function() (*inmanta.plugins.PluginMeta class method*), 295
add_user() (*in module inmanta.protocol.methods_v2*), 330
agent, **235**
agent_action() (*in module inmanta.protocol.methods_v2*), 330
all_agents_action() (*in module inmanta.protocol.methods_v2*), 330
ApplicationContext (*class in inmanta.server.extensions*), 351
apt.AptPackage (*built-in class*), 354
apt::Repository, 354
apt::Repository.base_url, 354
apt::Repository.host, 354
apt::Repository.name, 354
apt::Repository.release, 354
apt::Repository.repo, 354
apt::Repository.trusted, 354
apt::simpleRepo, 354
Attribute (*class in inmanta.ast.attribute*), 307
Attribute (*class in inmanta.model*), 115
available() (*inmanta.agent.handler.CRUDHandler method*), 300
aws.ElasticSearch (*built-in class*), 362
aws.ElasticSearchHandler (*built-in class*), 362
aws.ELB (*built-in class*), 361
aws.ELBHandler (*built-in class*), 362
aws.elbid()
built-in function, 361
aws.get_api_id()
built-in function, 361
aws.InternetGateway (*built-in class*), 361
aws.InternetGatewayHandler (*built-in class*), 363
aws.RDS (*built-in class*), 362
aws.RDSHandler (*built-in class*), 362
aws.Route (*built-in class*), 361
aws.RouteHandler (*built-in class*), 363
aws.SecurityGroup (*built-in class*), 361
aws.SecurityGroupHandler (*built-in class*), 363
aws.Subnet (*built-in class*), 361
aws.SubnetHandler (*built-in class*), 363
aws.VirtualMachine (*built-in class*), 362
aws.VirtualMachineHandler (*built-in class*), 362
aws.Volume (*built-in class*), 362
aws.VolumeHandler (*built-in class*), 362
aws.VPC (*built-in class*), 361
aws.VPCHandler (*built-in class*), 363
aws::agentConfig, 360
aws::analytics::ElasticSearch, 359
aws::analytics::ElasticSearch.access_policies, 360
aws::analytics::ElasticSearch.automated_snapshot_start_h
360
aws::analytics::ElasticSearch.dedicated_master_count, 360
aws::analytics::ElasticSearch.dedicated_master_enabled, 359
aws::analytics::ElasticSearch.dedicated_master_type, 359
aws::analytics::ElasticSearch.domain_name, 359
aws::analytics::ElasticSearch.ebs_enabled, 360
aws::analytics::ElasticSearch.elasticsearch_version, 359
aws::analytics::ElasticSearch.instance_count, 359
aws::analytics::ElasticSearch.instance_type, 359

aws::analytics::ElasticSearch.volume_size, 360
aws::analytics::ElasticSearch.volume_type, 360
aws::analytics::ElasticSearch.zone_awareness_enabled, 359
aws::AWSResource, 355
aws::AWSResource.provider, 355
aws::database::RDS, 360
aws::database::RDS.allocated_storage, 360
aws::database::RDS.engine, 360
aws::database::RDS.engine_version, 360
aws::database::RDS.flavor, 360
aws::database::RDS.master_user_name, 360
aws::database::RDS.master_user_password, 360
aws::database::RDS.name, 360
aws::database::RDS.port, 360
aws::database::RDS.public, 360
aws::database::RDS.subnet_group, 360
aws::database::RDS.tags, 360
aws::direction, 355
aws::ELB, 355
aws::ELB.dest_port, 355
aws::ELB.instances, 355
aws::ELB.listen_port, 355
aws::ELB.name, 355
aws::ELB.protocol, 355
aws::ELB.security_group, 355
aws::GroupRule, 355
aws::GroupRule.remote_group, 355
aws::instance_tenancy, 355
aws::InternetGateway, 355
aws::InternetGateway.name, 356
aws::InternetGateway.vpc, 356
aws::IPrule, 355
aws::IPrule.remote_prefix, 355
aws::protocol, 355
aws::Provider, 356
aws::Provider.access_key, 356
aws::Provider.auto_agent, 356
aws::Provider.availability_zone, 356
aws::Provider.name, 356
aws::Provider.region, 356
aws::Provider.secret_key, 356
aws::req, 360
aws::Route, 356
aws::Route.destination, 356
aws::Route.nexthop, 356
aws::Route.vpc, 356
aws::SecurityGroup, 356
aws::SecurityGroup.description, 356
aws::SecurityGroup.manage_all, 356
aws::SecurityGroup.name, 356
aws::SecurityGroup.retries, 356
aws::SecurityGroup.rules, 357
aws::SecurityGroup.vpc, 357
aws::SecurityGroup.wait, 357
aws::SecurityRule, 357
aws::SecurityRule.direction, 357
aws::SecurityRule.group, 357
aws::SecurityRule.ip_protocol, 357
aws::SecurityRule.port, 357
aws::SecurityRule.port_max, 357
aws::SecurityRule.port_min, 357
aws::Subnet, 357
aws::Subnet.availability_zone, 357
aws::Subnet.cidr_block, 357
aws::Subnet.map_public_ip_on_launch, 357
aws::Subnet.name, 357
aws::Subnet.vpc, 357
aws::VirtualMachine, 358
aws::VirtualMachine.name, 358
aws::VirtualMachine.public_key, 359
aws::VirtualMachine.security_groups, 359
aws::VirtualMachine.subnet, 359
aws::VirtualMachine.tags, 359
aws::VirtualMachine.volumes, 359
aws::VMAttributes, 357
aws::VMAttributes.ebs_optimized, 358
aws::VMAttributes.flavor, 358
aws::VMAttributes.ignore_extra_volumes, 358
aws::VMAttributes.ignore_wrong_image, 358
aws::VMAttributes.image, 358
aws::VMAttributes.root_volume_size, 358
aws::VMAttributes.root_volume_type, 358
aws::VMAttributes.source_dest_check, 358
aws::VMAttributes.subnet_id, 358
aws::VMAttributes.user_data, 358
aws::Volume, 359
aws::Volume.attachmentpoint, 359
aws::Volume.availability_zone, 359
aws::Volume.encrypted, 359
aws::Volume.name, 359
aws::Volume.size, 359
aws::Volume.tags, 359
aws::Volume.vm, 359
aws::Volume.volume_type, 359
aws::VPC, 358
aws::VPC.cidr_block, 358
aws::VPC.enableDnsHostnames, 358
aws::VPC.enableDnsSupport, 358
aws::VPC.instance_tenancy, 358
aws::VPC.internet_gateway, 358
aws::VPC.name, 358
aws::VPC.routes, 358
aws::VPC.subnets, 358

B

BadRequest (*class in inmanta.protocol.exceptions*), 113
BaseDocument (*class in inmanta.data*), 313
BaseHttpException (*class in inmanta.protocol.exceptions*), 113
BaseModel (*class in inmanta.data.model*), 317
Bool (*class in inmanta.ast.type*), 312

built-in function
  aws.elbid(), 361
  aws.get_api_id(), 361
  exec.in_shell(), 365
  lsm.all(), 372
  lsm.context_v2_unwrapper(), 372
  lsm.current_state(), 374
  lsm.format(), 374
  lsm.fsm_to_dot(), 374
  lsm.has_current_state(), 374
  lsm.is_validating(), 375
  lsm.render_dot(), 375
  lsm.update_read_only_attribute(), 375
  lsm.validate_service_index(), 375
  openstack.find_flavor(), 390
  openstack.find_image(), 390
  ssh.get_private_key(), 397
  ssh.get_public_key(), 397
  ssh.get_putty_key(), 397
  std.add_to_ip(), 409
  std.assert(), 409
  std.at(), 409
  std.attr(), 409
  std.capitalize(), 409
  std.contains(), 409
  std.count(), 409
  std.dict_get(), 409
  std.environment(), 409
  std.environment_name(), 409
  std.environment_server(), 409
  std.equals(), 409
  std.familyof(), 409
  std.file(), 409
  std.filter(), 409
  std.flatten(), 409
  std.generate_password(), 410
  std.get_env(), 410
  std.get_env_int(), 410
  std.getattr(), 410
  std.getfact(), 410
  std.hostname(), 410
  std.inlineif(), 410
  std.invert(), 410
  std.ip_address_from_interface(), 410
  std.ipindex(), 410
  std.is_base64_encoded(), 410
  std.is_instance(), 410
  std.is_unknown(), 410
  std.isset(), 410
  std.item(), 410
  std.key_sort(), 410
  std.len(), 411
  std.length(), 411
  std.limit(), 411
  std.list_files(), 411
  std.lower(), 411
  std.netmask(), 411
  std.network_address(), 411
  std.objid(), 411
  std.password(), 411
  std.prefixlen(), 411
  std.prefixlength_to_netmask(), 411
  std.print(), 411
  std.replace(), 411
  std.select(), 411
  std.sequence(), 411
  std.server_ca(), 412
  std.server_port(), 412
  std.server_ssl(), 412
  std.server_token(), 412
  std.source(), 412
  std.split(), 412
  std.template(), 412
  std.timestamp(), 412
  std.to_number(), 412
  std.type(), 412
  std.unique(), 412
  std.unique_file(), 412
  std.upper(), 412
  std.validate_type(), 412

C

cache() (*in module inmanta.agent.handler*), 297
calculate_diff() (*inmanta.agent.handler.CRUDHandler method*), 300
can_reload() (*inmanta.agent.handler.CRUDHandler method*), 300
cast() (*inmanta.ast.type.Primitive method*), 311
category (*inmanta.ast.export.Error attribute*), 351
character (*inmanta.ast.export.Position attribute*), 353
check_facts() (*inmanta.agent.handler.CRUDHandler method*), 300
check_facts() (*inmanta.agent.handler.ResourceHandler method*), 299
check_resource() (*inmanta.agent.handler.CRUDHandler method*), 300
check_resource() (*inmanta.agent.handler.ResourceHandler method*), 299
chmod() (*inmanta.agent.io.local.LocalIO method*), 304
chown() (*inmanta.agent.io.local.LocalIO method*), 304
clear() (*inmanta.plugins.PluginMeta class method*), 295
clear_database() (*in module inmanta.db.util*), 317
clear_environment() (*in module inmanta.protocol.methods*), 318
clone() (*inmanta.resources.Resource method*), 296
close() (*inmanta.agent.handler.CRUDHandler method*), 301
close() (*inmanta.agent.io.local.LocalIO method*), 304
code (*inmanta.protocol.common.Result attribute*), 313

ColumnNotFound (*class in inmanta.data.schema*), 114
Compile (*class in inmanta.data*), 314
compile_details() (*in module inmanta.protocol.methods_v2*), 331
CompileData (*class in inmanta.data.model*), 351
CompilerException (*class in inmanta.ast*), 294
configuration model, **235**
ConfigurationModel (*class in inmanta.data*), 315
Conflict (*class in inmanta.protocol.exceptions*), 114
ConstraintType (*class in inmanta.ast.type*), 312
Context (*class in inmanta.plugins*), 294
create_environment() (*in module inmanta.protocol.methods*), 318
create_project() (*in module inmanta.protocol.methods*), 319
create_resource() (*inmanta.agent.handler.CRUDHandler method*), 301
create_token() (*in module inmanta.protocol.methods*), 319
CRITICAL (*inmanta.const.LogLevel attribute*), 293
CRUDHandler (*class in inmanta.agent.handler*), 300

D

DEBUG (*inmanta.const.LogLevel attribute*), 293
delete_environment() (*in module inmanta.protocol.methods*), 319
delete_param() (*in module inmanta.protocol.methods*), 319
delete_project() (*in module inmanta.protocol.methods*), 319
delete_resource() (*inmanta.agent.handler.CRUDHandler method*), 301
delete_setting() (*in module inmanta.protocol.methods*), 319
delete_user() (*in module inmanta.protocol.methods_v2*), 331
delete_version() (*in module inmanta.protocol.methods*), 319
dependency_manager() (*in module inmanta.export*), 307
deploy (*inmanta.const.ResourceAction attribute*), 294
deploy() (*in module inmanta.protocol.methods*), 320
deploy() (*inmanta.agent.handler.CRUDHandler method*), 301
desired state, **235**
Dict (*class in inmanta.ast.type*), 312
DictPath (*class in inmanta.util.dict_path*), 102
diff() (*in module inmanta.protocol.methods*), 320
DirectValue (*class in inmanta.model*), 115
discovered_resource_create() (*in module inmanta.protocol.methods_v2*), 331
discovered_resource_create_batch() (*in module inmanta.protocol.methods_v2*), 331
discovered_resources_get() (*in module inmanta.protocol.methods_v2*), 331
discovered_resources_get_batch() (*in module inmanta.protocol.methods_v2*), 331
do_changes() (*inmanta.agent.handler.CRUDHandler method*), 301
do_changes() (*inmanta.agent.handler.ResourceHandler method*), 299
do_dryrun() (*in module inmanta.protocol.methods*), 320
do_reload() (*inmanta.agent.handler.CRUDHandler method*), 301
dryrun (*inmanta.const.ResourceAction attribute*), 294
dryrun_list() (*in module inmanta.protocol.methods*), 320
dryrun_report() (*in module inmanta.protocol.methods*), 320
dryrun_request() (*in module inmanta.protocol.methods*), 320
dryrun_trigger() (*in module inmanta.protocol.methods_v2*), 332
dryrun_update() (*in module inmanta.protocol.methods*), 321
DSL, **235**
DynamicProxy (*class in inmanta.execute.proxy*), 317

E

end (*inmanta.ast.export.Range attribute*), 353
entity, **235**
Entity (*class in inmanta.model*), 116
ENVIRONMENT
    inmanta-cli-environment-delete command line option, 256
    inmanta-cli-environment-modify command line option, 256
    inmanta-cli-environment-recompile command line option, 257
    inmanta-cli-environment-save command line option, 257
    inmanta-cli-environment-show command line option, 259
environment, **235**
Environment (*class in inmanta.data*), 315
environment_clear() (*in module inmanta.protocol.methods_v2*), 332
environment_create() (*in module inmanta.protocol.methods_v2*), 332
environment_create_token() (*in module inmanta.protocol.methods_v2*), 333
environment_delete() (*in module inmanta.protocol.methods_v2*), 333
environment_get() (*in module inmanta.protocol.methods_v2*), 333
environment_list() (*in module inmanta.protocol.methods_v2*), 333
environment_modify() (*in module inmanta.protocol.methods_v2*), 333
environment_setting_delete() (*in module inmanta.protocol.methods_v2*), 334

environment_setting_get() (*in module inmanta.protocol.methods_v2*), 334
environment_settings_list() (*in module inmanta.protocol.methods_v2*), 334
environment_settings_set() (*in module inmanta.protocol.methods_v2*), 334
Error (*class in inmanta.ast.export*), 351
ERROR (*inmanta.const.LogLevel attribute*), 293
ErrorCategory (*class in inmanta.ast.export*), 352
errors (*inmanta.data.model.CompileData attribute*), 351
exec.in_shell()
    built-in function, 365
exec.PosixRun (*built-in class*), 365
exec.Run (*built-in class*), 365
exec::execHost, 365
exec::Run, 363
exec::Run.command, 364
exec::Run.creates, 364
exec::Run.cwd, 364
exec::Run.environment, 364
exec::Run.host, 364
exec::Run.onlyif, 364
exec::Run.path, 364
exec::Run.reload, 364
exec::Run.reload_only, 364
exec::Run.returns, 364
exec::Run.skip_on_fail, 364
exec::Run.timeout, 364
exec::Run.unless, 364
execute() (*inmanta.agent.handler.CRUDHandler method*), 302
execute() (*inmanta.agent.handler.ResourceHandler method*), 299
expert feature, **235**
ExplicitPluginException (*class in inmanta.ast*), 294
ExternalException (*class in inmanta.ast*), 294

F

facts, **235**
facts() (*inmanta.agent.handler.CRUDHandler method*), 302
fields_updated() (*inmanta.agent.handler.HandlerContext method*), 298
file_exists() (*inmanta.agent.io.local.LocalIO method*), 304
file_stat() (*inmanta.agent.io.local.LocalIO method*), 304
Forbidden (*class in inmanta.protocol.exceptions*), 113
from_path() (*inmanta.module.Module class method*), 308
from_path() (*inmanta.module.ModuleLike class method*), 308
from_path() (*inmanta.module.ModuleV1 class method*), 308
from_path() (*inmanta.module.ModuleV2 class method*), 309

G

get() (*inmanta.module.Project class method*), 310
get_agent_process() (*in module inmanta.protocol.methods*), 321
get_agent_process_details() (*in module inmanta.protocol.methods_v2*), 334
get_agents() (*in module inmanta.protocol.methods_v2*), 335
get_all_facts() (*in module inmanta.protocol.methods_v2*), 335
get_api_docs() (*in module inmanta.protocol.methods_v2*), 336
get_base_type() (*inmanta.ast.type.Type method*), 311
get_by_id() (*inmanta.data.BaseDocument class method*), 313
get_client() (*inmanta.agent.handler.CRUDHandler method*), 302
get_client() (*inmanta.plugins.Context method*), 294
get_code() (*in module inmanta.protocol.methods*), 321
get_compile_data() (*in module inmanta.protocol.methods_v2*), 336
get_compile_queue() (*in module inmanta.protocol.methods*), 321
get_compile_reports() (*in module inmanta.protocol.methods_v2*), 336
get_compiler() (*inmanta.plugins.Context method*), 294
get_data_dir() (*inmanta.plugins.Context method*), 294
get_depended_by() (*inmanta.server.protocol.ServerSlice method*), 106
get_dependencies() (*inmanta.server.protocol.ServerSlice method*), 106
get_diff_of_versions() (*in module inmanta.protocol.methods_v2*), 337
get_dryrun_diff() (*in module inmanta.protocol.methods_v2*), 337
get_element() (*inmanta.util.dict_path.DictPath method*), 102
get_elements() (*inmanta.util.dict_path.DictPath method*), 102
get_environment() (*in module inmanta.protocol.methods*), 321
get_environment_id() (*inmanta.plugins.Context method*), 295
get_environment_metrics() (*in module inmanta.protocol.methods_v2*), 338
get_fact() (*in module inmanta.protocol.methods_v2*), 338
get_facts() (*in module inmanta.protocol.methods_v2*), 338

get_file() (*in module inmanta.protocol.methods*), 321
get_file() (*inmanta.agent.handler.CRUDHandler method*), 302
get_functions() (*inmanta.plugins.PluginMeta class method*), 296
get_installed_module() (*inmanta.module.ModuleSource method*), 309
get_key() (*inmanta.util.dict_path.DictPath method*), 102
get_list() (*inmanta.data.BaseDocument class method*), 313
get_logs_for_version() (*inmanta.data.ResourceAction class method*), 317
get_notification() (*in module inmanta.protocol.methods_v2*), 339
get_param() (*in module inmanta.protocol.methods*), 321
get_parameter() (*in module inmanta.protocol.methods*), 322
get_parameters() (*in module inmanta.protocol.methods_v2*), 339
get_path_sections() (*inmanta.util.dict_path.DictPath method*), 102
get_pip_config() (*in module inmanta.protocol.methods_v2*), 339
get_plugin_files() (*inmanta.module.Module method*), 308
get_project() (*in module inmanta.protocol.methods*), 322
get_queue_scheduler() (*inmanta.plugins.Context method*), 295
get_report() (*in module inmanta.protocol.methods*), 322
get_reports() (*in module inmanta.protocol.methods*), 322
get_resolver() (*inmanta.plugins.Context method*), 295
get_resource() (*in module inmanta.protocol.methods*), 322
get_resource_actions() (*in module inmanta.protocol.methods_v2*), 340
get_resource_events() (*in module inmanta.protocol.methods_v2*), 340
get_resources_for_agent() (*in module inmanta.protocol.methods*), 323
get_resources_for_version() (*inmanta.data.Resource class method*), 316
get_resources_in_version() (*in module inmanta.protocol.methods_v2*), 341
get_server_status() (*in module inmanta.protocol.methods*), 323
get_setting() (*in module inmanta.protocol.methods*), 323
get_source_code() (*in module inmanta.protocol.methods_v2*), 342
get_state() (*in module inmanta.protocol.methods*), 323
get_status() (*in module inmanta.protocol.methods*), 323
get_substitute_by_id() (*inmanta.data.Compile class method*), 314
get_sync_client() (*inmanta.plugins.Context method*), 295
get_type() (*inmanta.ast.attribute.Attribute method*), 307
get_type() (*inmanta.plugins.Context method*), 295
get_version() (*in module inmanta.protocol.methods*), 323
get_versions() (*inmanta.data.ConfigurationModel class method*), 315
getfact (*inmanta.const.ResourceAction attribute*), 294

H

halt_environment() (*in module inmanta.protocol.methods_v2*), 342
handle (*class in inmanta.protocol.decorators*), 111
handler, **235**
HandlerContext (*class in inmanta.agent.handler*), 298
hash_file() (*inmanta.agent.io.local.LocalIO method*), 304
heartbeat() (*in module inmanta.protocol.methods*), 324
heartbeat_reply() (*in module inmanta.protocol.methods*), 324

I

Id (*class in inmanta.resources*), 296
ignore_env() (*in module inmanta.protocol.methods*), 324
IgnoreResourceException (*class in inmanta.resources*), 296
INFO (*inmanta.const.LogLevel attribute*), 293
infrastructure, **235**
infrastructure-as-code, **236**
init_env() (*inmanta.env.VirtualEnv method*), 311
inmanta.data.TBaseDocument (*built-in variable*), 313
inmanta.model
    module, 115
inmanta.protocol.methods
    module, 318
inmanta.protocol.methods_v2
    module, 330
inmanta_lsm.const.LSM_ENV_VARS (*built-in variable*), 294
inmanta-cli command line option
    --host, 253
    --port, 253
inmanta-cli-action-log-list command line option

--action, 253
--environment, 253
--rvid, 253
-e, 253
inmanta-cli-action-log-show-messages command line option
--action-id, 254
--environment, 254
--rvid, 254
-e, 254
inmanta-cli-agent-list command line option
--environment, 254
-e, 254
inmanta-cli-agent-pause command line option
--agent, 254
--all, 254
--environment, 254
-e, 254
inmanta-cli-agent-unpause command line option
--agent, 255
--all, 255
--environment, 255
-e, 255
inmanta-cli-environment-create command line option
--branch, 255
--name, 255
--project, 255
--repo-url, 255
--save, 255
-b, 255
-n, 255
-p, 255
-r, 255
-s, 255
inmanta-cli-environment-delete command line option
ENVIRONMENT, 256
inmanta-cli-environment-modify command line option
--branch, 256
--name, 256
--repo-url, 256
-b, 256
-n, 256
-r, 256
ENVIRONMENT, 256
inmanta-cli-environment-recompile command line option
--update, 257
-u, 257
ENVIRONMENT, 257
inmanta-cli-environment-save command line option
ENVIRONMENT, 257
inmanta-cli-environment-setting-delete command line option
--environment, 258
--key, 258
-e, 258
-k, 258
inmanta-cli-environment-setting-get command line option
--environment, 258
--key, 258
-e, 258
-k, 258
inmanta-cli-environment-setting-list command line option
--environment, 258
-e, 258
inmanta-cli-environment-setting-set command line option
--environment, 258
--key, 258
--value, 258
-e, 258
-k, 258
-o, 258
inmanta-cli-environment-show command line option
--format, 259
ENVIRONMENT, 259
inmanta-cli-lsm command line option
--host, 259
--port, 259
inmanta-cli-lsm-resources-list command line option
--environment, 260
--instance, 260
--service-entity, 260
--version, 260
-e, 260
-i, 260
-s, 260
-v, 260
inmanta-cli-lsm-service-entities-list command line option
--environment, 260
-e, 260
inmanta-cli-lsm-service-instances-list command line option
--environment, 261
--service-entity, 261
-e, 261
-s, 261
inmanta-cli-monitor command line option
--environment, 261
-e, 261
inmanta-cli-param-get command line option
--environment, 261
--name, 261
--resource, 261

-e, 261
inmanta-cli-param-list command line option
--environment, 262
-e, 262
inmanta-cli-param-set command line option
--environment, 262
--name, 262
--value, 262
-e, 262
inmanta-cli-project-create command line option
--name, 262
-n, 262
inmanta-cli-project-delete command line option
PROJECT, 263
inmanta-cli-project-modify command line option
--name, 263
-n, 263
PROJECT, 263
inmanta-cli-project-show command line option
PROJECT, 264
inmanta-cli-token-create command line option
--agent, 264
--api, 264
--compiler, 264
--environment, 264
-e, 264
inmanta-cli-version-list command line option
--environment, 265
-e, 265
inmanta-cli-version-release command line option
--environment, 265
--full, 265
--push, 265
-e, 265
-p, 265
VERSION, 265
inmanta-cli-version-report command line option
--environment, 265
--version, 265
-e, 265
-i, 265
-l, 265
InmantaBootloader (*class in inmanta.server.bootloader*), 351
install_modules() (*inmanta.module.Project method*), 310
INSTALL_OPTS (*in module inmanta.module*), 308
InstallMode (*class in inmanta.module*), 307
instance, **236**
Integer (*class in inmanta.ast.type*), 312
InvalidMetadata (*class in inmanta.module*), 308
InvalidModuleException (*class in inmanta.module*), 308
is_compiling() (*in module inmanta.protocol.methods*), 324
is_dry_run() (*inmanta.agent.handler.HandlerContext method*), 298
is_editable() (*inmanta.module.ModuleV2 method*), 309
is_primitive() (*inmanta.ast.type.Type method*), 311
is_remote() (*inmanta.agent.io.local.LocalIO method*), 305
is_symlink() (*inmanta.agent.io.local.LocalIO method*), 305

L

lifecycle, **185**
line (*inmanta.ast.export.Position attribute*), 353
List (*class in inmanta.ast.type*), 312
list_agent_processes() (*in module inmanta.protocol.methods*), 324
list_agents() (*in module inmanta.protocol.methods*), 325
list_changes() (*inmanta.agent.handler.CRUDHandler method*), 302
list_changes() (*inmanta.agent.handler.ResourceHandler method*), 299
list_desired_state_versions() (*in module inmanta.protocol.methods_v2*), 342
list_dryruns() (*in module inmanta.protocol.methods_v2*), 342
list_environments() (*in module inmanta.protocol.methods*), 325
list_notifications() (*in module inmanta.protocol.methods_v2*), 342
list_params() (*in module inmanta.protocol.methods*), 325
list_projects() (*in module inmanta.protocol.methods*), 325
list_settings() (*in module inmanta.protocol.methods*), 325
list_users() (*in module inmanta.protocol.methods_v2*), 343
list_versions() (*in module inmanta.protocol.methods*), 325
Literal (*class in inmanta.ast.type*), 312
LiteralDict (*class in inmanta.ast.type*), 312
LiteralList (*class in inmanta.ast.type*), 312
load() (*inmanta.module.Project method*), 310
LocalIO (*class in inmanta.agent.io.local*), 304
Location (*class in inmanta.ast.export*), 352
Location (*class in inmanta.model*), 116
location (*inmanta.ast.export.Error attribute*), 351
login() (*in module inmanta.protocol.methods_v2*), 343
LogLevel (*class in inmanta.const*), 293

lsm.all()
    built-in function, 372
lsm.context_v2_unwrapper()
    built-in function, 372
lsm.current_state()
    built-in function, 374
lsm.format()
    built-in function, 374
lsm.fsm_to_dot()
    built-in function, 374
lsm.has_current_state()
    built-in function, 374
lsm.is_validating()
    built-in function, 375
lsm.LifecycleTransferHandler (*built-in class*), 377
lsm.LifecycleTransferResource (*built-in class*), 377
lsm.render_dot()
    built-in function, 375
lsm.ServiceInstanceHandler (*built-in class*), 377
lsm.ServiceInstanceResource (*built-in class*), 377
lsm.update_read_only_attribute()
    built-in function, 375
lsm.validate_service_index()
    built-in function, 375
lsm::attribute_modifier, 365
lsm::attribute_set_opt, 365
lsm::EmbeddedEntity, 366
lsm::EmbeddedEntity.__annotations, 366
lsm::EmbeddedEntity.__lsm_key_attributes, 366
lsm::InterServiceRelation, 366
lsm::labels, 365
lsm::LifecycleStateMachine, 366
lsm::LifecycleStateMachine.initial_state, 366
lsm::LifecycleStateMachine.name, 366
lsm::LifecycleStateMachine.render_graph, 366
lsm::LifecycleStateMachine.transfers, 366
lsm::LifecycleTransfer, 366
lsm::LifecycleTransfer.agent, 367
lsm::LifecycleTransfer.instance_id, 367
lsm::LifecycleTransfer.next_version, 367
lsm::LifecycleTransfer.purge_on_delete, 367
lsm::LifecycleTransfer.resources, 367
lsm::LifecycleTransfer.service_entity, 366
lsm::operation, 366
lsm::RelationAnnotations, 367
lsm::RelationAnnotations.annotations, 367
lsm::RelationshipMetadata, 367
lsm::RelationshipMetadata.modifier, 367
lsm::ServiceBase, 367
lsm::serviceBase, 372
lsm::ServiceBase.children, 367
lsm::ServiceBase.owned_resources, 368
lsm::ServiceBase.parent, 367
lsm::ServiceBase.purge_resources, 367
lsm::ServiceBase.resources, 368
lsm::ServiceEntity, 368
lsm::ServiceEntity.__annotations, 368
lsm::ServiceEntity.current_state, 368
lsm::ServiceEntity.entity_binding, 368
lsm::ServiceEntity.instance_id, 368
lsm::ServiceEntityBinding, 368
lsm::ServiceEntityBinding.allocation_spec, 369
lsm::ServiceEntityBinding.lifecycle, 369
lsm::ServiceEntityBinding.owned, 369
lsm::ServiceEntityBinding.owner, 369
lsm::ServiceEntityBinding.relation_to_owner, 369
lsm::ServiceEntityBinding.service_entity, 369
lsm::ServiceEntityBinding.service_entity_name, 369
lsm::ServiceEntityBinding.service_identity, 369
lsm::ServiceEntityBinding.service_identity_display_name, 369
lsm::ServiceEntityBinding.strict_modifier_enforcement, 369
lsm::ServiceEntityBindingV2, 369
lsm::ServiceEntityBindingV2.strict_modifier_enforcement, 369
lsm::ServiceInstance, 369
lsm::ServiceInstance.agent, 370
lsm::ServiceInstance.attributes, 370
lsm::ServiceInstance.instance_id, 370
lsm::ServiceInstance.purge_on_delete, 370
lsm::ServiceInstance.rejected_states, 370
lsm::ServiceInstance.service_entity, 369
lsm::ServiceInstance.skip_update_states, 370
lsm::setResourceSet, 372
lsm::State, 370
lsm::State.deleted, 370
lsm::State.export_resources, 370
lsm::State.label, 370
lsm::State.name, 370
lsm::State.purge_resources, 370
lsm::State.validate_others, 370
lsm::State.validate_self, 370
lsm::State.values, 370
lsm::stateConfig, 372
lsm::StateTransfer, 370
lsm::StateTransfer.api_set_state, 371
lsm::StateTransfer.auto, 371
lsm::StateTransfer.config_name, 371
lsm::StateTransfer.description, 371
lsm::StateTransfer.error, 371
lsm::StateTransfer.error_operation, 371
lsm::StateTransfer.on_delete, 371
lsm::StateTransfer.on_update, 371
lsm::StateTransfer.resource_based, 371

lsm::StateTransfer.source, 371
lsm::StateTransfer.target, 371
lsm::StateTransfer.target_operation, 371
lsm::StateTransfer.validate, 371

M

main.cf, **236**
ManagedResource (*class in inmanta.resources*), 296
master (*inmanta.module.InstallMode attribute*), 307
message (*inmanta.ast.export.Error attribute*), 351
metadata (*inmanta.module.ModuleLike property*), 308
method() (*inmanta.protocol.decorators method*), 110
mkdir() (*inmanta.agent.io.local.LocalIO method*), 305
mock_process_env() (*in module inmanta.env*), 310
model_computed_fields (*inmanta.ast.export.Error attribute*), 351
model_computed_fields (*inmanta.ast.export.Location attribute*), 352
model_computed_fields (*inmanta.ast.export.Position attribute*), 353
model_computed_fields (*inmanta.ast.export.Range attribute*), 353
model_computed_fields (*inmanta.data.model.CompileData attribute*), 351
model_config (*inmanta.ast.export.Error attribute*), 352
model_config (*inmanta.ast.export.Location attribute*), 352
model_config (*inmanta.ast.export.Position attribute*), 353
model_config (*inmanta.ast.export.Range attribute*), 353
model_config (*inmanta.data.model.CompileData attribute*), 351
model_fields (*inmanta.ast.export.Error attribute*), 352
model_fields (*inmanta.ast.export.Location attribute*), 352
model_fields (*inmanta.ast.export.Position attribute*), 353
model_fields (*inmanta.ast.export.Range attribute*), 353
model_fields (*inmanta.data.model.CompileData attribute*), 351
modify_environment() (*in module inmanta.protocol.methods*), 326
modify_project() (*in module inmanta.protocol.methods*), 326
module, **236**
    inmanta.model, 115
    inmanta.protocol.methods, 318
    inmanta.protocol.methods_v2, 330
Module (*class in inmanta.module*), 308
ModuleLike (*class in inmanta.module*), 308
ModuleMetadata (*class in inmanta.module*), 292
ModuleName (*in module inmanta.module*), 308
ModuleSource (*class in inmanta.module*), 309
ModuleV1 (*class in inmanta.module*), 308
ModuleV2 (*class in inmanta.module*), 309
ModuleV2Source (*class in inmanta.module*), 309

N

name (*inmanta.ast.variables.Reference attribute*), 311
NotFound (*class in inmanta.protocol.exceptions*), 113
notify_change() (*in module inmanta.protocol.methods*), 326
notify_change_get() (*in module inmanta.protocol.methods*), 326
NullableType (*class in inmanta.ast.type*), 311
Number (*class in inmanta.ast.type*), 312

O

openstack.EndPoint (*built-in class*), 390
openstack.EndpointHandler (*built-in class*), 394
openstack.find_flavor()
    built-in function, 390
openstack.find_image()
    built-in function, 390
openstack.Flavor (*built-in class*), 390
openstack.FlavorHandler (*built-in class*), 393
openstack.FloatingIP (*built-in class*), 391
openstack.FloatingIPHandler (*built-in class*), 394
openstack.HostPort (*built-in class*), 391
openstack.HostPortHandler (*built-in class*), 394
openstack.Image (*built-in class*), 391
openstack.ImageHandler (*built-in class*), 393
openstack.Network (*built-in class*), 391
openstack.NetworkHandler (*built-in class*), 393
openstack.Project (*built-in class*), 391
openstack.ProjectHandler (*built-in class*), 394
openstack.Role (*built-in class*), 392
openstack.RoleHandler (*built-in class*), 394
openstack.Router (*built-in class*), 392
openstack.RouterHandler (*built-in class*), 393
openstack.RouterPort (*built-in class*), 392
openstack.RouterPortHandler (*built-in class*), 393
openstack.SecurityGroup (*built-in class*), 392
openstack.SecurityGroupHandler (*built-in class*), 394
openstack.Service (*built-in class*), 392
openstack.ServiceHandler (*built-in class*), 394
openstack.Subnet (*built-in class*), 392
openstack.SubnetHandler (*built-in class*), 393
openstack.User (*built-in class*), 393
openstack.UserHandler (*built-in class*), 394
openstack.VirtualMachine (*built-in class*), 393
openstack.VirtualMachineHandler (*built-in class*), 393
openstack::AddressPair, 378
openstack::AddressPair.address, 378
openstack::AddressPair.mac, 378
openstack::admin_state, 378
openstack::agentConfig, 390
openstack::container_format, 378

openstack::direction, 378
openstack::disk_format, 378
openstack::EndPoint, 378
openstack::endPoint, 390
openstack::EndPoint.admin_url, 379
openstack::EndPoint.internal_url, 378
openstack::EndPoint.provider, 379
openstack::EndPoint.public_url, 378
openstack::EndPoint.region, 378
openstack::EndPoint.service, 379
openstack::EndPoint.service_id, 379
openstack::eth0Port, 390
openstack::fipAddr, 390
openstack::fipName, 390
openstack::Flavor, 379
openstack::Flavor.disk, 379
openstack::Flavor.ephemeral, 379
openstack::Flavor.extra_specs, 379
openstack::Flavor.flavor_id, 379
openstack::Flavor.is_public, 379
openstack::Flavor.name, 379
openstack::Flavor.provider, 379
openstack::Flavor.ram, 379
openstack::Flavor.rxtx_factor, 379
openstack::Flavor.swap, 379
openstack::Flavor.vcpus, 379
openstack::FloatingIP, 379
openstack::FloatingIP.address, 380
openstack::FloatingIP.external_network, 380
openstack::FloatingIP.force_ip, 380
openstack::FloatingIP.name, 379
openstack::FloatingIP.port, 380
openstack::FloatingIP.project, 380
openstack::FloatingIP.provider, 380
openstack::GroupRule, 380
openstack::GroupRule.remote_group, 380
openstack::Host, 380
openstack::Host.key_pair, 380
openstack::Host.project, 380
openstack::Host.provider, 380
openstack::Host.purge_on_delete, 380
openstack::Host.purged, 380
openstack::Host.security_groups, 380
openstack::Host.subnet, 380
openstack::Host.vm, 380
openstack::HostPort, 381
openstack::HostPort.dhcp, 381
openstack::HostPort.floating_ips, 381
openstack::HostPort.name, 381
openstack::HostPort.port_index, 381
openstack::HostPort.portsecurity, 381
openstack::HostPort.retries, 381
openstack::HostPort.subnet, 381
openstack::HostPort.vm, 381
openstack::HostPort.wait, 381
openstack::Image, 381
openstack::Image.container_format, 381
openstack::Image.disk_format, 382
openstack::Image.image_id, 382
openstack::Image.metadata, 382
openstack::Image.name, 381
openstack::Image.protected, 382
openstack::Image.provider, 382
openstack::Image.purge_on_delete, 382
openstack::Image.skip_on_deploy, 382
openstack::Image.uri, 381
openstack::Image.visibility, 382
openstack::IPrule, 381
openstack::IPrule.remote_prefix, 381
openstack::mac_addr, 378
openstack::Network, 382
openstack::Network.external, 382
openstack::Network.floating_ips, 382
openstack::Network.name, 382
openstack::Network.network_type, 382
openstack::Network.physical_network, 382
openstack::Network.project, 382
openstack::Network.provider, 382
openstack::Network.routers, 382
openstack::Network.segmentation_id, 382
openstack::Network.shared, 382
openstack::Network.subnets, 382
openstack::Network.vlan_transparent, 382
openstack::OpenStackResource, 383
openstack::OpenStackResource.send_event, 383
openstack::openstackVM, 390
openstack::Port, 383
openstack::Port.address, 383
openstack::Port.allowed_address_pairs, 383
openstack::Port.project, 383
openstack::Port.provider, 383
openstack::Project, 383
openstack::Project.description, 383
openstack::Project.enabled, 383
openstack::Project.floating_ips, 384
openstack::Project.name, 383
openstack::Project.networks, 383
openstack::Project.ports, 383
openstack::Project.provider, 383
openstack::Project.roles, 383
openstack::Project.routers, 383
openstack::Project.security_groups, 383
openstack::Project.subnets, 383
openstack::protocol, 378
openstack::Provider, 384
openstack::Provider.admin_url, 384
openstack::Provider.auto_agent, 384
openstack::Provider.connection_url, 384
openstack::Provider.endpoints, 384
openstack::Provider.flavors, 385
openstack::Provider.floating_ips, 384
openstack::Provider.images, 385
openstack::Provider.name, 384
openstack::Provider.networks, 384
openstack::Provider.password, 384

openstack::Provider.ports, 384
openstack::Provider.projects, 384
openstack::Provider.roles, 384
openstack::Provider.routers, 384
openstack::Provider.security_groups, 384
openstack::Provider.services, 384
openstack::Provider.subnets, 384
openstack::Provider.tenant, 384
openstack::Provider.token, 384
openstack::Provider.username, 384
openstack::Provider.users, 384
openstack::Provider.verify_cert, 384
openstack::Provider.virtual_machines, 385
openstack::providerRequire, 390
openstack::Role, 385
openstack::Role.project, 385
openstack::Role.provider, 385
openstack::Role.role, 385
openstack::Role.role_id, 385
openstack::Role.user, 385
openstack::roleImpl, 390
openstack::Route, 385
openstack::Route.destination, 385
openstack::Route.nexthop, 385
openstack::Route.router, 385
openstack::Router, 385
openstack::Router.admin_state, 386
openstack::Router.distributed, 386
openstack::Router.ext_gateway, 386
openstack::Router.ha, 386
openstack::Router.name, 386
openstack::Router.ports, 386
openstack::Router.project, 386
openstack::Router.provider, 386
openstack::Router.routes, 386
openstack::Router.subnets, 386
openstack::RouterPort, 386
openstack::RouterPort.name, 386
openstack::RouterPort.router, 386
openstack::RouterPort.subnet, 386
openstack::SecurityGroup, 386
openstack::SecurityGroup.description, 386
openstack::SecurityGroup.manage_all, 386
openstack::SecurityGroup.name, 386
openstack::SecurityGroup.project, 387
openstack::SecurityGroup.provider, 387
openstack::SecurityGroup.remote_group_rules, 387
openstack::SecurityGroup.retries, 386
openstack::SecurityGroup.rules, 387
openstack::SecurityGroup.virtual_machines, 387
openstack::SecurityGroup.wait, 386
openstack::SecurityRule, 387
openstack::SecurityRule.direction, 387
openstack::SecurityRule.group, 387
openstack::SecurityRule.ip_protocol, 387
openstack::SecurityRule.port, 387
openstack::SecurityRule.port_max, 387
openstack::SecurityRule.port_min, 387
openstack::Service, 387
openstack::Service.description, 387
openstack::Service.endpoint, 387
openstack::Service.name, 387
openstack::Service.provider, 387
openstack::Service.type, 387
openstack::sg, 390
openstack::Subnet, 387
openstack::Subnet.allocation_end, 388
openstack::Subnet.allocation_start, 388
openstack::Subnet.dhcp, 388
openstack::Subnet.disable_gateway_ip, 388
openstack::Subnet.dns_servers, 388
openstack::Subnet.gateway_ip, 388
openstack::Subnet.host_ports, 388
openstack::Subnet.name, 388
openstack::Subnet.network, 388
openstack::Subnet.network_address, 388
openstack::Subnet.project, 388
openstack::Subnet.provider, 388
openstack::Subnet.router, 388
openstack::Subnet.routers, 388
openstack::User, 388
openstack::User.email, 388
openstack::User.enabled, 388
openstack::User.name, 388
openstack::User.password, 388
openstack::User.provider, 388
openstack::User.roles, 389
openstack::VirtualMachine, 389
openstack::VirtualMachine.eth0_port, 389
openstack::VirtualMachine.host, 389
openstack::VirtualMachine.key_pair, 389
openstack::VirtualMachine.name, 389
openstack::VirtualMachine.ports, 389
openstack::VirtualMachine.project, 389
openstack::VirtualMachine.provider, 389
openstack::VirtualMachine.security_groups, 389
openstack::visibility, 378
openstack::VMAttributes, 389
openstack::VMAttributes.config_drive, 389
openstack::VMAttributes.flavor, 389
openstack::VMAttributes.image, 389
openstack::VMAttributes.metadata, 389
openstack::VMAttributes.personality, 389
openstack::VMAttributes.user_data, 389
orchestration, **236**
other (*inmanta.const.ResourceAction attribute*), 294

P

parse_id() (*inmanta.resources.Id class method*), 296
parser (*inmanta.ast.export.ErrorCategory attribute*), 352
ParserException (*class in inmanta.parser*), 294
Path (*in module inmanta.module*), 309

PGRestore (*class in inmanta.db.util*), 317
plugin, **236**
plugin (*inmanta.ast.export.ErrorCategory attribute*), 352
plugin() (*in module inmanta.plugins*), 295
PluginException (*class in inmanta.plugins*), 295
PluginMeta (*class in inmanta.plugins*), 295
PluginModuleFinder (*class in inmanta.loader*), 309
Position (*class in inmanta.ast.export*), 353
post() (*inmanta.agent.handler.CRUDHandler method*), 302
pre() (*inmanta.agent.handler.CRUDHandler method*), 302
prerelease (*inmanta.module.InstallMode attribute*), 307
prestart() (*inmanta.server.protocol.ServerSlice method*), 106
prestop() (*inmanta.server.protocol.ServerSlice method*), 106
Primitive (*class in inmanta.ast.type*), 311
PROJECT
    inmanta-cli-project-delete command line option, 263
    inmanta-cli-project-modify command line option, 263
    inmanta-cli-project-show command line option, 264
project, **236**
Project (*class in inmanta.module*), 310
project_create() (*in module inmanta.protocol.methods_v2*), 343
project_delete() (*in module inmanta.protocol.methods_v2*), 344
project_get() (*in module inmanta.protocol.methods_v2*), 344
project_list() (*in module inmanta.protocol.methods_v2*), 344
project_modify() (*in module inmanta.protocol.methods_v2*), 344
ProjectMetadata (*class in inmanta.module*), 287
ProjectNotFoundException (*class in inmanta.module*), 310
ProjectPipConfig (*class in inmanta.module*), 288
promote_desired_state_version() (*in module inmanta.protocol.methods_v2*), 344
provider() (*in module inmanta.agent.handler*), 297
pull (*inmanta.const.ResourceAction attribute*), 294
PurgeableResource (*class in inmanta.resources*), 296
push (*inmanta.const.ResourceAction attribute*), 294
put() (*inmanta.agent.io.local.LocalIO method*), 305
put_partial() (*in module inmanta.protocol.methods_v2*), 344
put_version() (*in module inmanta.protocol.methods*), 326

R

Range (*class in inmanta.ast.export*), 352
range (*inmanta.ast.export.Location attribute*), 352
read() (*inmanta.agent.io.local.LocalIO method*), 305
read_binary() (*inmanta.agent.io.local.LocalIO method*), 305
read_resource() (*inmanta.agent.handler.CRUDHandler method*), 303
readlink() (*inmanta.agent.io.local.LocalIO method*), 305
Reference (*class in inmanta.ast.variables*), 311
ReferenceValue (*class in inmanta.model*), 116
relation, **236**
Relation (*class in inmanta.model*), 116
RelationAttribute (*class in inmanta.ast.attribute*), 307
release (*inmanta.module.InstallMode attribute*), 307
release_version() (*in module inmanta.protocol.methods*), 327
remove() (*inmanta.agent.io.local.LocalIO method*), 306
remove() (*inmanta.util.dict_path.DictPath method*), 102
Report (*class in inmanta.data*), 316
reserve_version() (*in module inmanta.protocol.methods_v2*), 345
reset() (*inmanta.loader.PluginModuleFinder class method*), 309
resource, **236**
Resource (*class in inmanta.data*), 316
Resource (*class in inmanta.resources*), 296
resource handler, **236**
resource() (*in module inmanta.resources*), 296
resource_action_update() (*in module inmanta.protocol.methods*), 327
resource_deploy_done() (*in module inmanta.protocol.methods_v2*), 345
resource_deploy_start() (*in module inmanta.protocol.methods_v2*), 345
resource_details() (*in module inmanta.protocol.methods_v2*), 346
resource_did_dependency_change() (*in module inmanta.protocol.methods_v2*), 346
resource_event() (*in module inmanta.protocol.methods*), 327
resource_history() (*in module inmanta.protocol.methods_v2*), 346
resource_list() (*in module inmanta.protocol.methods_v2*), 347
resource_logs() (*in module inmanta.protocol.methods_v2*), 348
resource_str() (*inmanta.resources.Id method*), 296
ResourceAction (*class in inmanta.const*), 293
ResourceAction (*class in inmanta.data*), 316
ResourceHandler (*class in inmanta.agent.handler*), 298
ResourceIdStr (*in module inmanta.data.model*), 317
ResourcePurged (*class in inmanta.agent.handler*), 297

resources_status() (*in module inmanta.protocol.methods_v2*), 349
ResourceVersionIdStr (*in module inmanta.data.model*), 317
rest.RESTCall (*built-in class*), 396
rest.RESTHandler (*built-in class*), 396
rest::RESTCall, 395
rest::RESTCall.agent, 395
rest::RESTCall.auth_password, 395
rest::RESTCall.auth_user, 395
rest::RESTCall.body, 395
rest::RESTCall.form_encoded, 395
rest::RESTCall.headers, 395
rest::RESTCall.method, 395
rest::RESTCall.return_codes, 395
rest::RESTCall.skip_on_fail, 395
rest::RESTCall.ssl_verify, 395
rest::RESTCall.url, 395
rest::RESTCall.url_id, 395
rest::RESTCall.validate_return, 395
rest::restCallID, 396
Result (*class in inmanta.protocol.common*), 313
result (*inmanta.protocol.common.Result property*), 313
resume_environment() (*in module inmanta.protocol.methods_v2*), 349
return_value() (*inmanta.execute.proxy.DynamicProxy class method*), 317
rmdir() (*inmanta.agent.io.local.LocalIO method*), 306
run() (*inmanta.agent.io.local.LocalIO method*), 306
run_sync() (*inmanta.agent.handler.CRUDHandler method*), 303
run_sync() (*inmanta.plugins.Context method*), 295
runtime (*inmanta.ast.export.ErrorCategory attribute*), 352
RuntimeException (*class in inmanta.ast*), 294

S

ServerError (*class in inmanta.protocol.exceptions*), 114
ServerSlice (*class in inmanta.server.protocol*), 106
service entity, **185**
service instance, **185**
set() (*inmanta.module.Project class method*), 310
set_cache() (*inmanta.agent.handler.CRUDHandler method*), 303
set_element() (*inmanta.util.dict_path.DictPath method*), 102
set_fact() (*in module inmanta.protocol.methods_v2*), 349
set_fact() (*inmanta.agent.handler.HandlerContext method*), 298
set_param() (*in module inmanta.protocol.methods*), 328
set_parameter() (*in module inmanta.protocol.methods_v2*), 349
set_parameters() (*in module inmanta.protocol.methods*), 328
set_password() (*in module inmanta.protocol.methods_v2*), 350
set_setting() (*in module inmanta.protocol.methods*), 328
set_state() (*in module inmanta.protocol.methods*), 329
set_status() (*inmanta.agent.handler.HandlerContext method*), 298
ShutdownInProgress (*class in inmanta.protocol.exceptions*), 114
SkipResource (*class in inmanta.agent.handler*), 297
ssh.get_private_key()
    built-in function, 397
ssh.get_public_key()
    built-in function, 397
ssh.get_putty_key()
    built-in function, 397
ssh::Key, 396
ssh::Key.command, 396
ssh::Key.name, 396
ssh::Key.options, 396
ssh::Key.public_key, 396
ssh::Key.ssh_users, 396
ssh::SSHUser, 396
ssh::sshUser, 397
ssh::SSHUser.group, 397
ssh::SSHUser.home_dir, 397
ssh::SSHUser.host, 397
ssh::SSHUser.ssh_keys, 397
ssh::SSHUser.user, 397
start (*inmanta.ast.export.Range attribute*), 353
start() (*inmanta.server.protocol.ServerSlice method*), 106
stat_file() (*in module inmanta.protocol.methods*), 329
stat_file() (*inmanta.agent.handler.CRUDHandler method*), 303
stat_files() (*in module inmanta.protocol.methods*), 329
state, **186**
state machine, **186**
std.add_to_ip()
    built-in function, 409
std.assert()
    built-in function, 409
std.at()
    built-in function, 409
std.attr()
    built-in function, 409
std.capitalize()
    built-in function, 409
std.contains()
    built-in function, 409
std.count()
    built-in function, 409
std.dict_get()

built-in function, 409
std.environment()
built-in function, 409
std.environment_name()
built-in function, 409
std.environment_server()
built-in function, 409
std.equals()
built-in function, 409
std.familyof()
built-in function, 409
std.file()
built-in function, 409
std.filter()
built-in function, 409
std.flatten()
built-in function, 409
std.generate_password()
built-in function, 410
std.get_env()
built-in function, 410
std.get_env_int()
built-in function, 410
std.getattr()
built-in function, 410
std.getfact()
built-in function, 410
std.hostname()
built-in function, 410
std.inlineif()
built-in function, 410
std.invert()
built-in function, 410
std.ip_address_from_interface()
built-in function, 410
std.ipindex()
built-in function, 410
std.is_base64_encoded()
built-in function, 410
std.is_instance()
built-in function, 410
std.is_unknown()
built-in function, 410
std.isset()
built-in function, 410
std.item()
built-in function, 410
std.key_sort()
built-in function, 410
std.len()
built-in function, 411
std.length()
built-in function, 411
std.limit()
built-in function, 411
std.list_files()
built-in function, 411
std.lower()
built-in function, 411
std.netmask()
built-in function, 411
std.network_address()
built-in function, 411
std.objid()
built-in function, 411
std.password()
built-in function, 411
std.prefixlen()
built-in function, 411
std.prefixlength_to_netmask()
built-in function, 411
std.print()
built-in function, 411
std.replace()
built-in function, 411
std.resources.AgentConfig (*built-in class*), 413
std.resources.AgentConfigHandler (*built-in class*), 415
std.resources.Directory (*built-in class*), 413
std.resources.DirectoryHandler (*built-in class*), 415
std.resources.File (*built-in class*), 413
std.resources.Null (*built-in class*), 414
std.resources.NullProvider (*built-in class*), 414
std.resources.Package (*built-in class*), 413
std.resources.PosixFileProvider (*built-in class*), 414
std.resources.Service (*built-in class*), 414
std.resources.ServiceService (*built-in class*), 415
std.resources.Symlink (*built-in class*), 414
std.resources.SymlinkProvider (*built-in class*), 415
std.resources.SystemdService (*built-in class*), 415
std.resources.YumPackage (*built-in class*), 414
std.select()
built-in function, 411
std.sequence()
built-in function, 411
std.server_ca()
built-in function, 412
std.server_port()
built-in function, 412
std.server_ssl()
built-in function, 412
std.server_token()
built-in function, 412
std.source()
built-in function, 412
std.split()
built-in function, 412
std.template()
built-in function, 412
std.timestamp()
built-in function, 412

std.to_number()
    built-in function, 412
std.type()
    built-in function, 412
std.unique()
    built-in function, 412
std.unique_file()
    built-in function, 412
std.upper()
    built-in function, 412
std.validate_type()
    built-in function, 412
std::AgentConfig, 400
std::agentConfig, 408
std::AgentConfig.agent, 400
std::AgentConfig.agentname, 400
std::AgentConfig.autostart, 400
std::AgentConfig.uri, 400
std::alfanum, 397
std::any_http_url, 397
std::any_url, 397
std::ascii_word, 398
std::base64, 398
std::config_agent, 398
std::ConfigFile, 401
std::ConfigFile.group, 401
std::ConfigFile.mode, 401
std::ConfigFile.owner, 401
std::Content, 401
std::Content.sorting_key, 401
std::Content.value, 401
std::date, 398
std::datetime, 398
std::DefaultDirectory, 401
std::DefaultDirectory.group, 401
std::DefaultDirectory.mode, 401
std::DefaultDirectory.owner, 401
std::Directory, 401
std::Directory.group, 401
std::Directory.host, 401
std::Directory.mode, 401
std::Directory.owner, 401
std::Directory.path, 401
std::Directory.purge_on_delete, 401
std::dirHost, 408
std::DiscoveryResource, 402
std::email_str, 398
std::Entity, 402
std::Entity.provides, 402
std::Entity.requires, 402
std::File, 402
std::File.content, 402
std::File.content_seperator, 402
std::File.group, 402
std::File.host, 402
std::File.mode, 402
std::File.owner, 402
std::File.path, 402
std::File.prefix_content, 402
std::File.purge_on_delete, 402
std::File.send_event, 402
std::File.suffix_content, 402
std::fileHost, 408
std::Host, 403
std::Host.directories, 403
std::Host.files, 403
std::Host.host_config, 403
std::Host.host_groups, 403
std::Host.ip, 403
std::Host.os, 403
std::Host.packages, 403
std::Host.remote_agent, 403
std::Host.remote_port, 403
std::Host.remote_user, 403
std::Host.repository, 403
std::Host.services, 403
std::Host.symlinks, 403
std::HostConfig, 403
std::HostConfig.host, 403
std::hostDefaults, 408
std::HostGroup, 404
std::HostGroup.hosts, 404
std::HostGroup.name, 404
std::hoststring, 398
std::http_url, 398
std::ipv4_address, 398
std::ipv4_interface, 398
std::ipv4_network, 398
std::ipv6_address, 398
std::ipv6_interface, 399
std::ipv6_network, 399
std::ipv_any_address, 399
std::ipv_any_interface, 399
std::ipv_any_network, 399
std::ManagedDevice, 404
std::ManagedDevice.name, 404
std::ManagedResource, 404
std::ManagedResource.managed, 404
std::MutableBool, 404
std::MutableBool.value, 404
std::MutableNumber, 404
std::MutableNumber.value, 405
std::MutableString, 405
std::MutableString.value, 405
std::name_email, 399
std::negative_float, 399
std::negative_int, 399
std::non_empty_string, 399
std::none, 408
std::OS, 405
std::OS.family, 406
std::OS.member, 406
std::OS.name, 406
std::OS.python_cmd, 406
std::OS.version, 406
std::Package, 406

std::Package.host, 406
std::Package.name, 406
std::Package.state, 406
std::package_state, 399
std::Packages, 406
std::Packages.host, 406
std::Packages.name, 406
std::Packages.state, 406
std::pkgHost, 409
std::pkgs, 409
std::port, 399
std::positive_float, 399
std::positive_int, 400
std::printable_ascii, 400
std::PurgeableResource, 406
std::PurgeableResource.purge_on_delete, 407
std::PurgeableResource.purged, 407
std::Reload, 407
std::reload, 409
std::Reload.reload, 407
std::Reload.send_event, 407
std::Resource, 407
std::Resource.send_event, 407
std::ResourceSet, 407
std::ResourceSet.name, 407
std::ResourceSet.resources, 407
std::Service, 407
std::Service.host, 407
std::Service.name, 407
std::Service.onboot, 407
std::Service.state, 407
std::service_state, 400
std::serviceHost, 409
std::symHost, 409
std::Symlink, 408
std::Symlink.host, 408
std::Symlink.purge_on_delete, 408
std::Symlink.send_event, 408
std::Symlink.source, 408
std::Symlink.target, 408
std::testing::NullResource, 408
std::testing::NullResource.agentname, 408
std::testing::NullResource.fail, 408
std::testing::NullResource.name, 408
std::testing::NullResource.send_event, 408
std::time, 400
std::uuid, 400
stop() (*inmanta.server.protocol.ServerSlice method*), 106
store (*inmanta.const.ResourceAction attribute*), 294
String (*class in inmanta.ast.type*), 312
symlink() (*inmanta.agent.io.local.LocalIO method*), 306

T

TableNotFound (*class in inmanta.data.schema*), 114
to_dict() (*inmanta.model.Attribute method*), 115
to_dict() (*inmanta.model.DirectValue method*), 116
to_dict() (*inmanta.model.Entity method*), 116
to_dict() (*inmanta.model.Location method*), 116
to_dict() (*inmanta.model.ReferenceValue method*), 116
to_dict() (*inmanta.model.Relation method*), 117
to_dto() (*inmanta.data.Compile method*), 315
to_int (*inmanta.const.LogLevel property*), 293
to_path() (*in module inmanta.util.dict_path*), 103
to_wild_path() (*in module inmanta.util.dict_path*), 103
TRACE (*inmanta.const.LogLevel attribute*), 293
transfer, **186**
trigger, **186**
trigger() (*in module inmanta.protocol.methods*), 329
trigger_agent() (*in module inmanta.protocol.methods*), 329
Type (*class in inmanta.ast.type*), 311
type (*inmanta.ast.attribute.Attribute property*), 307
type (*inmanta.ast.export.Error attribute*), 352
type_string() (*inmanta.ast.type.Type method*), 311
TypedDict (*class in inmanta.ast.type*), 312
TypedList (*class in inmanta.ast.type*), 312
TYPES (*in module inmanta.ast.type*), 313

U

ubuntu.UbuntuService (*built-in class*), 415
UnauthorizedException (*class in inmanta.protocol.exceptions*), 113
Union (*class in inmanta.ast.type*), 312
unknown, **237**
Unknown (*class in inmanta.execute.util*), 297
unload() (*inmanta.module.Module method*), 308
unload_inmanta_plugins() (*in module inmanta.loader*), 309
unwrap() (*inmanta.execute.proxy.DynamicProxy class method*), 317
update_agent_map() (*in module inmanta.protocol.methods_v2*), 350
update_changes() (*inmanta.agent.handler.HandlerContext method*), 298
update_notification() (*in module inmanta.protocol.methods_v2*), 350
update_resource() (*inmanta.agent.handler.CRUDHandler method*), 303
upload_code_batched() (*in module inmanta.protocol.methods*), 329
upload_file() (*in module inmanta.protocol.methods*), 330
upload_file() (*inmanta.agent.handler.CRUDHandler method*), 304
uri (*inmanta.ast.export.Location attribute*), 352
use_virtual_env() (*inmanta.env.VirtualEnv method*), 311
user::execGroup, 416
user::execUser, 416
user::Group, 416

user::Group.host, 416
user::Group.name, 416
user::Group.system, 416
user::User, 416
user::User.group, 416
user::User.groups, 416
user::User.homedir, 416
user::User.host, 416
user::User.name, 416
user::User.shell, 416
user::User.system, 416

V

validate() (*inmanta.ast.attribute.Attribute method*), 307
validate() (*inmanta.ast.type.Type method*), 311
Value (*class in inmanta.model*), 117
VERSION
    inmanta-cli-version-release command line option, 265
versioned_resource_details() (*in module inmanta.protocol.methods_v2*), 350
VirtualEnv (*class in inmanta.env*), 311

W

WARNING (*inmanta.const.LogLevel attribute*), 293
with_base_type() (*inmanta.ast.type.Type method*), 311

Y

yum::redhatRepo, 417
yum::Repository, 417
yum::Repository.baseurl, 417
yum::Repository.enabled, 417
yum::Repository.gpgcheck, 417
yum::Repository.gpgkey, 417
yum::Repository.host, 417
yum::Repository.metadata_expire, 417
yum::Repository.metalink, 417
yum::Repository.mirrorlist, 417
yum::Repository.name, 417
yum::Repository.skip_if_unavailable, 417
yum::validateInput, 417