Inmanta Documentation

Release 2022.3

Inmanta NV

Sep 30, 2022

CONTENTS

Welcome to the Inmanta documentation!

Inmanta is an automation and orchestration tool to efficiently deploy and manage your software services, including all (inter)dependencies to other services and the underpinning infrastructure. It eliminates the complexity of managing large-scale, heterogeneous infrastructures and highly distributed systems.

The key characteristics of Inmanta are:

- **Integrated**: Inmanta integrates configuration management and orchestration into a single tool, taking infrastructure as code to a whole new level.
- **Powerful configuration model**: Infrastructure and application services are described using a high-level configuration model that allows the definition of (an unlimited amount of) your own entities and abstraction levels. It works from a single source, which can be tested, versioned, evolved and reused.
- **Dependency management**: Inmanta's configuration model describes all the relations between and dependencies to other services, packages, underpinning platforms and infrastructure services. This enables efficient deployment as well as provides an holistic view on your applications, environments and infrastructure.
- **End-to-end compliance**: The architecture of your software service drives the configuration, guaranteeing consistency across the entire stack and throughout distributed systems at any time. This compliance with the architecture can be achieved thanks to the integrated management approach and the configuration model using dependencies.

Currently, the Inmanta project is mainly developed and maintained by Inmanta nv.

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 3.9, `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
```

(continues on next page)

(continued from previous page)

```
author: Inmanta
author_email: code@inmanta.com
license: ASL 2.0
copyright: 2022 Inmanta
modulepath: libs
downloadpath: libs
repo:
- type: package
    url: https://packages.inmanta.com/public/quickstart/python/simple/
install_mode: release
requires:
```

- 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.

1.1.1 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 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.

1.1.2 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 and password is *admin*):

```
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.

1.1.3 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 gear icon on the top right of the Web Console, then click on Environment, scroll down all the way to the bottom of the page and copy the environment ID.

1.1.4 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.

1.1.5 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 = "admin"
    )
```

(continues on next page)

(continued from previous page)

```
)

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"
    )
)
```

- 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.

1.1.6 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
```

(continues on next page)

(continued from previous page)

```
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",
        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",
)

```

(continues on next page)

(continued from previous page)

```
56 leaf1_net_instance_int1 = srnetinstance::Interface(
57     parent_network_instance = leaf1_net_instance,
58     name = "ethernet-1/1.0"
59 )
60
61 # |OSPF| #
62
63 leaf1_protocols = srnetinstance::Protocols(
64     parent_network_instance = leaf1_net_instance,
65     ospf = leaf1_ospf
66 )
67
68 leaf1_ospf_instance = srospf::Instance(
69         parent_ospf = leaf1_ospf,
70         name = "1",
71         router_id = "10.20.30.210",
72         admin_state = "enable",
73         version = "ospf-v2"
74 )
75
76 leaf1_ospf = srprotocols::Ospf(
77     parent_protocols = leaf1_protocols,
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.

1.1.7 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 dashboard, 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.

1.1.8 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
```

1.1.9 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
```

1.2 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/.

1.3 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(
```

(continues on next page)

(continued from previous page)

```
    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.

1.3.1 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.

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
```

(continues on next page)

(continued from previous page)

```
|
|__ 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.

1.3.2 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 the latest Python 3.6 and git to be installed.

2.1.1 Install the software

RHEL 8

For RHEL 8 based systems use dnf:

```
sudo tee /etc/yum.repos.d/inmanta-oss-stable.repo <<EOF
[inmanta-oss-stable]
name=inmanta-oss-stable
baseurl=https://packages.inmanta.com/public/oss-stable/rpm/el/\$releasever/\$basearch
repo_gpgcheck=1
enabled=1
gpgkey=https://packages.inmanta.com/public/oss-stable/gpg.A34DD0A274F07713.key
gpgcheck=1
sslverify=1
sslcacert=/etc/pki/tls/certs/ca-bundle.crt
metadata_expire=300
pkg_gpgcheck=1
autorefresh=1
type=rpm-md
EOF

sudo dnf install -y inmanta-oss inmanta-oss-server inmanta-oss-agent
```

The first package (inmanta-oss) contains all the code and the commands. The server and the agent packages install config files and systemd unit files. The web-console is installed with the server package.

Debian, Ubuntu and derivatives.

First make sure Python >= 3.9 and git are installed. Inmanta requires many dependencies so it is recommended to create a virtual env. Next install inmanta with pip install in the newly created virtual env.

Please note, the path to the virtual env is arbitrary. Your desired path can override below example.

```
# Install GCC, python3 >= 3.9 and pip
sudo apt-get update
```

(continues on next page)

(continued from previous page)

```
sudo apt-get install build-essential
sudo apt-get install python3-pip

# Install wheel and inmanta in a python venv
sudo apt-get install python3-venv
sudo python3 -m venv /opt/inmanta
sudo /opt/inmanta/bin/pip install -U pip wheel
sudo /opt/inmanta/bin/pip install inmanta
sudo /opt/inmanta/bin/inmanta --help

# Install PostgreSQL
sudo apt-get install postgresql postgresql-client
```

Download the configuration files named `inmanta.cfg` and `extensions.cfg` (these names are arbitrary) in your virtual env:

```
sudo mkdir /opt/inmanta/inmanta.d
sudo apt-get install wget
sudo wget -O /opt/inmanta/inmanta.cfg "https://raw.githubusercontent.com/inmanta/inmanta-
↪core/master/misc/inmanta.cfg"
sudo wget -O /opt/inmanta/inmanta.d/extensions.cfg "https://raw.githubusercontent.com/
↪inmanta/inmanta-core/master/misc/extensions.cfg"
```

If you want to use the web-console you need to install it as well:

Get the pre-built package from our **`web-console github page<https://github.com/inmanta/web-console/packages/>`_**. Click on the the package name to go to the package's main page, then on the right hand side under `Assets`, you will see the compressed package. Download and extract it to your desired directory (preferably, on the same virtual env which was created earlier, in this case, /opt/inmanta). Next, open the `inmanta.cfg` file and at the bottom of the file, under the `[web-console]` section, change the `path` value to the `dist` directory of where you extracted the pre-built package. For instance:

```
path=/opt/inmanta/web-console/package/dist
```

Then the Inmanta server can be started using below command (please note, below command has to be run after completing the *Configure server*) part:

```
sudo /opt/inmanta/bin/inmanta -vv -c /opt/inmanta/inmanta.cfg --config-dir /opt/inmanta/
↪inmanta.d server
```

Other

First make sure Python >= 3.9 and git are installed. Inmanta requires many dependencies so it is recommended to create a virtual env. Next install inmanta with `pip install` in the newly created virtual env.

Please note, the path to the virtual env is arbitrary. Your desired path can override below example.

```
# Install python3 >= 3.9 and git
# If git is not already installed, by running git in your terminal, the installation␣
↪guide will be shown
sudo python3 -m venv /opt/inmanta
sudo /opt/inmanta/bin/pip install -U pip wheel
sudo /opt/inmanta/bin/pip install inmanta
sudo /opt/inmanta/bin/inmanta --help
```

Install PostgreSQL using this guide

Download the configuration files named `inmanta.cfg` and `extensions.cfg` (these names are arbitrary) in your virtual env:

```
sudo mkdir /opt/inmanta/inmanta.d
sudo wget -O /opt/inmanta/inmanta.cfg "https://raw.githubusercontent.com/inmanta/inmanta-
↪core/master/misc/inmanta.cfg"
sudo wget -O /opt/inmanta/inmanta.d/extensions.cfg "https://raw.githubusercontent.com/
↪inmanta/inmanta-core/master/misc/extensions.cfg"
```

If you want to use the web-console you need to install it as well:

Get the pre-built package from our **`web-console github page<https://github.com/inmanta/web-console/packages/>`_**. Click on the the package name to go to the package's main page, then on the right hand side under `Assets`, you will see the compressed package. Download and extract it to your desired directory (preferably, on the same virtual env which was created earlier, in this case, /opt/inmanta). Next, open the `inmanta.cfg` file and at the bottom of the file, under the `[web-console]` section, change the `path` value to the `dist` directory of where you extracted the pre-built package. For instance:

```
path=/opt/inmanta/web-console/package/dist
```

Then the Inmanta server can be started using below command (please note, below command has to be run after completing the *Configure server*) part:

```
sudo /opt/inmanta/bin/inmanta -vv -c /opt/inmanta/inmanta.cfg --config-dir /opt/inmanta/
↪inmanta.d server
```

Windows

On Windows only the compile and export commands are supported. This is useful in the *Push to server* deployment mode of inmanta. First make sure you have Python >= 3.9 and git. Inmanta requires many dependencies so it is recommended to create a virtual env. Next install inmanta with pip install in the newly created virtual env.

```
# Install python3 >= 3.9 and git
python3 -m venv C:\inmanta\env
C:\inmanta\env\Script\pip install inmanta
C:\inmanta\env\Script\inmanta --help
```

Source

Get the source either from our release page on github or clone/download a branch directly.

```
git clone https://github.com/inmanta/inmanta-core.git
cd inmanta
pip install -c requirements.txt .
```

Warning: When you use Inmanta modules that depend on python libraries with native code, python headers and a working compiler are required as well.

2.1.2 Configure server

This guide goes through the steps to set up an Inmanta service orchestrator server. This guide assumes a RHEL 8 based server is used. The rpm packages install the server configuration file in */etc/inmanta/inmanta.cfg*.

Optional step 1: Setup SSL and authentication

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

Step 2: Install PostgreSQL 13

PostgreSQL 13 can be installed by following the installation guide for your platform.

For RHEL8-based systems:

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

Step 3: Setup a PostgreSQL database for the Inmanta server

Initialize the PostgreSQL server:

```
sudo /usr/bin/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 sh -c "createuser --pwprompt inmanta; 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
```

Restart the PostgreSQL server to apply the changes made in the `pg_hba.conf` file:

```
sudo systemctl restart postgresql
```

Step 4: 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 5: 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 6: 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 if 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 7: 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 8: Start the Inmanta server

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

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

The server dashboard is now available on the port and host configured in step 7.

Optional Step 9: 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 10: Configure logging

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

2.2 Manage 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
```

Currently the following features are available:

- core.server::dashboard

An example feature file is:

```
slices:
    core.server:
        dashboard: false
```

2.3 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.

2.3.1 Pull the image

Use docker pull to get the desired image:

```
docker pull ghcr.io/inmanta/orchestrator:2022
```

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

2.3.2 Start the server with docker-compose

Here is a minimalist 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
        networks:
            inm_net:
                ipv4_address: 172.30.0.2

    inmanta-server:
        container_name: inmanta_orchestrator
        image: ghcr.io/inmanta/orchestrator:2022
        ports:
            - 8888:8888
        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
```

(continues on next page)

(continued from previous page)

```
            config:
                - subnet: 172.30.0.0/16
```

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 default server config included in the container images assumes that the orchestrator will have a database with hostname `inmanta_db` reachable 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.

2.3.3 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: ghcr.io/inmanta/orchestrator:2022
    ports:
        - 8888:8888
    volumes:
        - ./resources/my-server-conf.cfg:/etc/inmanta/inmanta.cfg
```

2.3.4 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.

2.3.5 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>"
```

2.3.6 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: ghcr.io/inmanta/orchestrator:2022
    ports:
        - 8888:8888
    volumes:
        - ./resources/my-server-conf.cfg:/etc/inmanta/inmanta.cfg
        - ./resources/my-env-file:/etc/inmanta/env
```

2.3.7 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
```

2.3.8 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).

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 6: 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 6: 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-oss-agent` package is required on the machine that will run the agent.

```
sudo tee /etc/yum.repos.d/inmanta-oss-stable.repo <<EOF
[inmanta-oss-stable]
name=Inmanta OSS stable
baseurl=https://packages.inmanta.com/public/oss-stable/rpm/el/\$releasever/\$basearch
gpgcheck=1
gpgkey=https://packages.inmanta.com/public/oss-stable/gpg.A34DD0A274F07713.key
repo_gpgcheck=1
enabled=1
enabled_metadata=1
EOF

sudo dnf install -y inmanta-oss-agent
```

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 *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.

Warning: The Inmanta dashboard has been deprecated in favour of the Inmanta Web Console.

CHAPTER

THREE

DASHBOARD DOCUMENTATION

Warning: The Inmanta dashboard has been deprecated in favour of the Inmanta Web Console.

3.1 The project overview

When opening the dashboard the project overview page will be the first page greeting you. It has several elements:

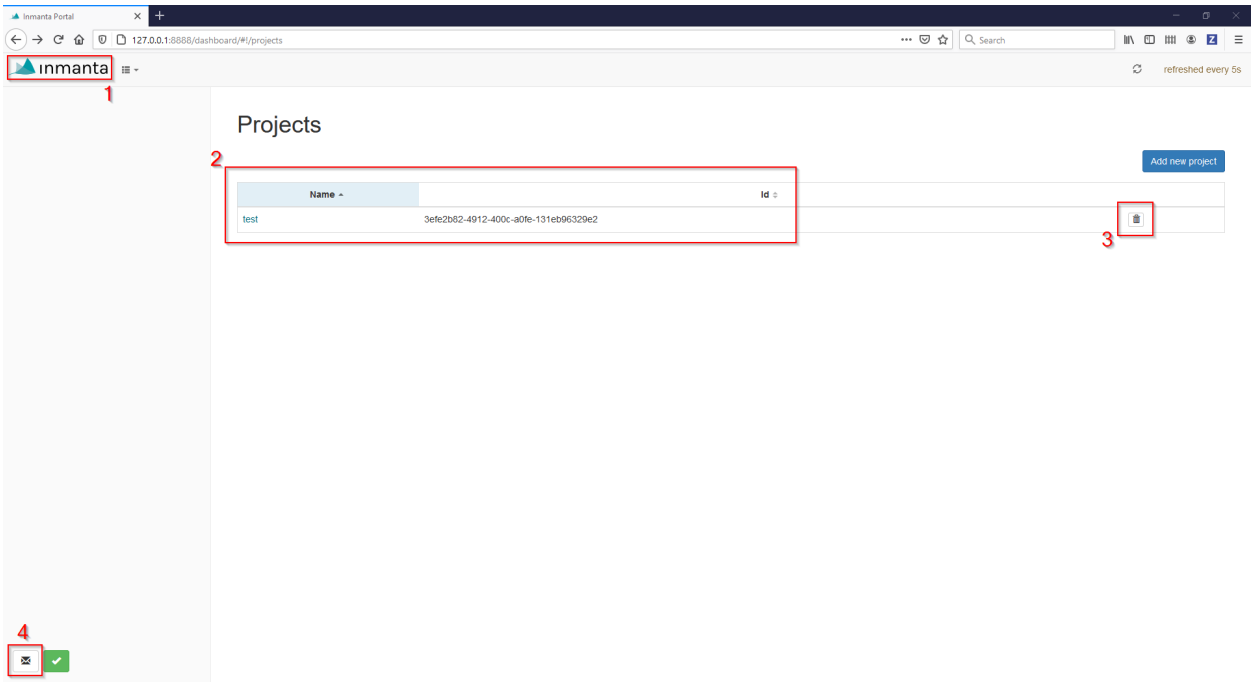


Fig. 1: The project overview page

Let's go over what some of these buttons do:

1. The "inmanta logo" button: takes you back to this page.
2. List of created projects: Lists all existing projects. For more on projects see *the glossary*. Clicking on the name of a project will take you to its page.
3. Project delete button: Deletes the project and all the environments it contains. This will delete the history and environments, but it will not purge the system of changes made or managed by the orchestrator.

4. Report an issue: If you run into any issues/bugs, this button will take you to a page where you can open a new issue.

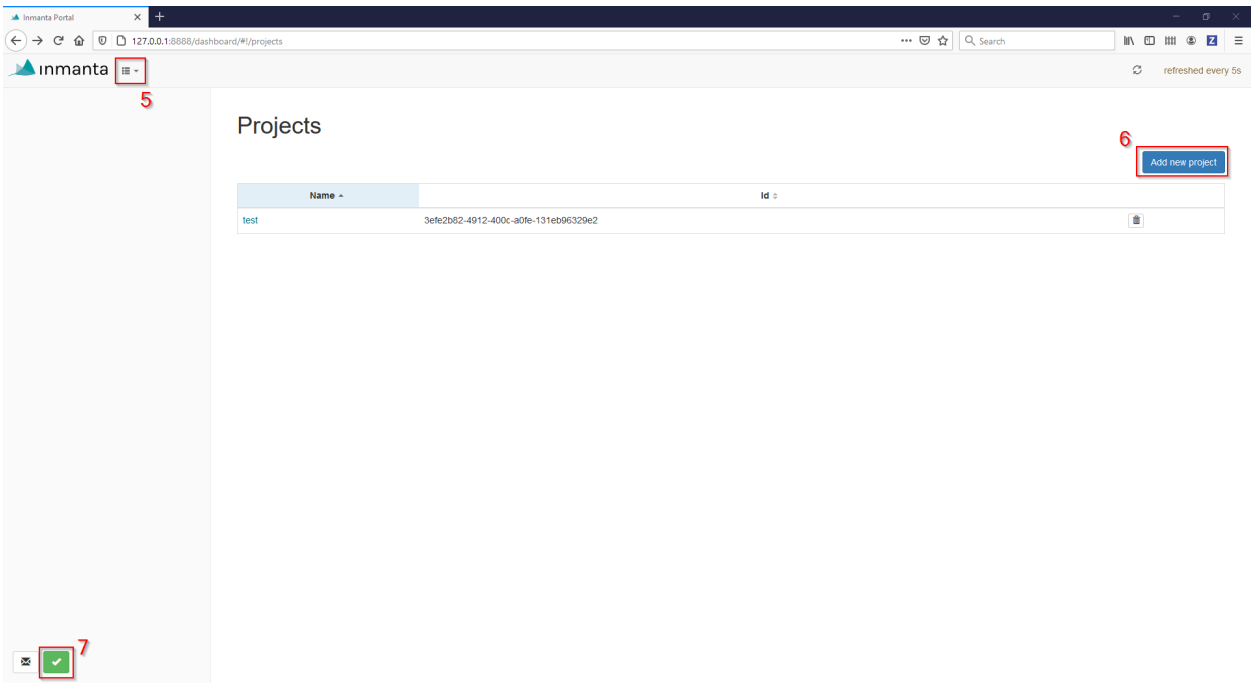


Fig. 2: The project overview page

5. Environment navigation button: Displays a list of projects and their environments. Allows navigation to any environment managed by this orchestrator, by simply clicking it's name.
6. Add new project button: This will take you through the creation of a new project and the creation of its first environment.
7. Green checkmark: This will take you to the orchestrator status page, displaying all sorts of useful information about the orchestrator instance. If the dashboard loses its connection to the server, this green checkmark will turn into a red cross.

3.2 Create a new project

Using the `Add new project` button we can create new projects:

Once `Create` is pressed, you are immediately taken to the "Create a new Environment" screen. This will help you set up your first environment. Pressing cancel will leave the project empty.

The two screenshots above are equivalent to the following inmanta-cli commands:

```
inmanta-cli project create -n dashboard-test
inmanta-cli environment create -n quickstart-env -p quickstart -r https://github.com/
↪inmanta/quickstart.git -b master
```

When in an environment, a new button at the bottom will appear:

This big red button will stop all of the orchestrator's operations for the current environment.

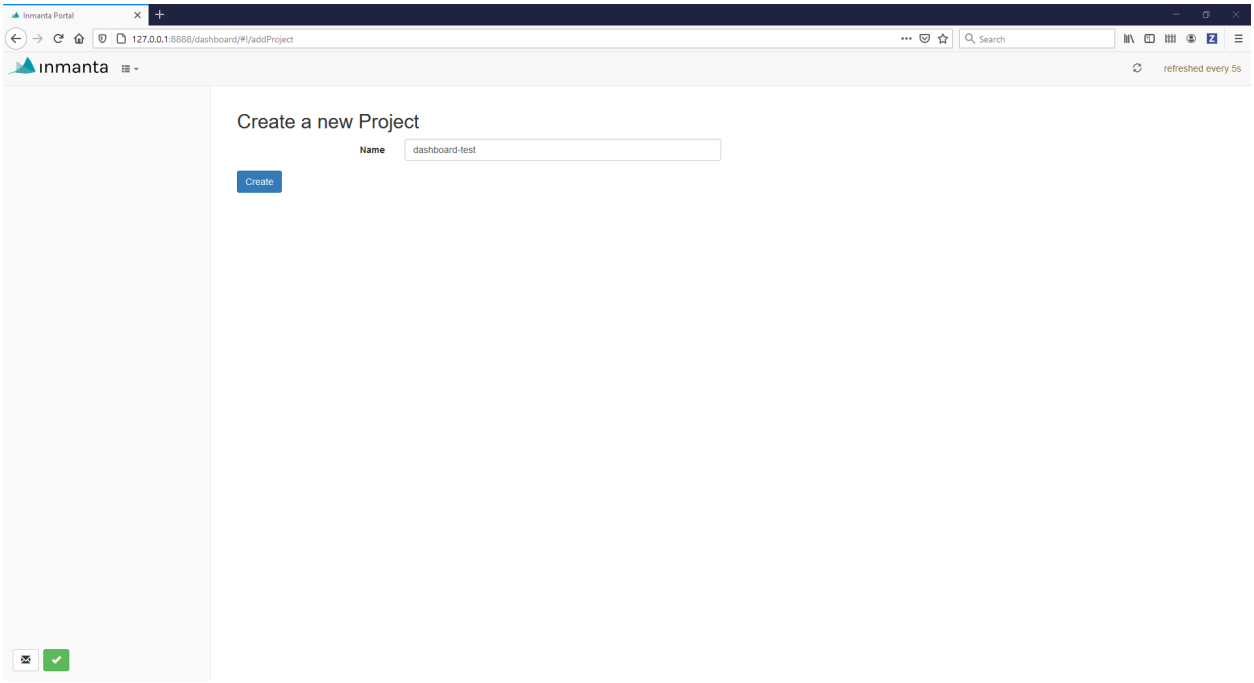


Fig. 3: Adding a new project

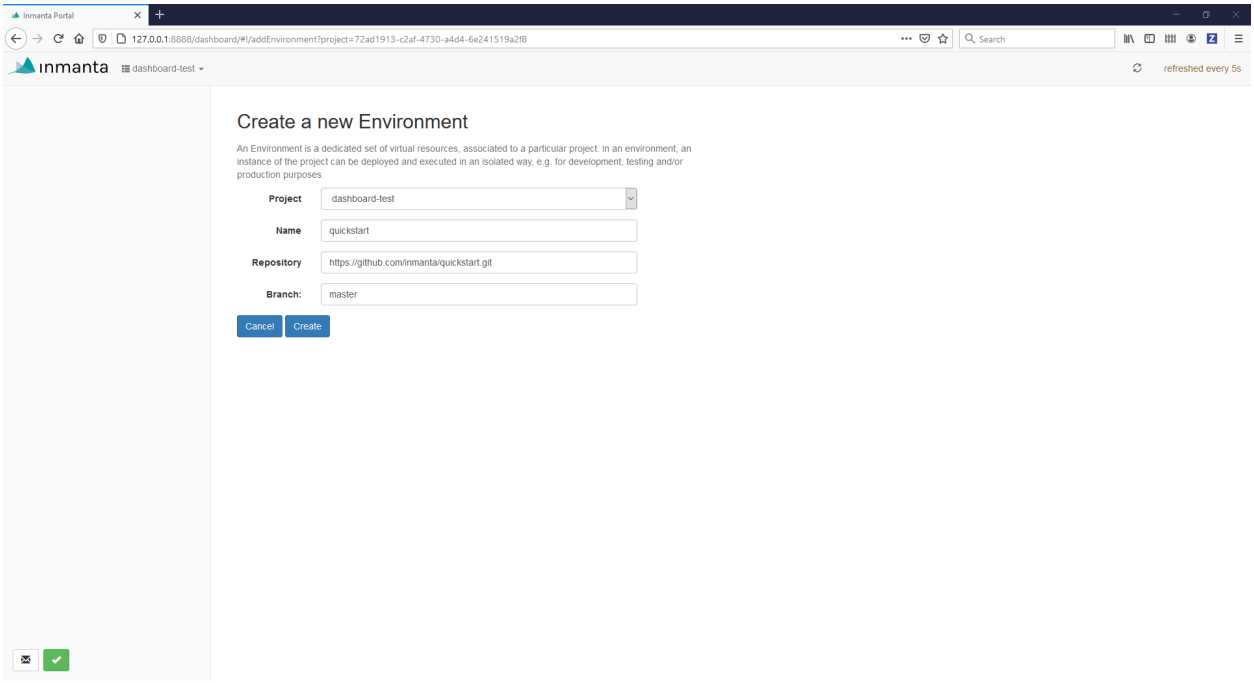


Fig. 4: Creating a new Environment

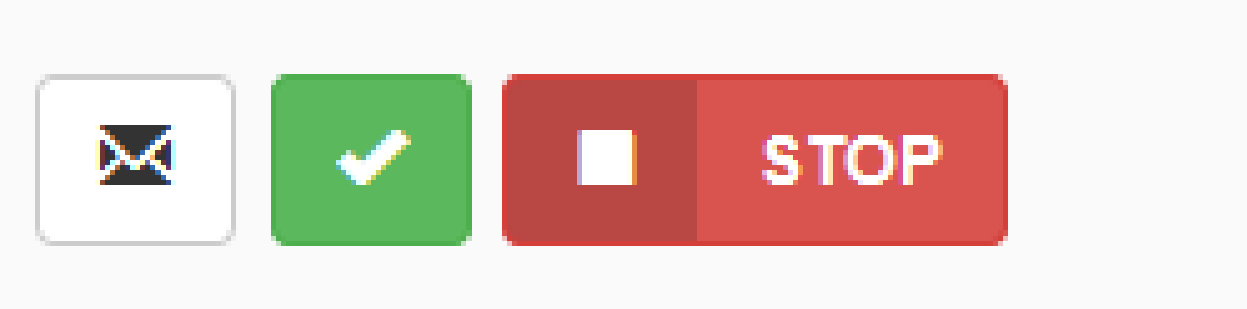


Fig. 5: The emergency stop button

3.3 The Environment Portal

Once you press the create button, you will be taken to the portal of the newly created environment:

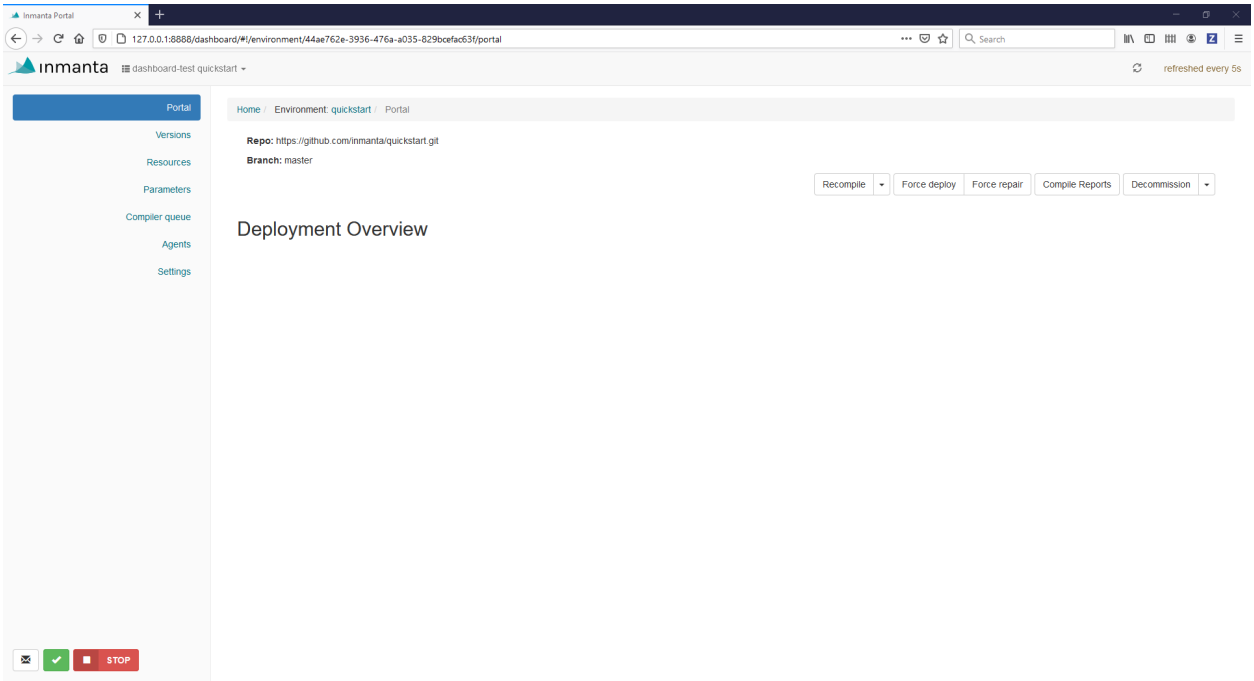


Fig. 6: A newly created environment

This environment is currently empty because the model has not been compiled yet. We can use the `Recompile` button to do this. This will clone the repository if it hadn't been already and then compile the current model. There is also an extra option for the recompile, which is `Update project & Recompile`. This will pull in any new commits and then compile the model.

Once the compile has succeeded, the orchestrator will automatically deploy the model. The deployment state is then shown in the portal.

Using the `Compile Reports` button we can diagnose problems if our compile failed.

You can click the arrow icon next to any item to expand it and see the output of the executed command.

Next, we have the `Force deploy` and `Force repair` buttons. Those are similar in function, and can be confusing to new users:

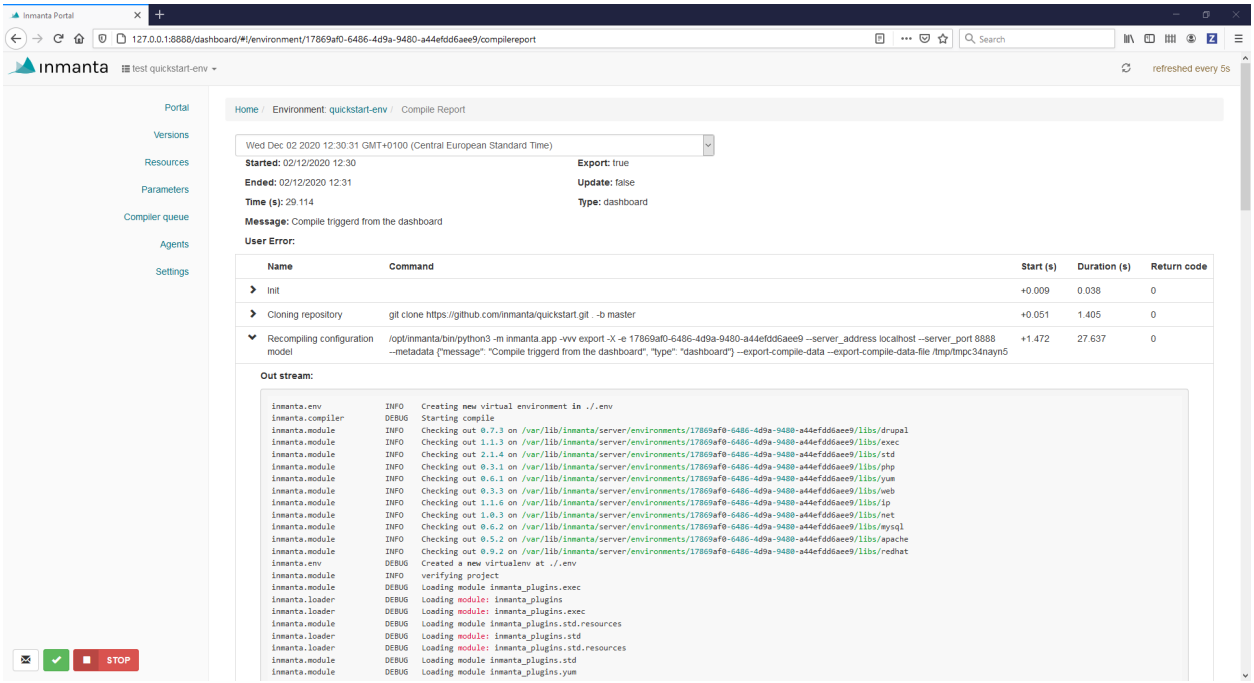


Fig. 7: A compile report

- The `Force deploy` button will go through *Every* resource and redeploy the resource.
- The `Force repair` button by contrast, will only go through resources that are currently not in a deployed state.

Finally we have the `Decommission`, `Edit`, `Clone` and `Clear` buttons, found under the `Decommission` dropdown menu:

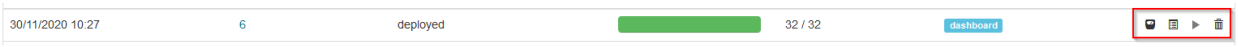
- Decommission: pushes a model that purges all resources deployed by the model.
- Edit: change the configuration of the environment, such as the git repo url or what branch to use.
- Clone: create a new environment using the same git repo and branch
- Clear: Clears the environment. This will remove all versions and compilations. It does not decommission the currently deployed model.

Note: When using `Clear` followed by a `Recompile`, the version number will be incremented as if the previous versions were still there, but these versions will no longer be present.

3.4 The Version Overview

Below the `Portal` we have the `Versions`. This will take us to an overview of all previously compiled versions of the model and their state. Do note that a version is created for every compile and this is not tied to the model being updated in git.

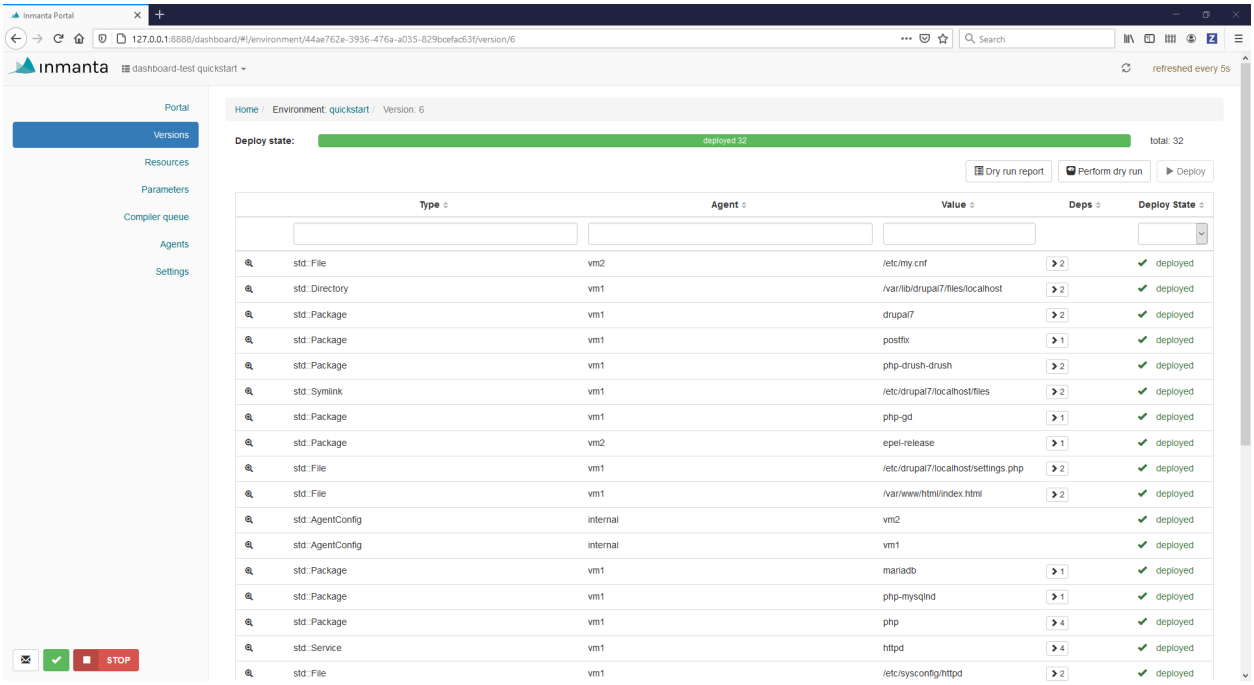
Each version has 4 buttons on the right to interact with it:



They are, in order from left to right:

- Perform dry run: the orchestrator will go through all resources in the model and compare their current state to their desired state. This can be useful to double-check what effect the deployment of a version might have on your current environment.
- Dry run report: will take you to the report of the last performed dry run, without performing a new one.
- Release version: If `auto-deploy` in the `environment settings` is set to `False`, this button can be used to deploy the model, otherwise this button will be grayed out.
- Remove version: removes the selected version from the inmanta environment.

Finally, clicking the version number will take us to the overview of that particular version. It gives the same options as the version overview does and it displays a list of all resources and their current state.



Using the filters we can filter for resources by type, by agent used to deploy the resource, by value and by deploy state. This display is continuously updated, both during deploys and after, when the orchestrator goes through all resources to make sure they remain in the desired state.

Taking a closer look at the a specific resource, there are 2 important buttons, the `Dependency` button and a `magnifying glass`. The `Dependency` button is only available if a resource depends on other resources. When pressed, it will add lines to the table displaying each dependency and it's current state:

Fig. 8: Resource dependencies

The *magnifying glass* icon will take us to an in depth overview of the resource. This will show a complete breakdown of the resource's desired state at the top and an action log at the bottom.

The desired state breakdown allows for easy inspection of the impact the resource will have. For example, the resource

in the image below will deploy a file with path */etc/my.cnf* and file permission *644*. We can even inspect the file's content.

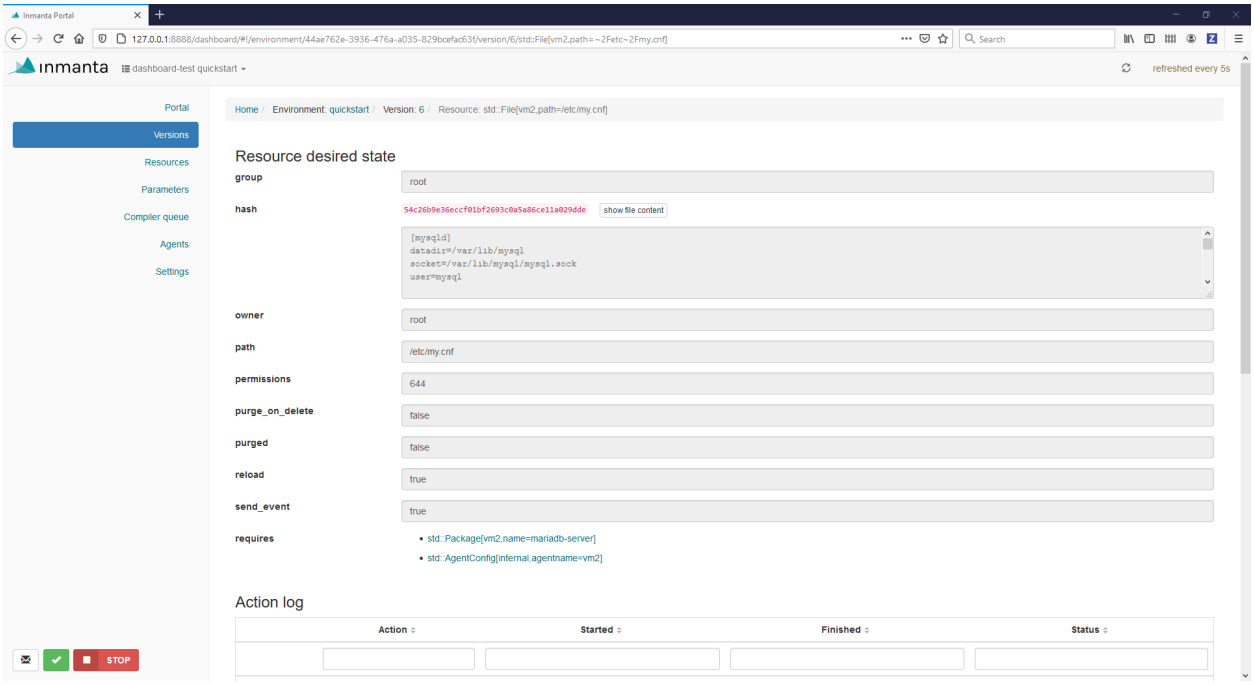


Fig. 9: Resource desired state view

The action log shows a log of actions taken on the given resource. This varies from dry-runs to deploys. This log will typically start filling up with deploys due to the orchestrator enforcing the desired state. Again we can further inspect an action by pressing the drop down arrow.

>	+1	deploy	30/11/2020 12:14:14.129	30/11/2020 12:14:14.129	✔ deployed
v	+4	deploy	30/11/2020 12:04:59.040	30/11/2020 12:04:59.087	✔ deployed
		30/11/2020 12:04:59.040	DEBUG	Start run for resource std::File[vm2,path=/etc/my.cnf],v=6 because Repair run started at 2020-11-30 12:04:50	
		30/11/2020 12:04:59.040	DEBUG	Start deploy 3270005e-7169-409b-a47e-5e55f80c935a of resource std::File[vm2,path=/etc/my.cnf],v=6	
		30/11/2020 12:04:59.061	DEBUG	Calling read_resource	
		30/11/2020 12:04:59.087	DEBUG	End run for resource std::File[vm2,path=/etc/my.cnf],v=6 in deploy 3270005e-7169-409b-a47e-5e55f80c935a	
>	+1	pull	30/11/2020 12:04:50.113	30/11/2020 12:04:50.116	

Fig. 10: Resource action view

Each of these logs can then be further analyzed by pressing the `magnifying glass`.

3.5 The Resources Overview

The resources overview, not to be confused with the similar resource version overview, gives an overview of all known resources. This is not only for resources of the currently deployed model, but potentially resources from older models and the state they are in.

While not as in depth as the resource version overview, it does link every resource to its deployed version, so the resource can be inspected there.

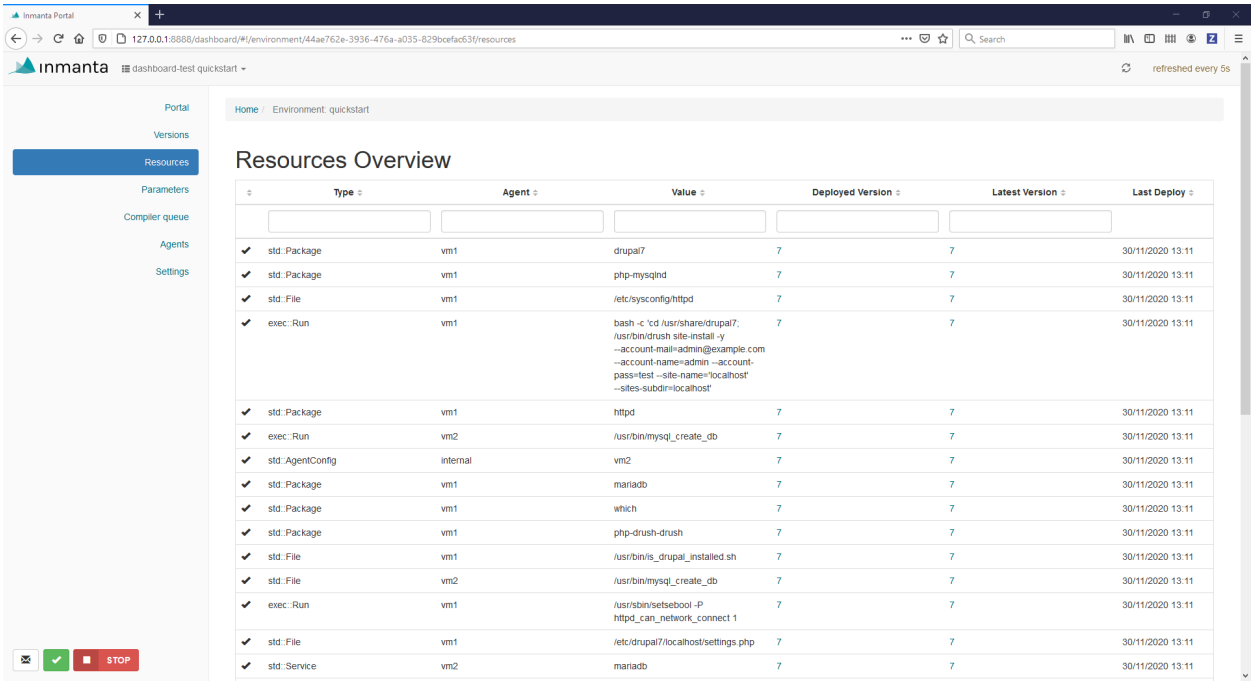


Fig. 11: The Resources Overview

3.6 The Parameters View

The parameter overview gives a list of parameters. Parameters are part of the model, but their value may or may not be known at compile time. For example, the IP address of a virtual machine that is created by the model.

Each parameter can be individually inspected or deleted. Inspecting the resource allows us to read additional metadata if any is available.

3.7 The Agent Overview

The agent overview shows different agents and the state they are in.

This overview allows us to `Force deploy` and `Force repair` resources on a per agent basis. Pausing an agent stops deployments for that agent. Useful when, for example, diagnosing problems on the machine the agent deploys to, without having to stop enforcement of the whole model.

The Agent Processes overview, lists the different processes running agents. Clicking on the `magnifying glass` allows us to inspect each process in more detail:

Here we can find the process `pid`, the ip addresses the server has bound to and what version of Python inmanta is running on, amongst other things.

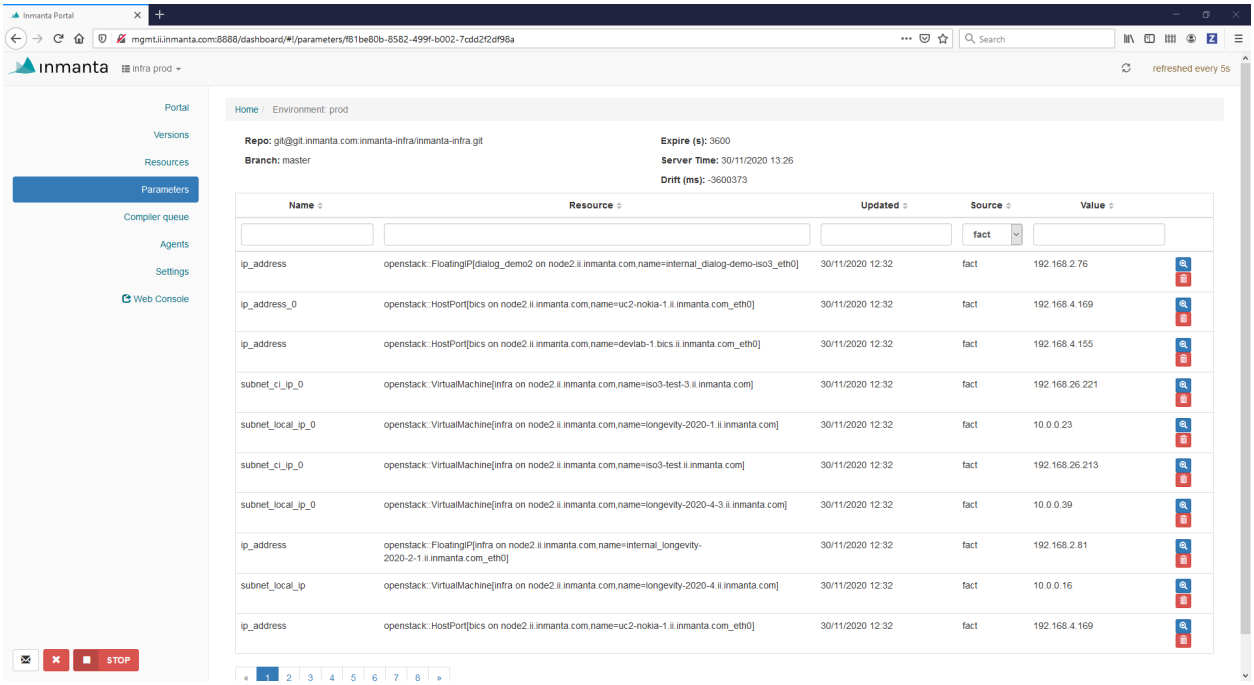


Fig. 12: The Parameter overview

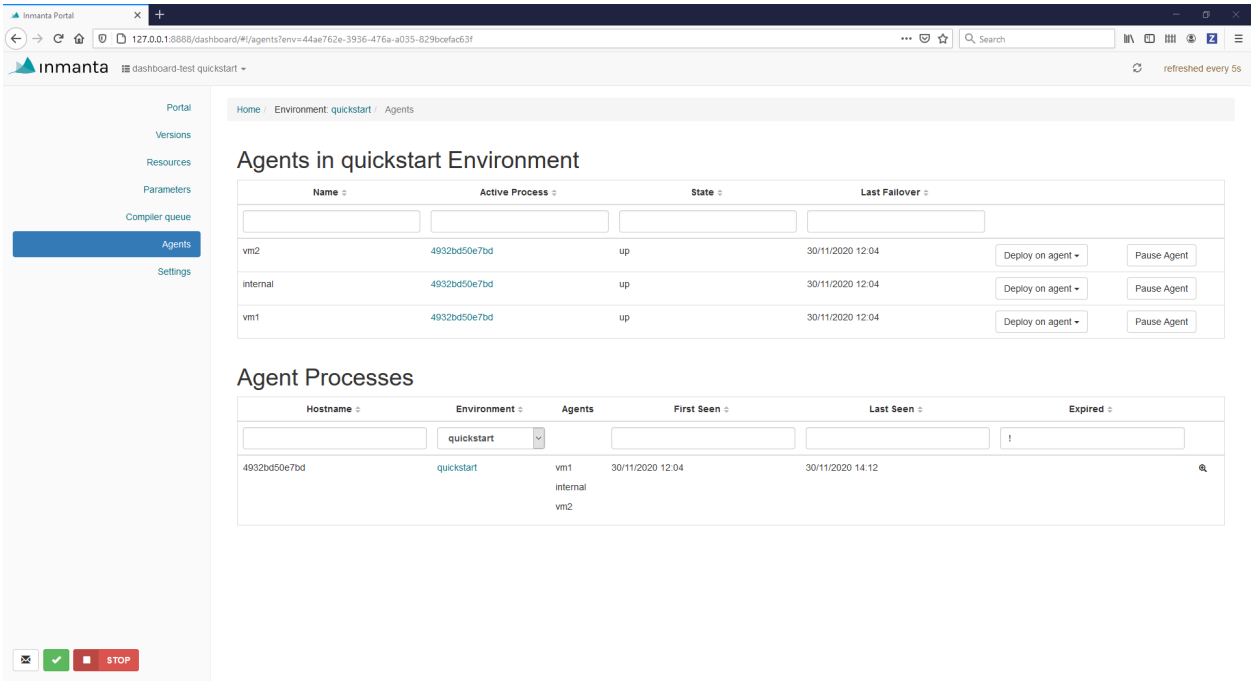


Fig. 13: The Agent overview

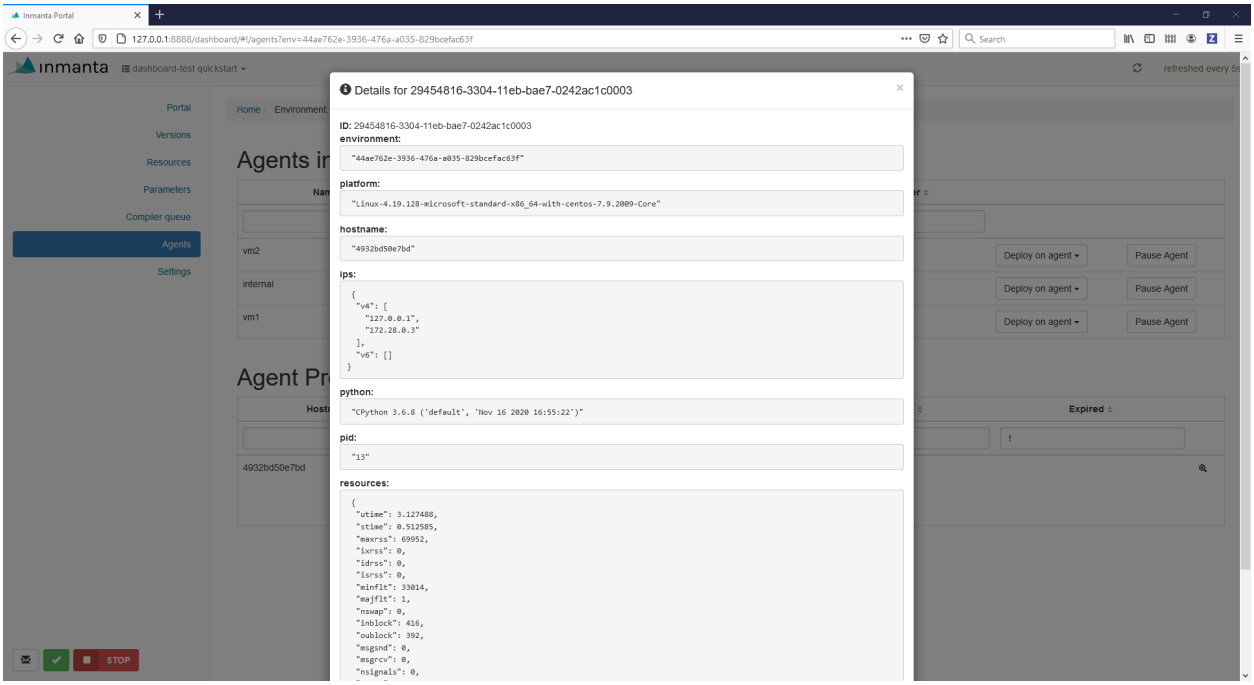


Fig. 14: Agent process inspection

3.8 Environment Settings

The settings menu shows settings that are configured per environment.

Hovering over the information icon tells you what each setting does, the edit icon allows for updating the setting and the delete button clears the setting and applies a default value if available.

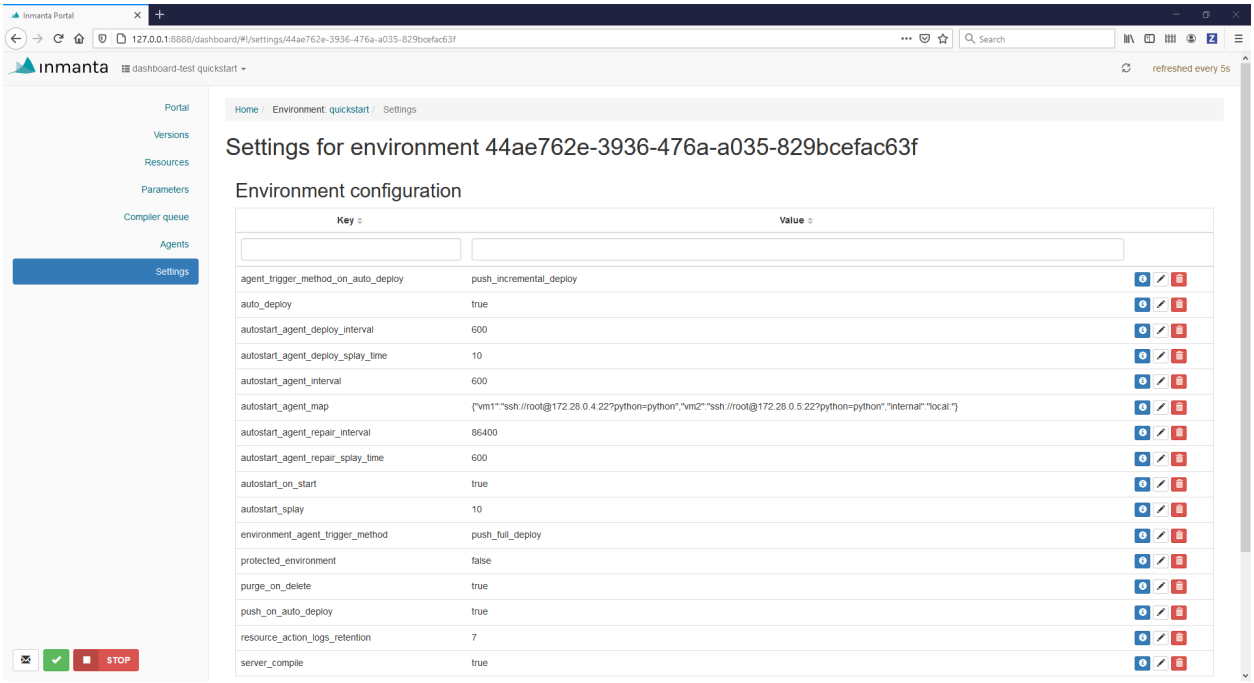


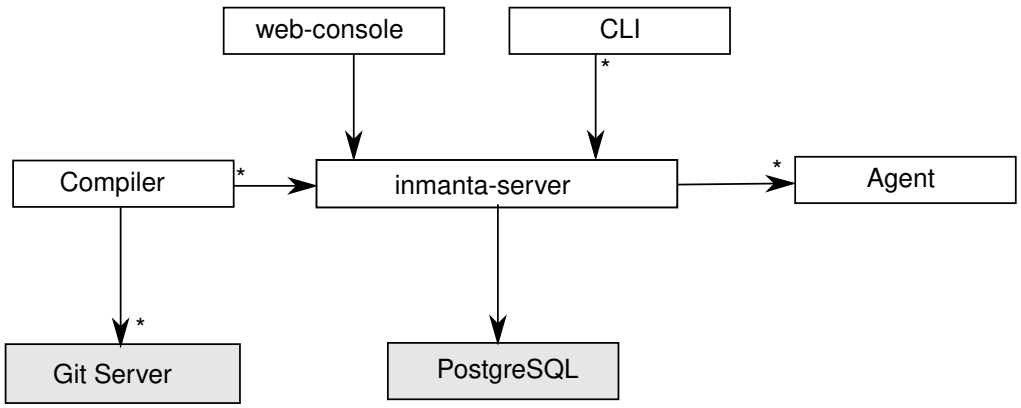
Fig. 15: The Environment settings

CHAPTER

FOUR

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.

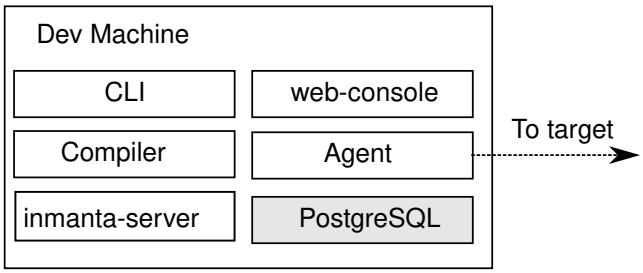
4.1 Usage modes

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.

4.1.1 All in one

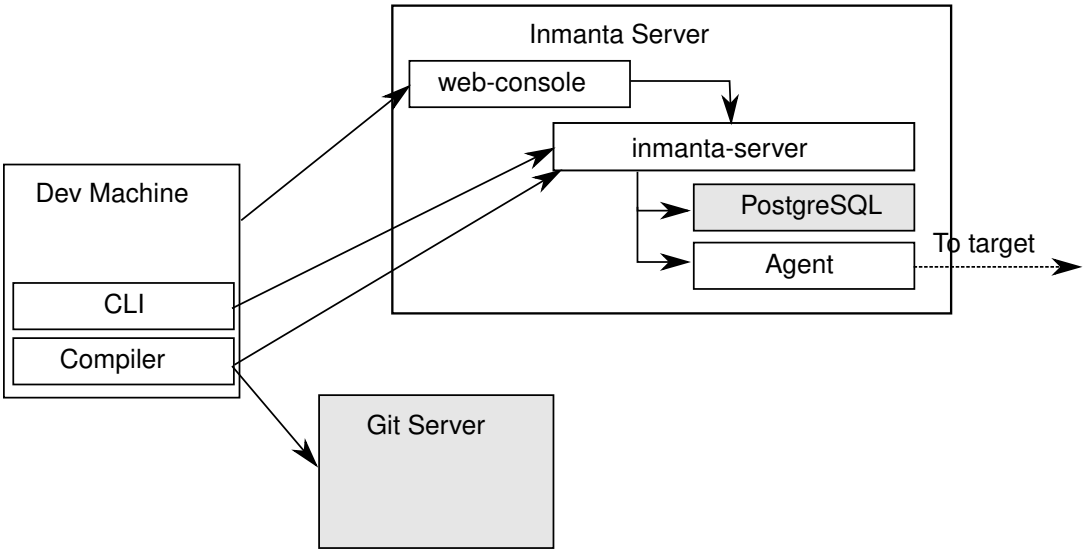


In a all-in-one deployment, all components (server, agent and postgres) are started embedded in the compiler and terminated after the deploy is complete. No specific setup is required. To deploy the current model, use:

```
inmanta deploy
```

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

4.1.2 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:

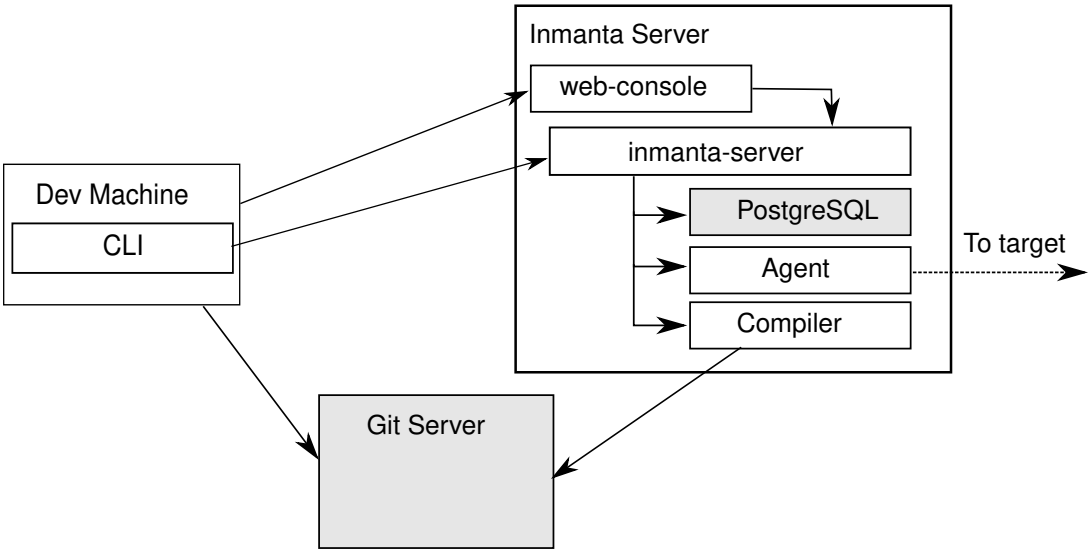
```
[config]
environment=$ENV_ID

[compiler_rest_transport]
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.

4.1.3 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.

4.2 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.

4.3 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

4.3.1 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.

4.3.2 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.

4.3.3 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

FIVE

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.

5.1 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
```

5.2 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
```

5.3 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
```

(continues on next page)

(continued from previous page)

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

5.4 Primitive types

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

```
assert = true
assert = int("1") == 1
assert = number("1.2") == 1.2
assert = number(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[]`, `number[]`, `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"
```

5.5 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
    | functioncall
    | value 'is' 'defined'
    ;
```

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

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.

5.6 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)
```

5.7 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;
```

(continues on next page)

(continued from previous page)

```
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
```

5.8 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.

Note: In previous version another relation syntax was used that was less natural to read and allowed only bidirectional relations. The relation above was defined as `File file [1:] -- [1] Service service` This synax is deprecated but still widely used in many modules.

5.9 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)
```

5.10 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="",
                                 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
                   ;
```

5.11 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
```

5.12 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';
```

5.13 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.

5.14 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.

5.15 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.

5.15.1 String interpolation

String interpolation allows variables to be included as parameters inside a string.

The included variables are resolved in the lexical scope of the string they are included in.

Interpolating strings

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

To prevent string interpolation, use raw strings

```
# this string will go into the variable as is
# containing the {{ and \n
motd = r"Welcome to {{hostname}}\n"
```

5.15.2 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 }}
```

5.16 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

SIX

MODULE GUIDES

6.1 Graph module usage

The graph module provides two exporters: 1. class diagram exporter to convert inmanta model into a [plantuml](https://plantuml.com/) class diagram 2. an instance diagram exporter that generates dot and png files based on a diagram definition.

Warning: This module is experimental code. It will not affect the result of what will be deployed. However the generation of diagram may not work very consist.

6.1.1 Class Diagrams

Add following snippet to your model:

```
graph::ClassDiagram(name="my_diagram", moduleexpression=["std::.*"], header="""
skinparam monochrome true
skinparam shadowing false
set namespaceSeparator ::
left to right direction""")
```

then export using

```
inmanta -vv export  -j x.json --export-plugin=classdiagram
plantuml my_diagram.puml -tsvg
```

This will produce a class diagram for the module 'std'.

6.1.2 Diagram definition

Add following snippet to your model:

Add the graph filter file *./files/files_and_hosts.g*

```
std::Host
std::File
std::File.host
```

This will filter all *std::Host* and *std::File* instance out of the model and add them to the graph. The statement *std::File.host* will add all the *host* relations of all the files to the graph as well.

to generate the graph

```
inmanta -vv export  -j x.json --export-plugin=graph
```

This will create a file *my_graph.dot* and *my_graph.png*

6.1.3 Install

Add in the .inmanta file of your project in the config section graph to the export option. For example:

```
[config]
environment=f603387a-f1af-4148-a286-c4d309ef4ada
export=graph
```

When *inmanta export* is called, the compiler will not only send the resources to the orchestration server but also call the graph export plugin.

6.1.4 Settings

The plugin has settings that can be added to the inmanta config file (.inmanta or other specified). All settings are set in the [graph] section.

- output-dir: The location where all the generated graphs are stored.
- **types: A list of file types that should be generated. By default a png is generated. This list can contain multiple values** seperated with commas. If only the dot file is required an empty value should be provided.

6.1.5 Diagram definition

The export plugin searches in the complete configuration module to instances of graph::Graph. This instance defines the name of the generated file and a config attribute that provides the graph instruction by means of a very limited DSL.

Each line of the diagram DSL can contain empty lines, comments (start with #), an entity type with optional settings and relation definitions also with optional settings. The DSL selects both the entity instances and relations between these instances to show in a diagram.

Entity type

Select all instance of a certain entity by specifying the full name of the type.

Between square brackets options can be specified:

- **label: The label of the instances. It can be either an attribute of the entity or a string indicated with double** quotes. This string can contain formatters between curly braces {}. Between these braces name of the attributes can be used.
- container: If set to true, this node will be treated as a container that can contain other nodes. See, type=contained_in

For example: ` `std::File[label=path] std::Service[label="Service name {name}"]` `

Entity relations

With the full name of the entity and the name of the relation, edges between instances are added to the graph.

Between square brackets options can be specified:

- label: The label on the edge. Either the name of the attribute when nothing is specified or a string with double quotes.
- **type: This can changes the relation type. The options are:**
  - **contained_in: This means that this relation indicates that the node should be placed inside the target node of** relation.

6.2 OpenStack

The openstack module provides support for managing various resources on OpenStack, including virtual machines, networks, routers, ...

This guide explains how to start virtual machines on OpenStack.

6.2.1 Prerequisites

This tutorial requires you to have an account on an OpenStack. The example below loads the required credentials from environment variables, just like the OpenStack command line tools. Additionally, the following parameters are also required:

ssh_public_key	Your public ssh key (the key itself, not the name of the file it is in)
network_name	The name of the Openstack network to connect the VM to
subnet_name	The name of the Openstack subnet to connect the VM to
network_address	The network address of the subnet above
flavor_name	The name of the Openstack flavor to create the VM from
image_id	The ID of the Openstack image to boot the VM from
os	The OS of the image

The model below exposes these parameters at the top of the code snippet.

6.2.2 Creating machines

```
import openstack
import ssh
import redhat
import ubuntu

## Edit this parameters
image_id = ""
network_name = ""
subnet_name = ""
network_address = ""

flavor_name = ""
```

(continues on next page)

(continued from previous page)

```
ssh_public_key=""

# change OS parameter to match the actual image. If an OS is not modelled in an existing module,
# std::linux can be used for example. However, other modules might not have support for a
# generic os definition such as std::linux
os = redhat::fedora23
## End edit

# register ssh key
ssh_key = ssh::Key(name="mykey", public_key=ssh_public_key)

# Define the OpenStack provider to use
provider = openstack::Provider(name="iaas_openstack", connection_url=std::get_env("OS_AUTH_URL"),
                               username=std::get_env("OS_USERNAME"),
                               password=std::get_env("OS_PASSWORD"),
                               tenant=std::get_env("OS_PROJECT_NAME"))

# Define the project/tenant to boot the VM in, but do not let inmanta manage it
project = openstack::Project(provider=provider, name=provider.tenant, description="",enabled=true,
                             managed=false)

# Define the network objects to connect the virtual machine to but again, do not manage them
net = openstack::Network(provider=provider, project=project, name=network_name,managed=false)
subnet = openstack::Subnet(provider=provider, project=project, network=net, dhcp=true,managed=false,
                           name=subnet_name, network_address=network_address)

# Define the virtual machine
vm = openstack::Host(provider=provider, project=project, key_pair=ssh_key, name="testhost",
                     image=image_id, os=os, flavor=flavor_name, user_data="",subnet=subnet)
```

6.2.3 Getting the agent on the machine

The user_data attribute of the *openstack::VirtualMachine* entity can inject a shell script that is executed at first boot of the virtual machine (through cloud-init). Below is an example script to install the inmanta agent (from RPM) and let it connect back to the management server.

```
#!/bin/bash

hostname {{ name }}
setenforce 0

cat > /etc/yum.repos.d/inmanta.repo <<EOF
[bartvanbrabant-inmanta]
```

(continues on next page)

(continued from previous page)

```
name=Copr repo for inmanta owned by bartvanbrabant
baseurl=https://copr-be.cloud.fedoraproject.org/results/bartvanbrabant/inmanta/fedora-\
↪$releasever-\$basearch/
type=rpm-md
skip_if_unavailable=True
gpgcheck=1
gpgkey=https://copr-be.cloud.fedoraproject.org/results/bartvanbrabant/inmanta/pubkey.gpg
repo_gpgcheck=0
enabled=1
enabled_metadata=1
EOF

dnf install -y python3-inmanta-agent

cat > /etc/inmanta/agent.cfg <<EOF
[config]
heartbeat-interval = 60
fact-expire = 60
state-dir=/var/lib/inmanta
environment={{ env_id }}
agent-names=\$node-name
[agent_rest_transport]
port={{port}}
host={{env_server}}
EOF

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

6.2.4 Pushing config to the machine

To install config:

```
#put a file on the machine
std::ConfigFile(host = host1, path="/tmp/test", content="I did it!")
```

6.2.5 Actual usage

Creating instances of *openstack::Host*, as shown above requires many parameters and relations, creating a model that is hard to read. Often, these parameters are all the same within a single model. This means that Inmanta can encapsulate this complexity.

In a larger model, a new `Host` type can encapsulate all settings that are the same for all hosts. Additionally, an entity that represents the *infrastructure* can hold shared configuration such as the provider, monitoring, shared networks, global parameters,...)

For example (full source here)

Applied to the example above the main file is reduced to:

```
import mymodule
import ssh
import redhat
import ubuntu

## Edit this parameters
image_id = ""
network_name = ""
subnet_name = ""
network_address = ""

flavor_name = ""
ssh_public_key=""

# change OS parameter to match the actual image. If an OS is not modelled in an existing module,
# std::linux can be used for example. However, other modules might not have support for a
# generic os definition such as std::linux
os = redhat::fedora23
## End edit

# register ssh key
ssh_key = ssh::Key(name="mykey", public_key=ssh_public_key)

# create the cluster
cluster = mymodule::MyCluster(network_name=network_name, subnet_name=subnet_name,
                              image_id=image_id, flavor=flavor_name, key=ssh_key,
                              network_address=network_address, os=os)

# make a vm!
host1 = mymodule::MyHost(name="testhost", cluster=cluster)
```

With the following module:

```
import openstack
import ssh


entity MyCluster:
    """
        A cluster object that represents all shared config and infrastructure,
        including connecting to OpenStack.
    """
    string network_name
    string subnet_name
    string network_address
    string image_id
    string flavor
end

#input: the ssh key for all VMs
MyCluster.key [1] -- ssh::Key

```

(continues on next page)

(continued from previous page)

```
#input: the OS for all VMs
MyCluster.os [1] -- std::OS

#internal: objects needed to construct hosts
MyCluster.provider [1] -- openstack::Provider
MyCluster.project [1] -- openstack::Project
MyCluster.net [1] -- openstack::Network
MyCluster.subnet [1] -- openstack::Subnet

implementation connection for MyCluster:
    # Define the OpenStack provider to use
    self.provider = openstack::Provider(name="iaas_openstack",
                                        connection_url=std::get_env("OS_AUTH_URL"),
                                        username=std::get_env("OS_USERNAME"),
                                        password=std::get_env("OS_PASSWORD"),
                                        tenant=std::get_env("OS_PROJECT_NAME"))

    # Define the project/tenant to boot the VM in, but do not let inmanta manage it
    self.project = openstack::Project(provider=self.provider, name=self.provider.tenant,
                                      description="", enabled=true, managed=false)

    # Define the network objects to connect the virtual machine to but again, do not␣
↪manage them
    self.net = openstack::Network(provider=self.provider, project=self.project,
                                  name=self.network_name, managed=false)
    self.subnet = openstack::Subnet(provider=self.provider, project=self.project,
                                    network=self.net, dhcp=true, name=self.subnet_name,
                                    network_address=self.network_address, managed=false)
end

implement MyCluster using connection

#define our own host type
entity MyHost extends openstack::Host:
end

#input: the cluster object
MyCluster.hosts [0:] -- MyHost.cluster [1]

implementation myhost for MyHost:
    #wire up all config for agent injection
    env_name = std::environment_name()
    env_id = std::environment()
    env_server = std::environment_server()
    port = std::server_port()

    #wire up all config for vm creation
    self.provider = cluster.provider
    self.project = cluster.project
    self.image = cluster.image_id
    self.subnet = cluster.subnet
    self.user_data = std::template("mymodule/user_data.tmpl")
```

(continues on next page)

(continued from previous page)

```
    self.key_pair = cluster.key
    self.os = cluster.os
    self.flavor = cluster.flavor
end

# use our implemenation
# and also the catchall std::hostDefaults
# and the openstackVM implementation that sets the ip and create the eth0 port
implement MyHost using myhost, std::hostDefaults, openstack::openstackVM,␣
↪openstack::eth0Port
```

If this were not an example, we would make the following changes:

- hardcode the `image_id` and `os` (and perhaps `flavor`) into the defintion of `myhost`.
- the parameters on top would be moved to either a forms or filled in directly into the constructor.
- use `std::password` to store passwords, to prevent accidential check-ins with passwords in the source

CHAPTER

SEVEN

MODEL DEVELOPER DOCUMENTATION

7.1 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`.

7.1.1 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.

7.1.2 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
```

7.1.3 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
```

7.1.4 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.

V2 module source

Add the pip index where your modules are hosted to `project.yml` as a repo of type `package`. For example, for modules hosted on PyPi:

```
repo:
    - url: https://pypi.org/simple
      type: package
```

7.1.5 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
inmanta module 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.

7.1.6 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
```

7.2 Project creation guide

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

7.2.1 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:

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

(continues on next page)

(continued from previous page)

```
      type: package
install_mode: release
requires:
```

7.2.2 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.

7.3 Module creation guide

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

7.3.1 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.

7.4 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.

7.4.1 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

  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.
- 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.

7.4.2 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.

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

```
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*.

7.4.3 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
```

7.4.4 Inmanta module template

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

7.4.5 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.

7.5 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.

7.5.1 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 `inmanta module 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 `inmanta module 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`.

7.5.2 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.

7.5.3 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.

7.6 Releasing and distributing modules

7.6.1 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`.

7.6.2 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 ease the use inmanta provides a command (`inmanta modules commit`) to modify module versions, commit to git and place the correct tag.

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 *
```

To commit, use the module tool. This will create a new dev release.

```
inmanta module commit --patch -m "Fixed small bug"
```

This command will set the version to the next dev version (*+0.0.1dev*) and add a timestamp.

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.

Release Versions

To make an actual release (without *.dev* at the end):

```
inmanta module commit -r -m "First Release"
```

This will remove the *.dev* version and automatically set the right tags on the module.

To automatically freeze all dependencies of this module to the currently checked out versions

```
inmanta module freeze --recursive --operator ==
```

Or for the current project

```
inmanta project freeze --recursive --operator ==
```

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.

7.6.3 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`.

7.7 Developing 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:

```
from inmanta.plugins import plugin

@plugin
def hello():
    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():
    """
        Prints inmanta
    """
    print("inmanta")
```

A more complex plugin accepts arguments and returns a value. 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"):
    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*.

7.8 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.

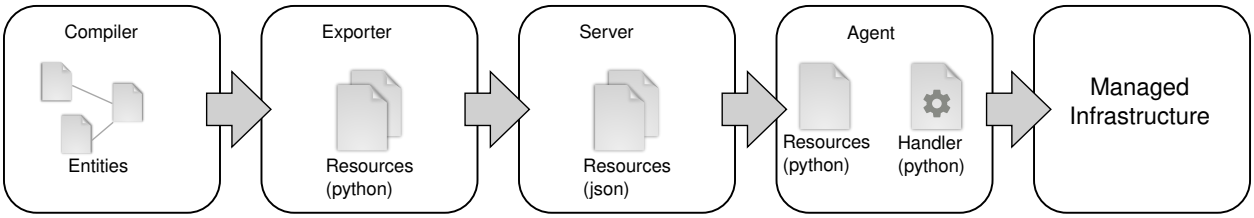
To integrate a new platform into the orchestrator, you must take the following steps:

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.

7.8.1 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.

7.8.2 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")

    @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.

7.8.3 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.

7.8.4 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 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)
```

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):
   # ...
   return connection
```

7.9 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.

7.9.1 Install the pytest-inmanta package

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

```
pip install pytest-inmanta
```

7.9.2 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:

```
# example_module/tests/test_hostname.py

def test_hostname(project, inmanta_plugins):
    host = "test"
    fqdn = f"{host}.something.com"
    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

7.10 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
```

7.11 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.

7.11.1 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

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

7.11.2 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

```
z
EQUIVALENT TO {x, y, z} DUE TO STATEMENTS:
    `x = y` AT ./assignment-loop.cf:1
    `y = z` AT ./assignment-loop.cf:2
    `z = x` AT ./assignment-loop.cf:3
├── u
│   SET BY `z = u`
│   AT ./assignment-loop.cf:6
│   └── 1
│       SET BY `u = 1`
│       AT ./assignment-loop.cf:7
└── 0
    SET BY `x = 0`
    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

```
1  entity A:
2      number n
3  end
4
5  implement A using std::none
6
7  x = A(n = 0)
8
9  template = x
10
11 y = A(n = template.n)
12 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

x = A(n = 0)
```

Listing 9: data trace for implementation.cf

```
attribute n on __config__::A instance
SUBTREE for __config__::A instance:
    CONSTRUCTED BY `A(n=0)`
    AT ./implementation.cf:11
├── 0
    SET BY `A(n=0)`
    AT ./implementation.cf:11
└── 42
    SET BY `self.n = 42`
    AT ./implementation.cf:8
    IN IMPLEMENTATION WITH self = __config__::A instance
        CONSTRUCTED BY `A(n=0)`
        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

```
attribute m on __config__::A instance
SUBTREE for __config__::A instance:
```

(continues on next page)

(continued from previous page)

```
    CONSTRUCTED BY `A(n=42,m=0)`
    AT ./index.cf:10

    INDEX MATCH: `__config__::A instance`
        CONSTRUCTED BY `A(n=42,m=1)`
        AT ./index.cf:11
├── 1
│   SET BY `A(n=42,m=1)`
│   AT ./index.cf:11
└── 0
    SET BY `A(n=42,m=0)`
    AT ./index.cf:10
```

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

7.11.3 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.

7.11.4 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,
```

(continues on next page)

(continued from previous page)

```
)
```

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

```
attribute service on __config__::Port instance
SUBTREE for __config__::Port instance:
    CONSTRUCTED BY `Port(host=self.host,portn=self.portn)`
    AT ./service.cf:22
    IN IMPLEMENTATION WITH self = __config__::Service instance
        CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
        AT ./service.cf:26

    INDEX MATCH: `__config__::Port instance`
        CONSTRUCTED BY `Port(host=self.host,portn=self.portn)`
        AT ./service.cf:22
        IN IMPLEMENTATION WITH self = __config__::Service instance
            CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
            AT ./service.cf:33
├── __config__::Service instance
│   SET BY `self.port = Port(host=self.host,portn=self.portn)`
│   AT ./service.cf:22
│   IN IMPLEMENTATION WITH self = __config__::Service instance
│       CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
│       AT ./service.cf:33
│   CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
│   AT ./service.cf:33
└── __config__::Service instance
    SET BY `self.port = Port(host=self.host,portn=self.portn)`
    AT ./service.cf:22
    IN IMPLEMENTATION WITH self = __config__::Service instance
        CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
        AT ./service.cf:26
```

(continues on next page)

(continued from previous page)

```
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.

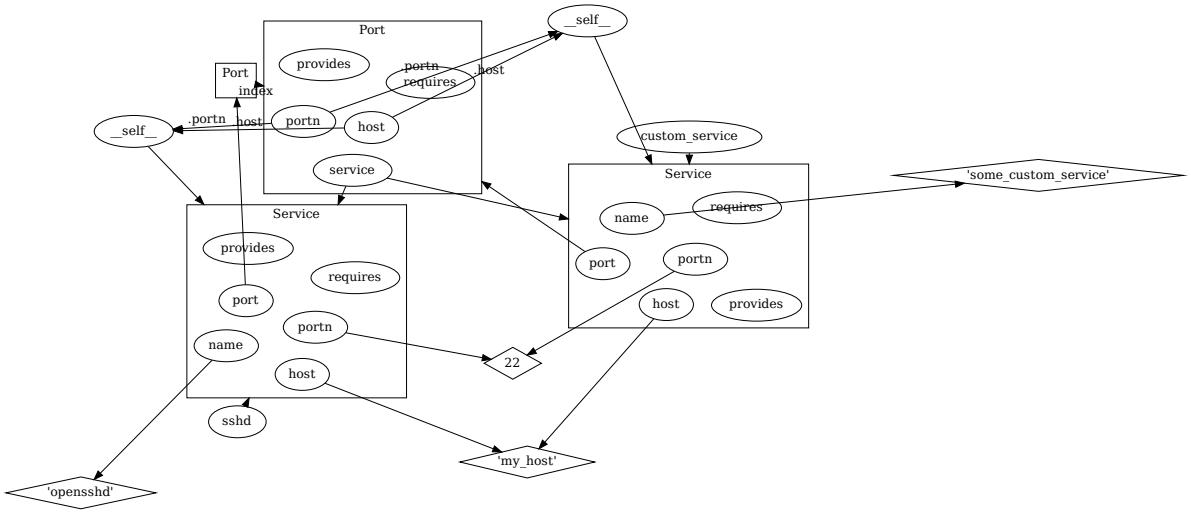
Lines 2–14 in the trace give some additional information about the `Port` instance. It indicates there is an index match between the `Port` instances constructed in the implementations for both `Service` instances. This illustrates the existence of the two branches at lines 15 and 23, and why the assignment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

7.11.5 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.

7.12 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.

7.12.1 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.

7.12.2 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.

7.12.3 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.

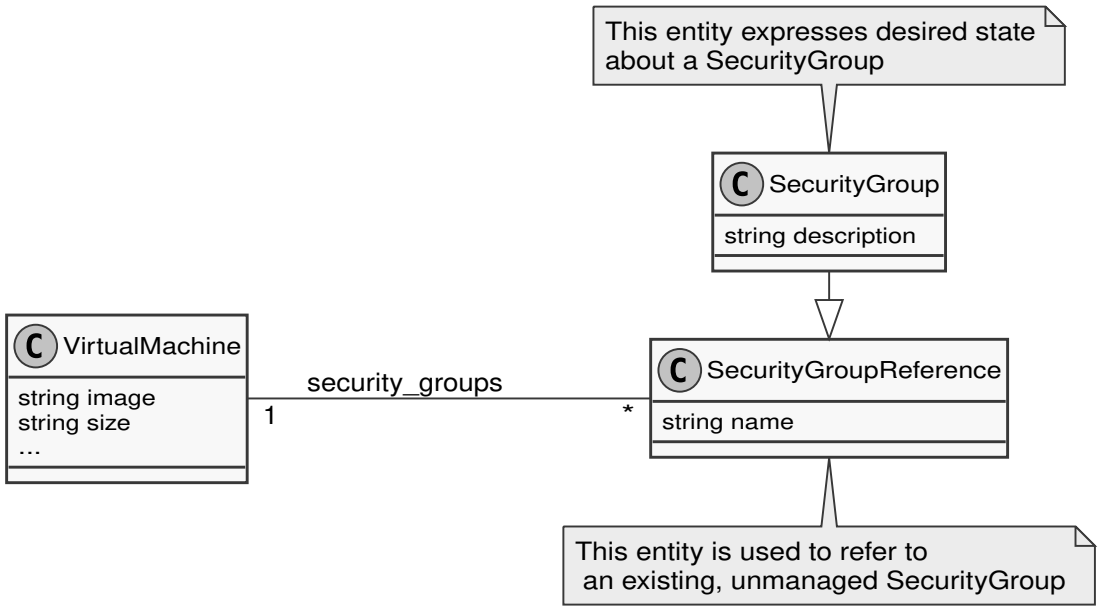
When starting to build up a model, textual reference are attractive, as the modeling effort required is very limited. It is however difficult to migrate away from the textual references later on, because this is a breaking change for any existing model.

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

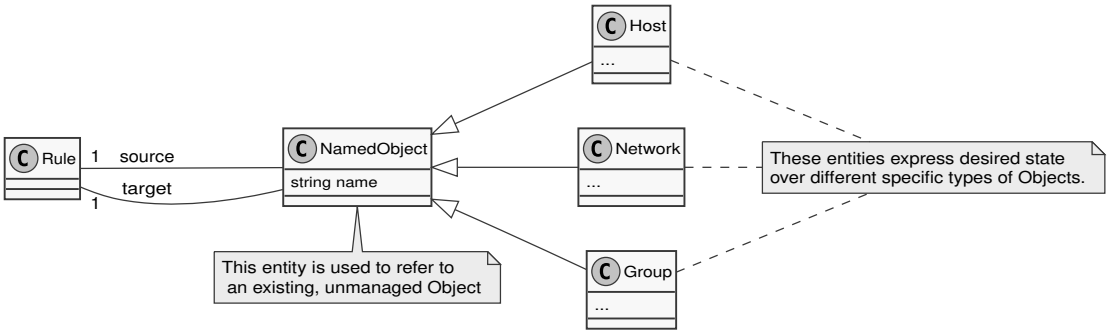
be valid. All attributes and relations still have to be filled in correctly. For entities with many non-optional relations, this is also not the best solution.

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:



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.



7.13 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*.

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.

7.13.1 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
```

(continues on next page)

(continued from previous page)

```
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
```

7.13.2 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
```

(continues on next page)

(continued from previous page)

```
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)
```

(continues on next page)

(continued from previous page)

```
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.
- The new version of the model that emerges from a partial compilation should have a dependency graph that is closed within the resource sets that were exported. i.e., it should not depend on any resource sets other than those that were exported.
- 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

Two 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.

Limitations

- **The compiler cannot verify all constraints that would be verified when a full build is run. Some index constraints, for insta**
  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 * * *`.

7.13.3 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).

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

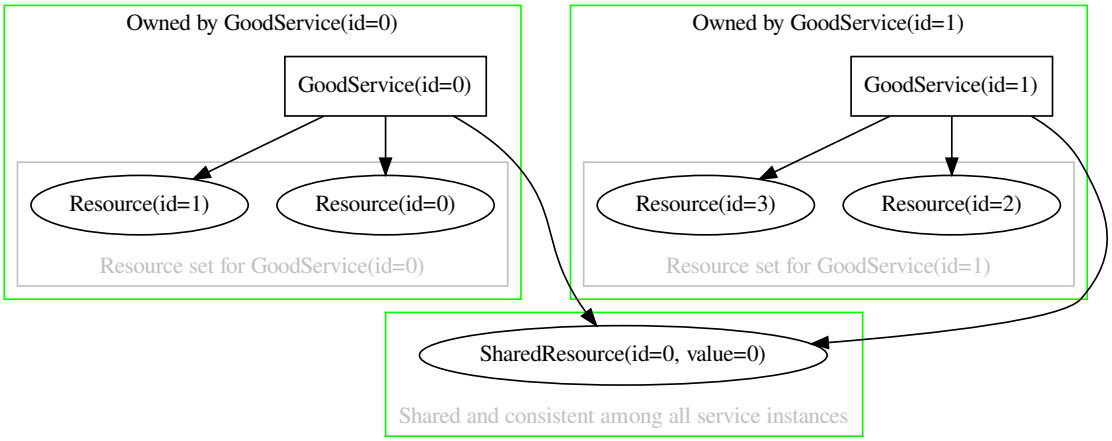


Fig. 1: A good service for partial compiles.

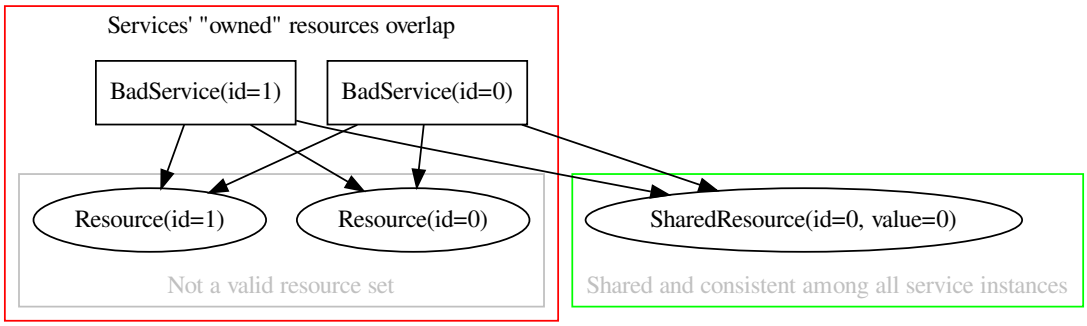


Fig. 2: A bad service for partial compiles: no owned resources

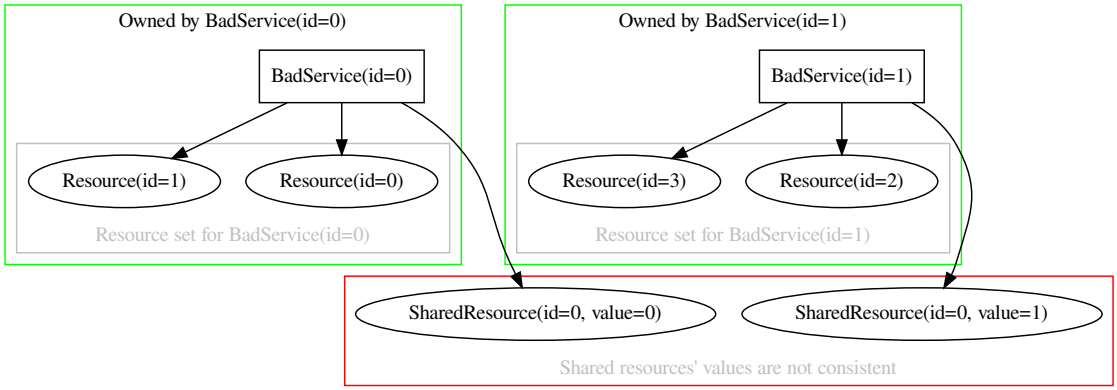


Fig. 3: A bad service for partial compiles: conflicting shared resources

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.

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:
```

(continues on next page)

(continued from previous page)

```
      - 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.

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.

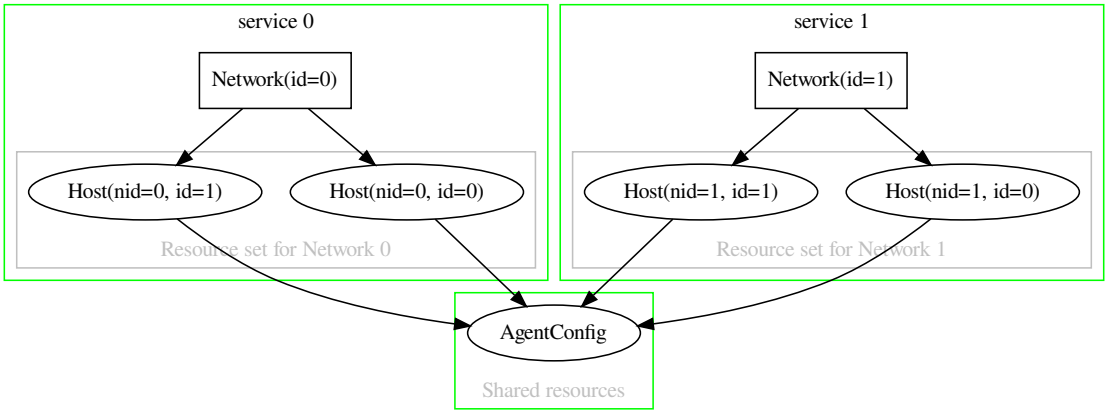


Fig. 4: Two valid service instances with their resource sets

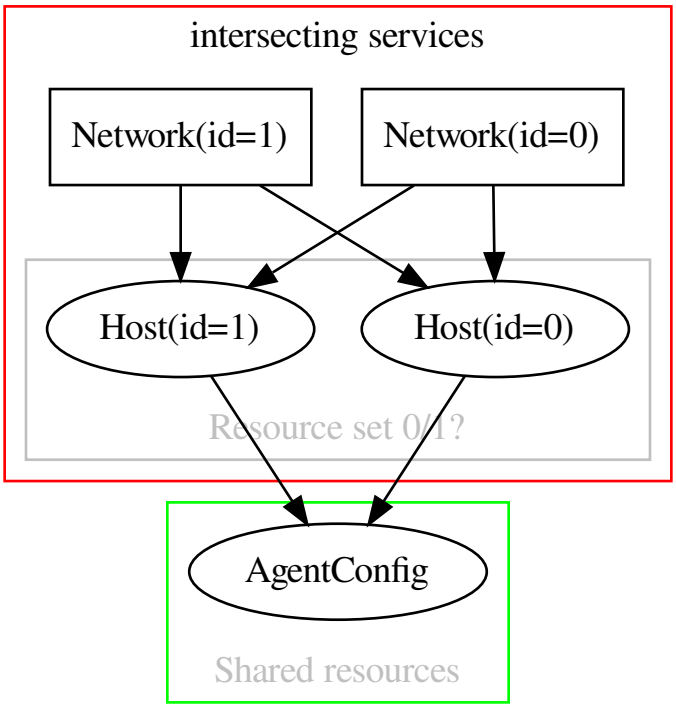


Fig. 5: Two invalid service instances with a resource set conflict

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:

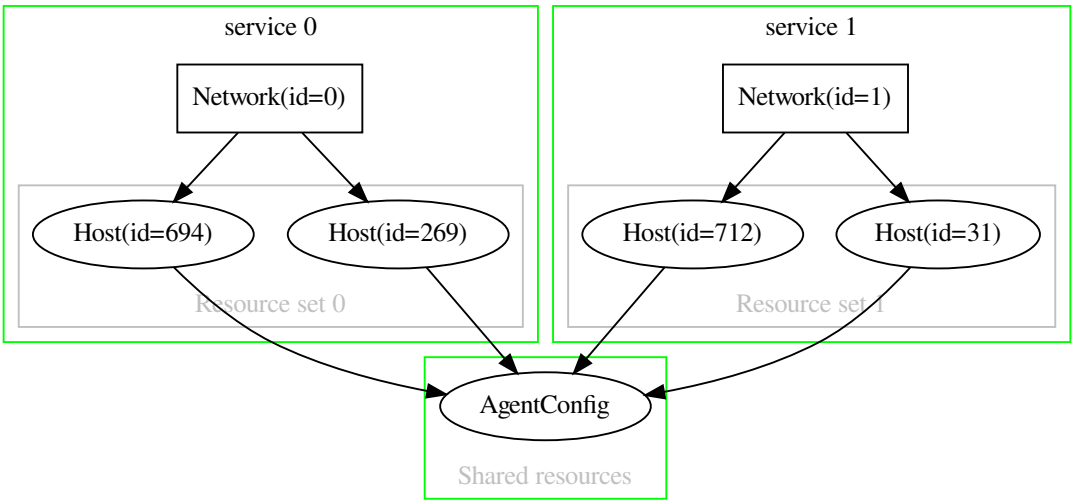


Fig. 6: Two valid services with their resource sets, using allocation

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.

CHAPTER

EIGHT

PLATFORM DEVELOPER DOCUMENTATION

8.1 Creating a new server extension

Inmanta server extensions are separate Python packages with their own release cycle that can add additional server slices 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.

8.1.1 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.

```
# File: extension.py
from inmanta.server.extensions import ApplicationContext

def setup(application: ApplicationContext) -> None:
    application.register_slice(<server-slice-instance>)
```

Tip: Indicate which version of the Inmanta core is compatible with the developed extension by pinning the version of the Inmanta core in the `requirements.txt` file of the extension.

8.1.2 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: inmanta.server.protocol.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.

8.1.3 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
```

8.1.4 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.

8.2 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.

8.2.1 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

async def update(connection: Connection) -> None:
    schema = """
    ALTER TABLE public.test
    ADD COLUMN new_column;
    """
    await connection.execute(schema)
```

8.2.2 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.

8.2.3 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.

```
1  # File: tests/db/test_v202101010_to_v202102220.py
2  @pytest.mark.db_restore_dump(os.path.join(os.path.dirname(__file__), "dumps",
   ↪"v202101010.sql"))
3  async def test_add_new_column_to_test_table(
4      migrate_db_from: abc.Callable[[], abc.Awaitable[None]],
5      get_columns_in_db_table: abc.Callable[[str], list[str]],
6  ) -> None:
7      """
8      Verify that the database migration script v202102220.py correctly adds the column␣
   ↪new_column to the table test.
9      """
10     # Assert state before migration
11     assert "new_column" not in await get_columns_in_db_table(table_name="test")
12     # Migrate DB schema
13     await migrate_db_from()
14     # Assert state after migration
15     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.

8.3 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.

8.3.1 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.

```
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: typing.Dict[str, inmanta.protocol.common.ArgOption] = {}, timeout: typing.Optional[int] = None, server_agent: bool = False, api: typing.Optional[bool] = None, agent_server: bool = False, validate_sid: typing.Optional[bool] = None, client_types: typing.List[inmanta.const.ClientType] = [<ClientType.api: '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.

8.3.2 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

@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[[...], Optional[Union[int, Tuple[int, Optional[Dict[str, Any]]], ReturnValue[ReturnTypes], ReturnValue[None], BaseModel, uuid.UUID, inmanta.types.StrictNonIntBool, float, datetime.datetime, str, Sequence[Union[BaseModel, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]], Mapping[str, Union[BaseModel, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]]]]], api_version: Optional[int] = 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 verision. 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

8.4 Documentation writing

Inmanta uses Sphinx to generate documentation.

8.4.1 Inmanta code documentation

Modules

Python core

8.4.2 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.

8.5 Exceptions

For more details about Compiler Exceptions, see *Compiler exceptions*

8.5.1 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: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: `tornado.web.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: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

An exception raised when access is denied (403)

class inmanta.protocol.exceptions.**UnauthorizedException**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

An exception raised when access to this resource is unauthorized

class inmanta.protocol.exceptions.**BadRequest**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

This exception is raised for a malformed request

class inmanta.protocol.exceptions.**NotFound**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

This exception is used to indicate that a request or reference resource was not found.

class inmanta.protocol.exceptions.**Conflict**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

This exception is used to indicate that a request conflicts with the current state of the resource.

class inmanta.protocol.exceptions.**ServerError**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

An unexpected error occurred in the server

class inmanta.protocol.exceptions.**ShutdownInProgress**(*message: Optional[str] = None*, *details: Optional[Dict[str, Any]] = None*)

Bases: *inmanta.protocol.exceptions.BaseHttpException*

This request can not be fulfilled because the server is going down

8.5.2 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

8.6 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**
   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 %}
}
```

8.7 Type Export Format

class inmanta.model.**Attribute**(*mytype: str*, *nullable: bool*, *multi: bool*, *comment: str*, *location:* inmanta.model.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,
```

(continues on next page)

(continued from previous page)

```
    "comment": self.comment,
    "location": self.location.to_dict()
}
```

class inmanta.model.**DirectValue**(*value:* inmanta.model.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,* inmanta.model.Attribute*]*, *relations: Dict[str,* inmanta.model.Relation*]*, *location:* inmanta.model.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, Optional[int]]*, *reverse: str*, *comment: str*, *location:* inmanta.model.Location, *source_annotations: List[*inmanta.model.Value*]*, *target_annotations: List[*inmanta.model.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*

8.8 Platform Developers Guide

8.8.1 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.

8.8.2 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
```

8.8.3 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

NINE

ADMINISTRATOR DOCUMENTATION

9.1 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

9.1.1 Project Release for Production

This process describes the steps to prepare an inmanta project for production release.

Context

- The project development and testing is complete
- All modules have been properly released.
- 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. Verify in *project.yml* that *install_mode* is set to *release*.
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.
```

4. 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
5. Commit this change (*git commit -a*)

6. Push to the release branch (*git push*)

9.1.2 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 setting are on (this should always be the case)
   - *purge_on_delete = False*
   - *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

9.1.3 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)
    - *purge_on_delete = False*
    - *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*

9.1.4 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 setting are on (this should always be the case)

    * [ ] `purge_on_delete = False`
    * [ ] `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
```

(continues on next page)

(continued from previous page)

```
    * [ ] `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`

   * 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.
```

9.2 Configuration

9.2.1 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 alfabetical order of the filesnames. 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.

9.2.2 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 alfabetical order of the filenames. Filenames which appear later in the alfabetical 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*

9.3 Setting up authentication

This guide explains how to enable ssl and setup authentication.

SSL is not strictly required for authentication but higly recommended. Inmanta uses bearer tokens for authorizing users and services. These tokens should be kept private and are visible in plain-text in the request headers without SSL.

9.3.1 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
```

9.3.2 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.

9.3.3 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 uses 3rd party auth brokers. Currently the web-console only supports redirecting users to keycloak for authentication.

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 a 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.

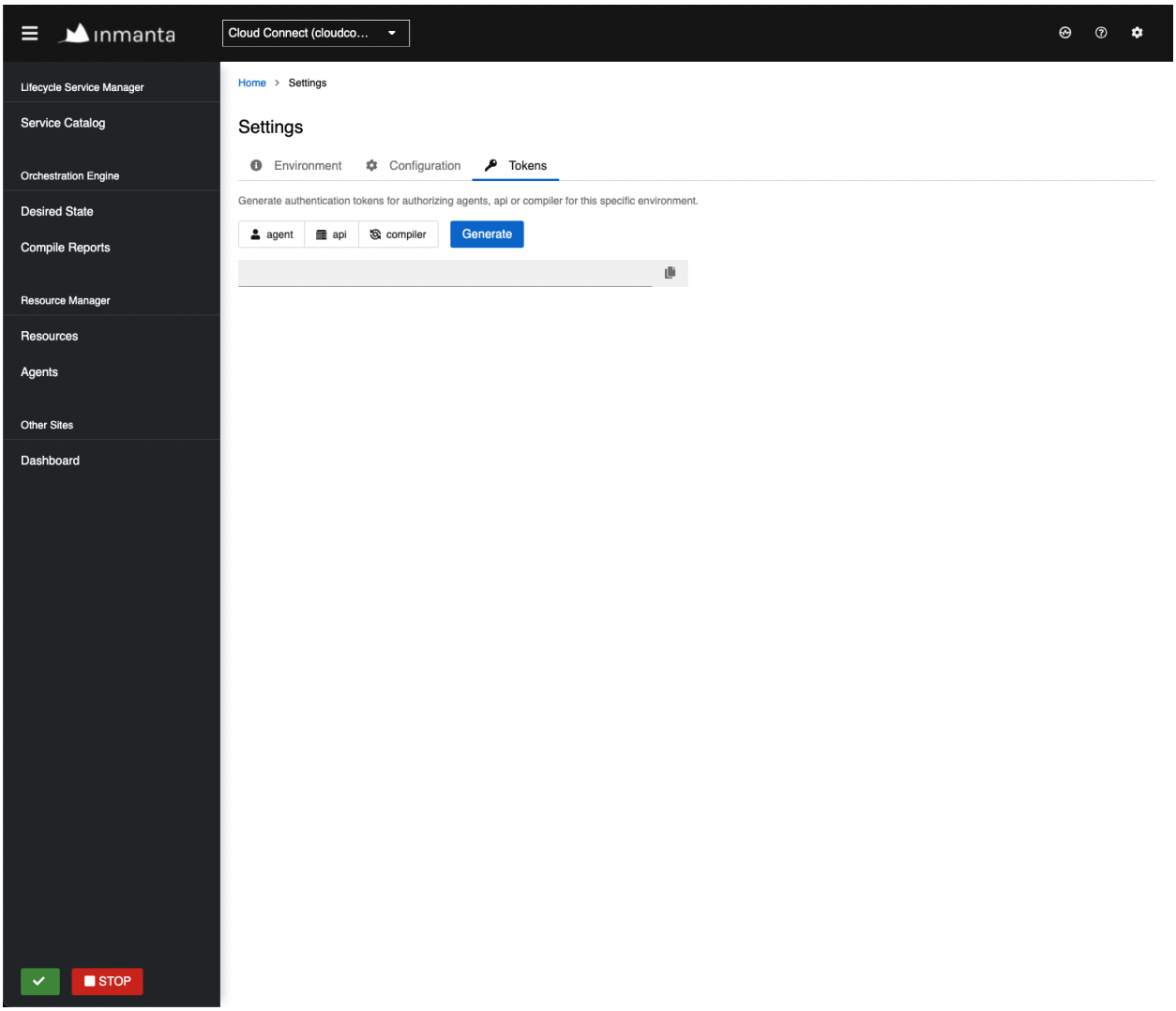


Fig. 1: Generating a new token in the web-console.

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.
- 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 key 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('='));"
```

9.3.4 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 3.3

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

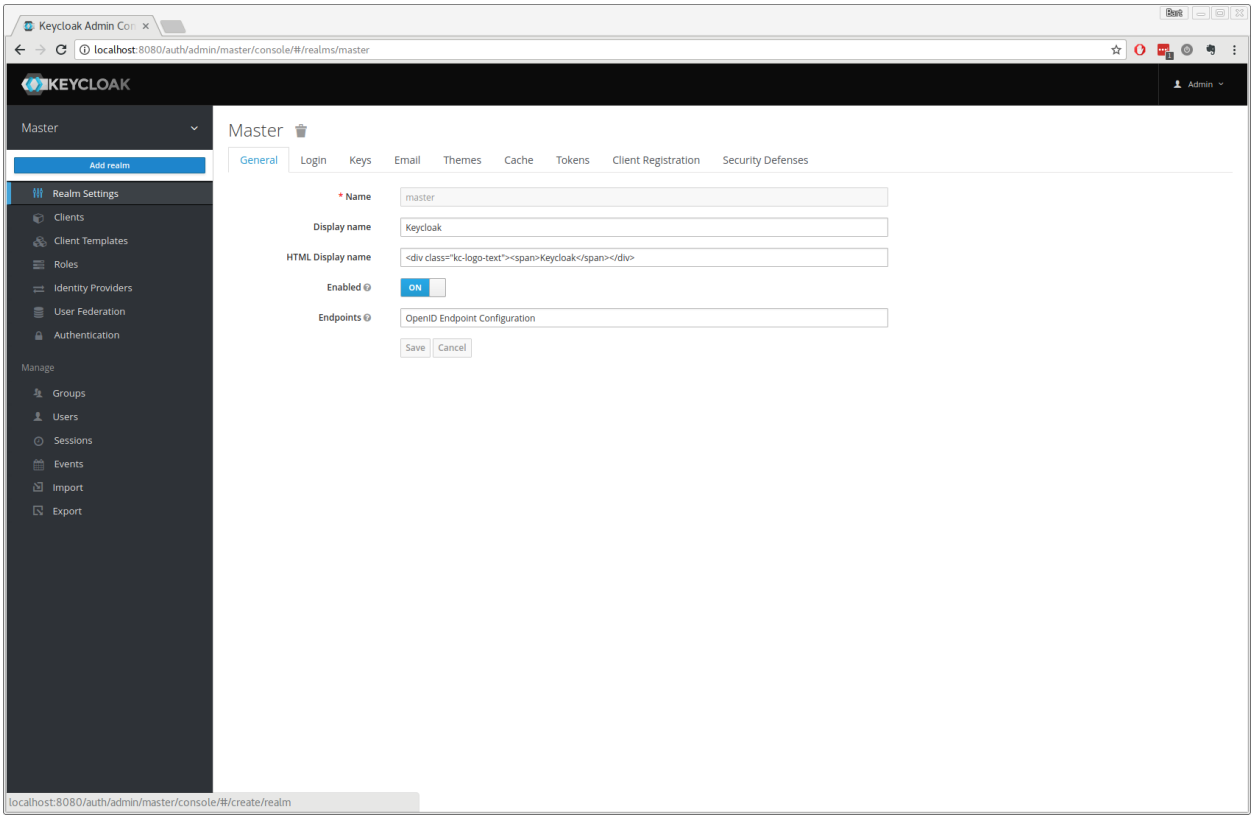


Fig. 2: Create a new realm

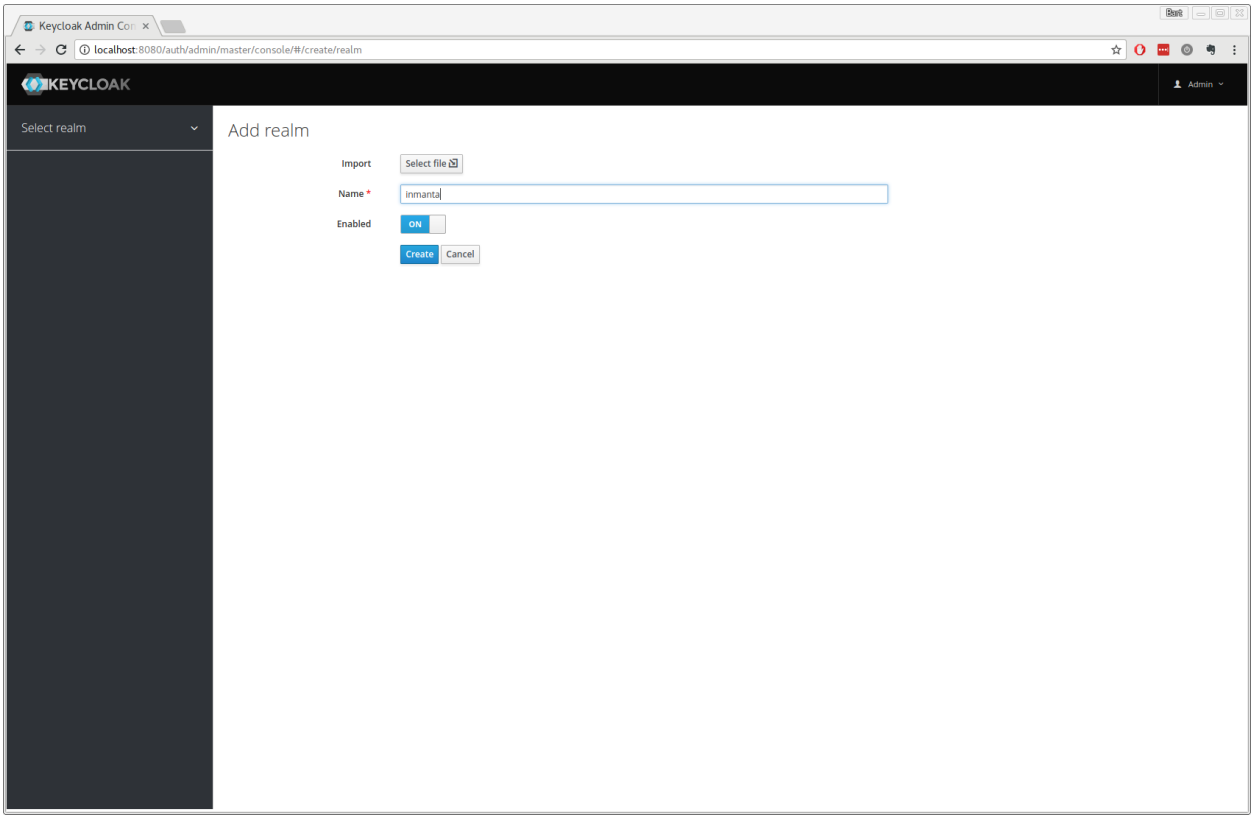


Fig. 3: Specify a name for the realm

Step 2: Add a new client to keycloak

Make sure the correct realm is active (the name is shown in the title of the left sidebar) to which you want to add a new client.

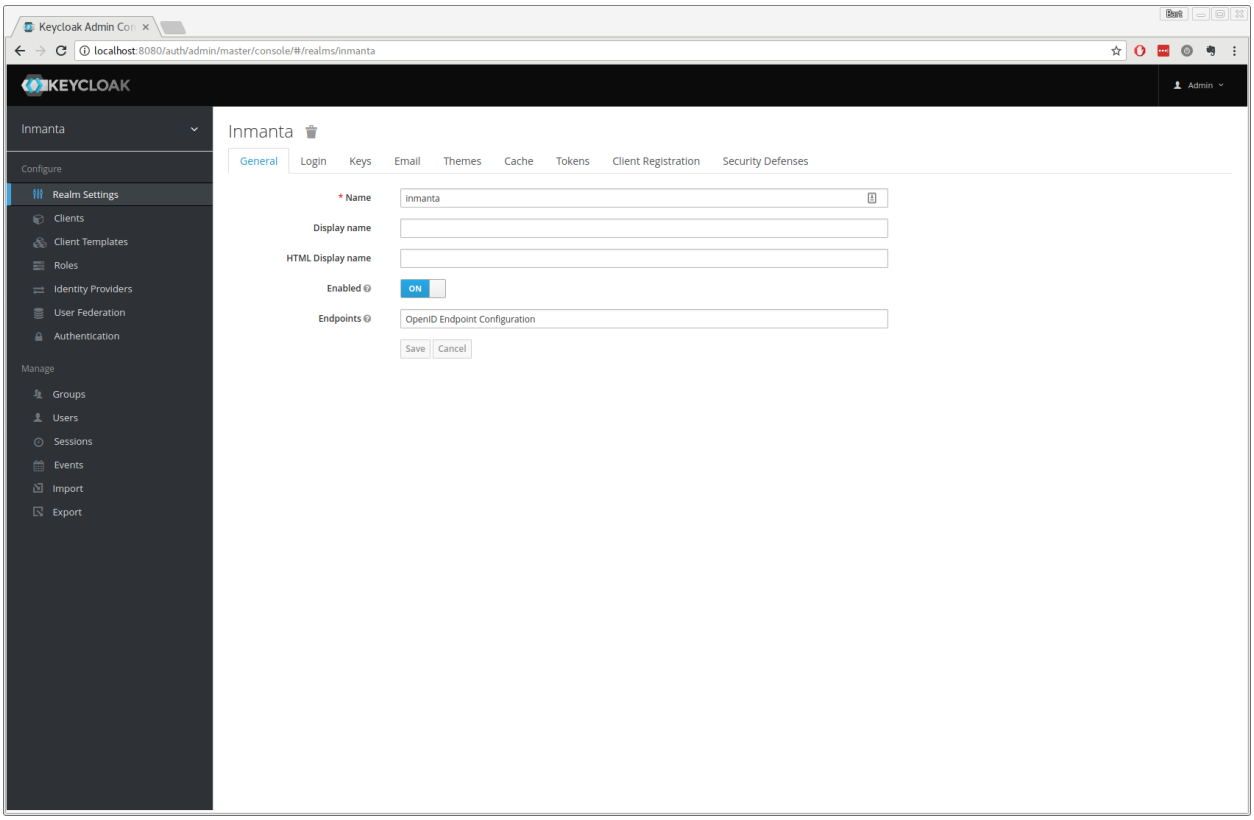


Fig. 4: The start page of a realm. Here you can edit names, policies, … of the realm. The defaults are sufficient for inmanta authentication. This shows the inmanta realm.

Go to client 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 vallid callback URLs. As a best practice, also add the allowed origins. See the screenshot below as an example.

Add a mapper to add custom claims to the issued tokens for the API client type. Open de mappers tab of your new client and click *add*.

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
- Add to access token: on

Click save.

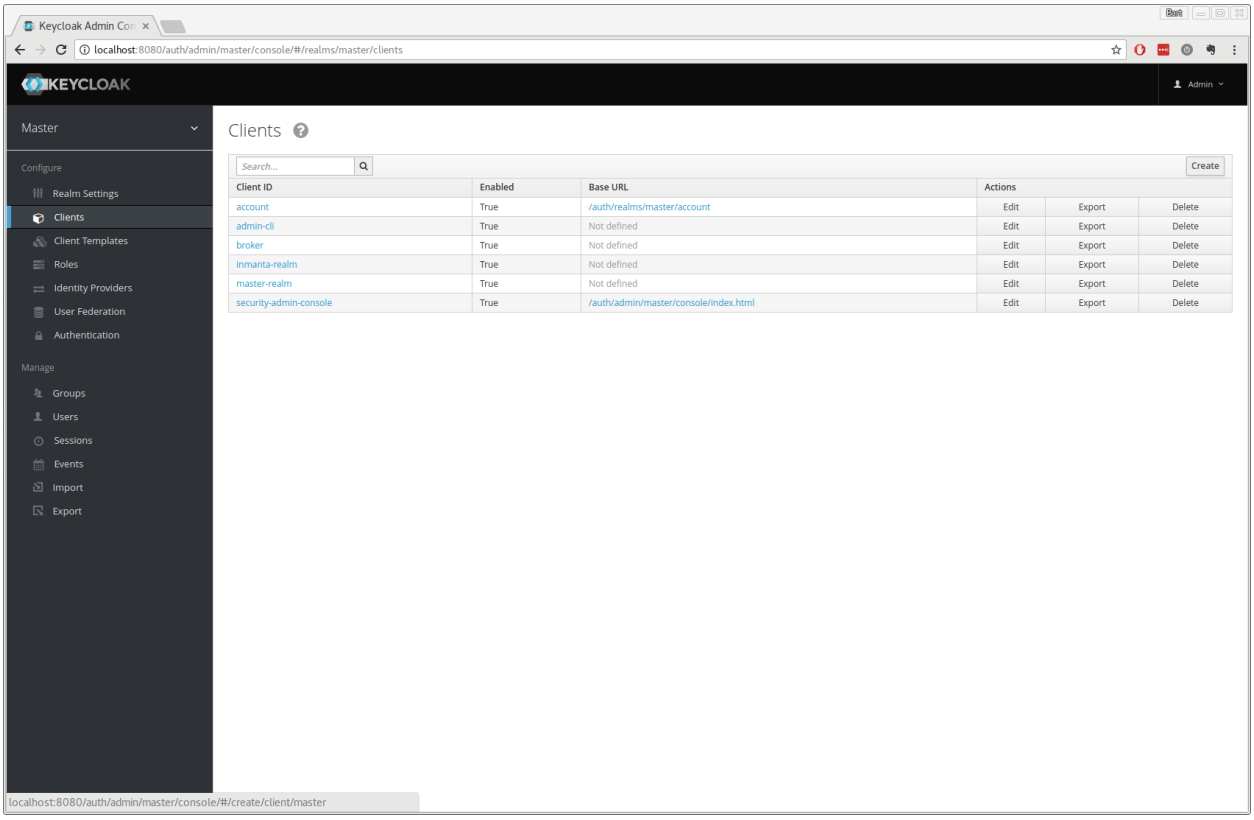


Fig. 5: Clients in the master realm. Click the create button to create an inmanta client.

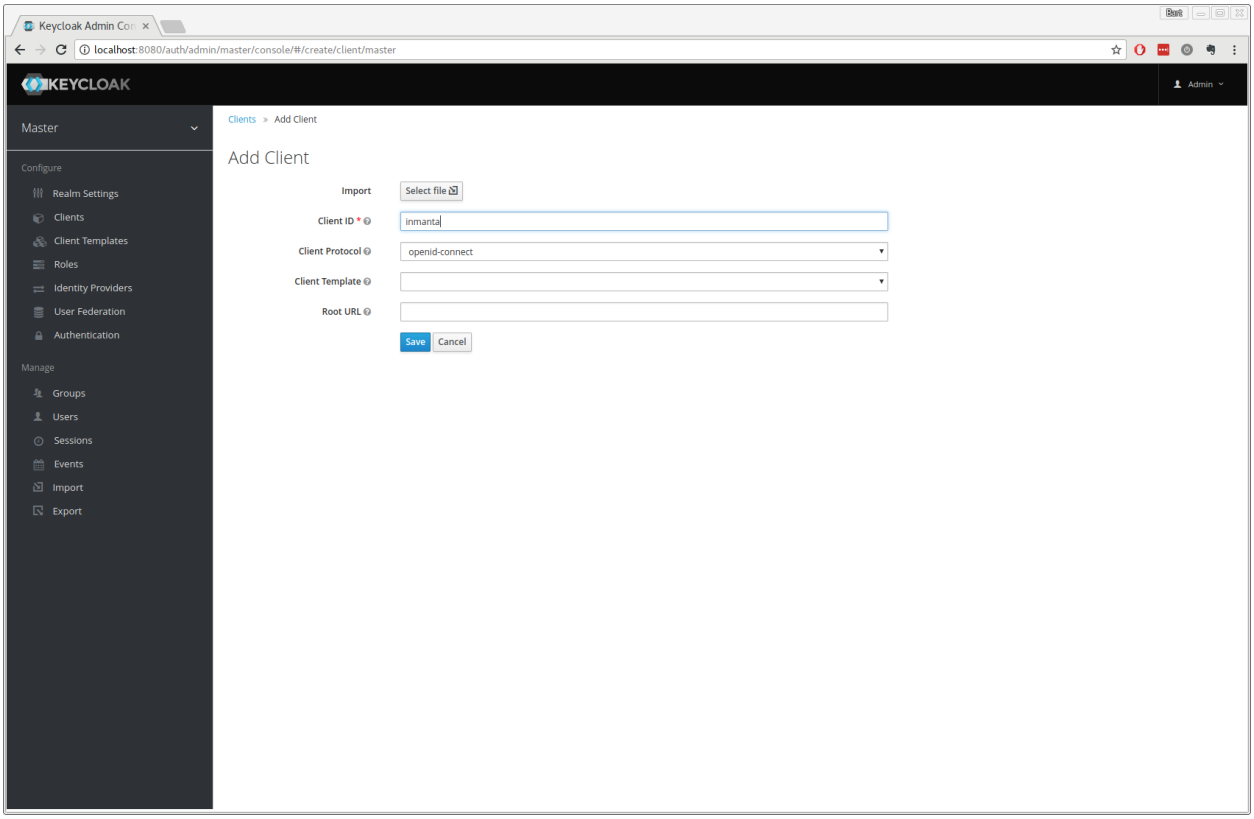


Fig. 6: Create client screen

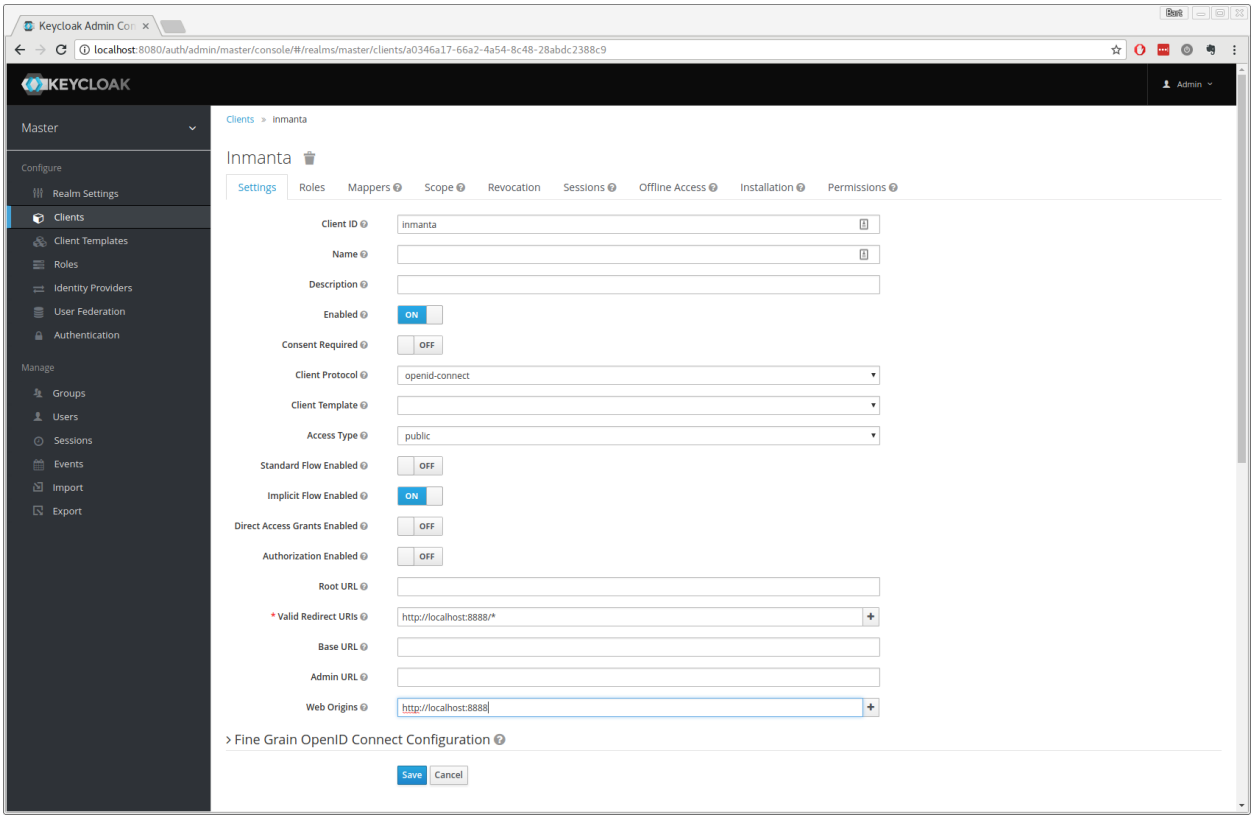


Fig. 7: Allow implicit flows (others may be disabled) and configure allowed callback urls of the web-console.

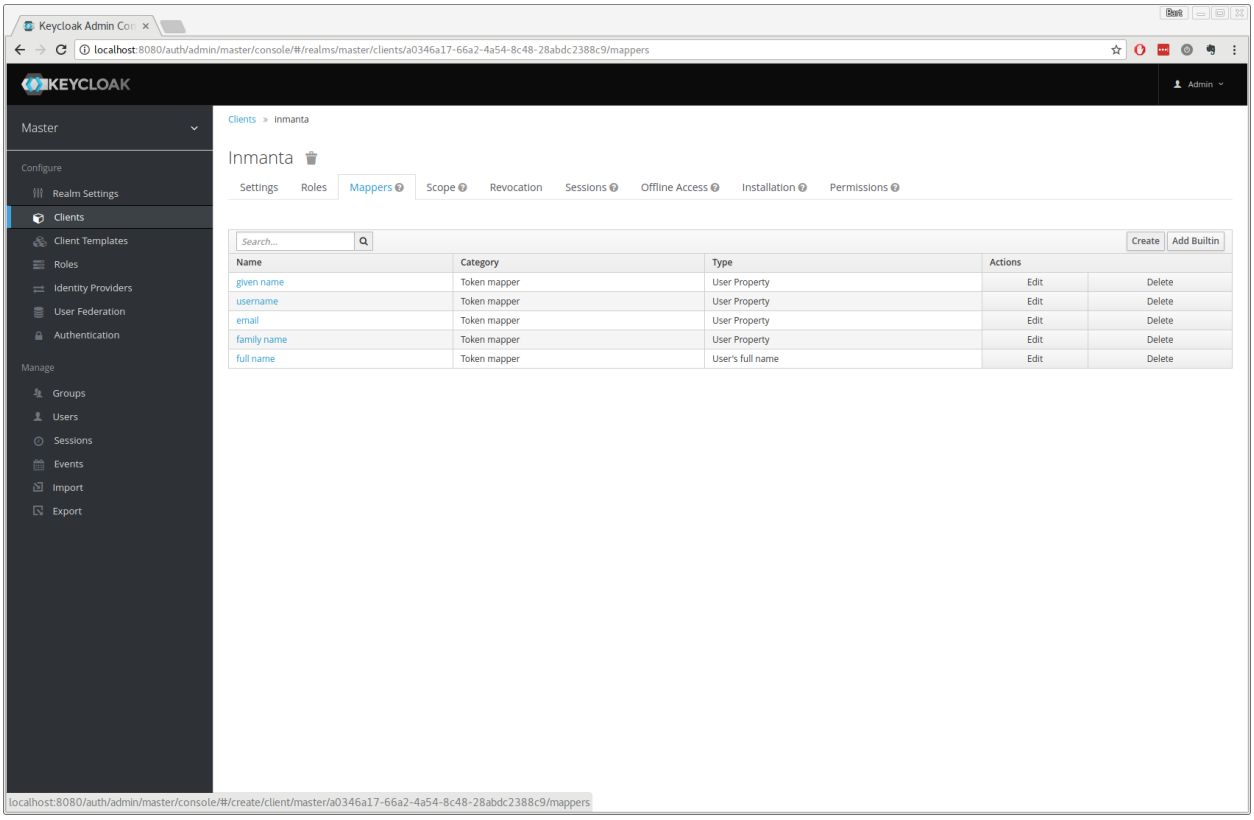


Fig. 8: Add a custom mapper to the client to include `:urn:inmanta:ct`

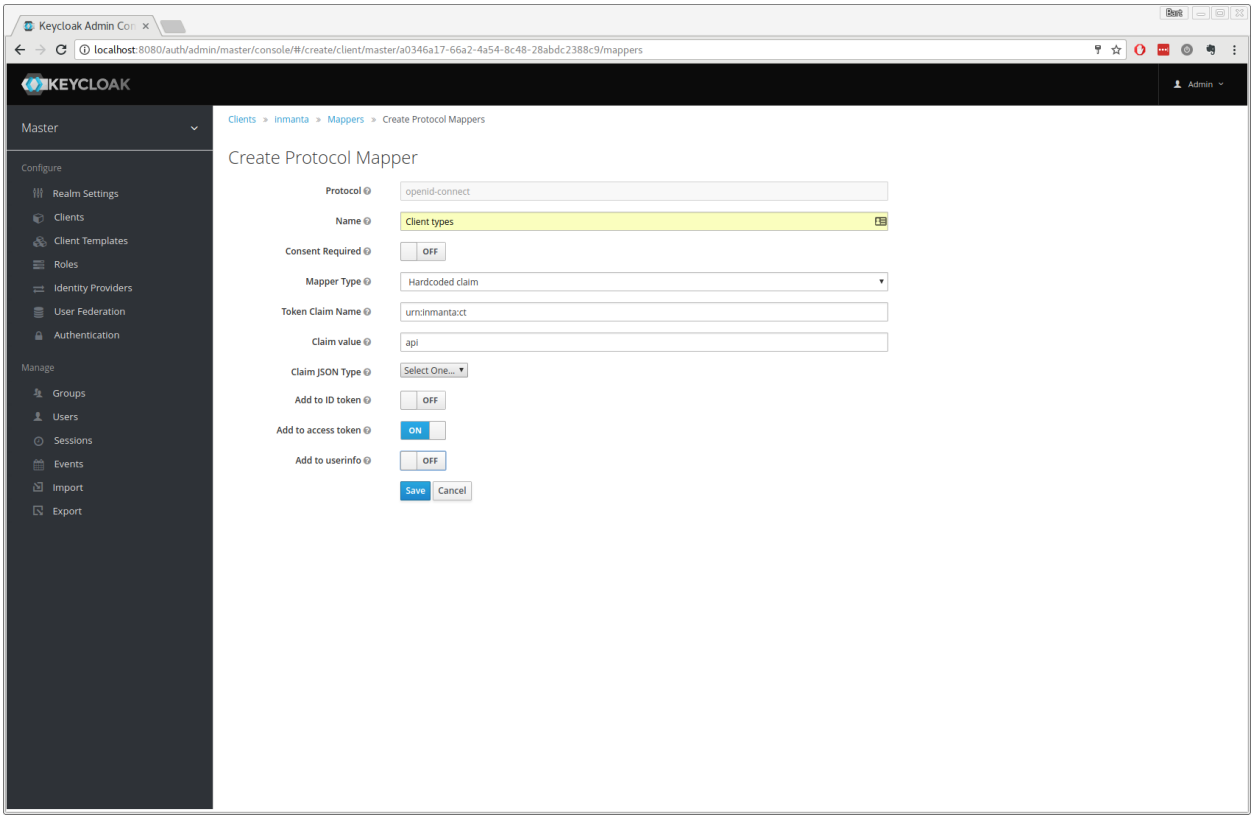


Fig. 9: Add the ct claim to all access tokens for this client.

Step 3: Configure inmanta server

Go to the installation tab and 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.

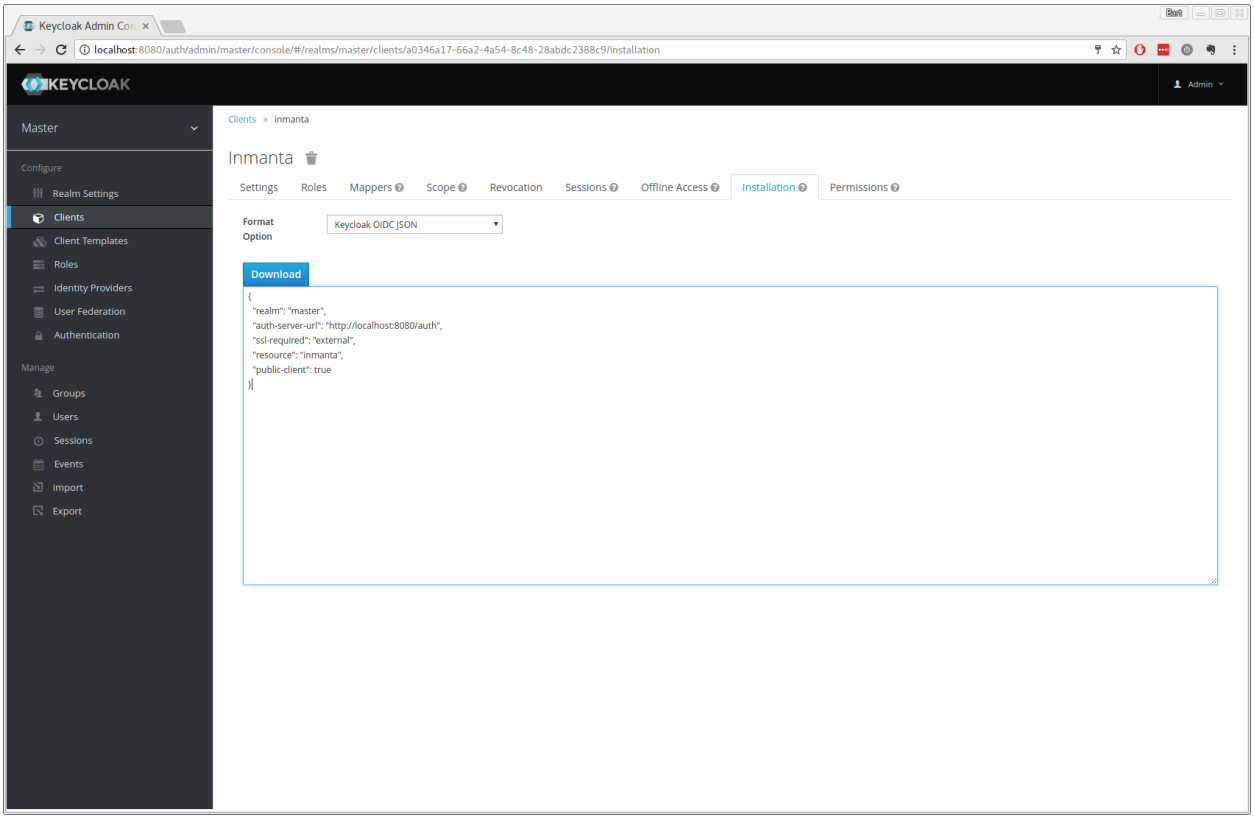


Fig. 10: Show the correct configuration parameters in JSON format.

Add the keycloak configuration parameters to the dashboard section of the server configuration file. Add a configuration file called */etc/inmanta/inmanta.d/keycloak.cfg*. Add the realm, auth_url and client_id to the dashboard section. Use the parameters from the installation json file created by keycloak.

```
[dashboard]
# keycloak specific configuration
realm=master
auth_url=http://localhost:8080/auth
client_id=inmanta
```

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.

```
[auth_jwt_keycloak]
algorithm=RS256
sign=false
client_types=api
issuer=http://localhost:8080/auth/realms/master
audience=inmanta
jwks_uri=http://localhost:8080/auth/realms/master/protocol/openid-connect/certs
```

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/auth/realms/master/.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.

9.4 Environment variables

Environment variables can be supplied to the Inmanta server and it's agents.

9.4.1 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"),
                               password=std::get_env("OS_PASSWORD"),
                               tenant="dev")
```

9.4.2 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.

9.5 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.

9.5.1 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 resources 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 produces 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.

9.5.2 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 inmanta-server
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.

9.6 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.

9.6.1 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
```

9.6.2 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/dashboards/10089

9.6.3 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.cpu	int	ns	The result of a small CPU benchmark, executed every second. Provides a baseline for machine performance.

CHAPTER

TEN

EXTENSIONS

10.1 Inmanta Web Console

The Inmanta Web Console is a web GUI for the Inmanta Service Orchestrator.

10.1.1 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.

10.1.2 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

ELEVEN

FREQUENTLY ASKED QUESTIONS

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.

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
```

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

TWELVE

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 environment is required, but often multiple environments of the same infrastructure are available such as development, integration and testing.

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.

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 That 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 is 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 idenfies 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

THIRTEEN

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.

13.1 Command Reference

All inmanta commands and services are started by the `inmanta` command. This page provides an overview of all subcommands available:

13.1.1 inmanta

```
usage: inmanta [-h] [-p] [-c CONFIG_FILE] [--config-dir CONFIG_DIR]
               [--log-file LOG_FILE] [--log-file-level LOG_FILE_LEVEL]
               [--timed-logs] [-v] [--warnings {warn,ignore,error}] [-X]
               [--version]
               {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
--log-file-level	Log level for messages going to the logfile: 0=ERROR, 1=WARNING, 2=INFO, 3=DEBUG Default: 2
--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 of the compiler. 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 subcomponents

Default: False

Sub-commands:

server

Start the inmanta server

```
inmanta server [-h]
```

agent

Start the inmanta agent

```
inmanta agent [-h]
```

compile

Compile the project to a configuration model

```
inmanta compile [-h] [-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

-e The environment to compile this model for

-X, --extended-errors Show stack traces for compile errors

Default: False

--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] [-m [MODULE]]
                {add,list,do,update,install,status,push,verify,commit,create,freeze,
↪build,v1tov2}
                ...
```

Named Arguments

-m, --module Module to apply this command to

subcommand

cmd Possible choices: add, list, do, update, install, status, push, verify, commit, create, freeze, build, v1tov2

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] [--v1] [--v2] [--override] module_req
```

Positional Arguments

module_req The name of the module, optionally with a version constraint.

Named Arguments

--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] [-r]
```

Named Arguments

-r (deprecated) Output a list of requires that can be included in project.yml

Default: False

do

Execute a command on all loaded modules

```
inmanta modules do [-h] command
```

Positional Arguments

command the command to execute

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 modules update [-h] [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

--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

install

Install a module in the active Python environment. Only works for v2 modules: v1 modules can only be installed in the context of a project.

This command might reinstall Python packages in the development venv if the currently installed versions are not compatible with the dependencies specified by the installed module.

Like *pip install*, this command does not reinstall a module for which the same version is already installed, except in editable mode.

```
inmanta modules install [-h] [-e] [path]
```

Positional Arguments

path The path to the module.

Named Arguments

-e, --editable Install in editable mode.

Default: False

status

Run a git status on all modules and report

```
inmanta modules status [-h]
```

push

Run a git push on all modules and report

```
inmanta modules push [-h]
```

verify

Verify dependencies and frozen module versions

```
inmanta modules verify [-h]
```

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] [--v1] name
```

Positional Arguments

name	The name of the module

Named Arguments

--v1	Create a v1 module. By default a v2 module is created. Default: False

freeze

Set all version numbers in project.yml

```
inmanta modules freeze [-h] [-o OUTFILE] [-r] [--operator {==,~=,>=}]
```

Named Arguments

-o, --outfile	File in which to put the new project.yml, default is the existing project.yml
-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 ~=

build

Build a Python package from a V2 module.

```
inmanta modules build [-h] [-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

-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]
```

project

Subcommand to manage the project

```
inmanta project [-h] {freeze,init,install,update} ...
```

subcommand

cmd Possible choices: freeze, init, install, update

Sub-commands:

freeze

Set all version numbers in project.yml

```
inmanta project freeze [-h] [-o OUTFILE] [-r] [--operator {==,~=,>=}]
```

Named Arguments

-o, --outfile File in which to put the new project.yml, default is the existing project.yml

-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] --name NAME [--output-dir OUTPUT_DIR] [--default]
```

Named Arguments

--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] [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

--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] [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

--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] [--dry-run] [-f MAIN_FILE] [--dashboard]
```

Named Arguments

--dry-run Only report changes

Default: False

-f Main file

Default: “main.cf”

--dashboard Start the dashboard and keep the server running. The server uses the current project as the source for server recompiles

Default: False

export

Export the configuration

```
inmanta export [-h] [-g] [-j JSON] [-e ENVIRONMENT] [-d] [--full] [-m]
               [--server_address SERVER] [--server_port PORT] [--token TOKEN]
               [--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]
               [--no-strict-deps-check] [--strict-deps-check]
```

Named Arguments

-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 Enable SSL

Default: False

--ssl-ca-cert Certificate authority for SSL

-X, --extended-errors Show stack traces for compile errors

Default: False

-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.

--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

13.1.2 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

-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
```

Arguments

ENVIRONMENT

Required argument

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

13.2 Configuration Reference

This document lists all options for the inmanta server and inmanta agent.

The options are listed per config section.

13.2.1 agent_rest_transport

host
: **Type** str

Default `localhost`

IP address or hostname of the server

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

13.2.2 client_rest_transport

host

Type str

Default localhost

IP address or hostname of the server

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

13.2.3 cmdline_rest_transport

host

Type str

Default `localhost`

IP address or hostname of the server

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

13.2.4 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.

13.2.5 compiler_rest_transport

host

Type str

Default `localhost`

IP address or hostname of the server

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

13.2.6 config

agent-deploy-interval

Type Time, the number of seconds represented as an integer value

Default `0`

The number of seconds between two (incremental) deployment runs of the agent. 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-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 `None`

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 $node-name

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

Default 600

The number of seconds between two repair runs (full deploy) of the agent. Set 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.

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.

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.

13.2.7 dashboard

auth-url

Type str

Default `None`

The auth url of the keycloak server to use.

client-id

Type str

Default `None`

The client id configured in keycloak for this application.

enabled

Type Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default `True`

Determines whether the server should host the dashboard or not

path

Type str

Default `/usr/share/inmanta/dashboard`

The path on the local file system where the dashboard can be found

realm

Type str

Default `inmanta`

The realm to use for keycloak authentication.

13.2.8 database

connection-pool-max-size

Type int

Default `10`

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

13.2.9 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

13.2.10 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

13.2.11 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

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.

available-versions-to-keep

Type int

Default `10`

[DEPRECATED: use AVAILABLE_VERSIONS_TO_KEEP environment setting] On boot and at regular intervals the server will purge older versions. This is the number of most recent versions to keep available.

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

delete-currupt-files

Type Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default `True`

The server logs an error when it detects a file got corrupted. When set to true, the server will also delete the file, so on subsequent compiles the missing file will be recreated.

enabled-extensions

Type List of comma-separated values

Default

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 *server.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

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

13.2.12 server_rest_transport

port

Type int

Default `8888`

[DEPRECATED USE *server.bind-port*] The port on which the server listens for connections

13.2.13 unknown_handler

default

Type str

Default `prune-agent`

default method to handle unknown values

13.2.14 web-console

enabled

Type Boolean value, represented as any of true, false, on, off, yes, no, 1, 0. (Case-insensitive)

Default `True`

Should the server should host the web-console or not

json-parser

Type str

Default `Native`

Whether the web-console should use the 'Native' or the 'BigInt' JSON Parser. 'BigInt' is useful when the web-console has to show very large integers (larger than 2^53 - 1).

path

Type str

Default `/usr/share/inmanta/web-console`

The path on the local file system where the web-console can be found

13.3 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 (e.g. *min hour dom month dow*). 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 int

Default 600

The deployment interval of the autostarted agents. 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_interval

Type int

Default 600

[DEPRECATED] Agent interval for autostarted agents in seconds

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 int

Default 86400

The repair interval of the autostarted agents. 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.

autostart_splay

Type int

Default 10

[DEPRECATED] Splay time for autostarted agents.

available_versions_to_keep

Type int

Default 100

The number of versions to keep stored in the database

environment_agent_trigger_method

Type enum: push_incremental_deploy, push_full_deploy

Default push_full_deploy

The agent trigger method to use. If push_on_auto_deploy is enabled, agent_trigger_method_on_auto_deploy overrides this setting

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, deleted or decommissioned.

purge_on_delete

Type bool

Default False

Enable purge on delete. When set to true, the server will detect the absence of resources with purge_on_delete set to true and automatically purges them.

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.

13.4 Compiler Configuration Reference

13.4.1 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: Optional[str] = None*, *freeze_recursive: bool = False*, *freeze_operator: inmanta.module.ConstrainedStrValue = '~='*, *author: Optional[str] = None*, *author_email: Optional[pydantic.networks.NameEmail] = None*, *license: Optional[str] = None*, *copyright: Optional[str] = None*, *modulepath: List[str] = []*, *repo: List[*inmanta.module.ModuleRepoInfo*] = []*, *downloadpath: Optional[str] = None*, *install_mode:* inmanta.module.InstallMode *= InstallMode.release*, *relation_precedence_policy: List[inmanta.module.ConstrainedStrValue] = []*, *strict_deps_check: bool = True*)

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) This key determines what version of a module should be selected when a module is downloaded. For more information see *InstallMode*.
- **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: When the type is set to package, the URL field should contains the URL of the Python package repository. The repository should be PEP 503 (the simple repository API) compliant.

  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 determined 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 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.

class inmanta.module.**ModuleRepoInfo**(*, *url: str*, *type:* inmanta.module.ModuleRepoType = *ModuleRepoType.git*)

Bases: pydantic.main.BaseModel

class inmanta.module.**ModuleRepoType**(*value*)

Bases: enum.Enum

An enumeration.

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
  - url: https://pypi.org/simple/
    type: package
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: ~=
```

13.4.2 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: Optional[str] = None, freeze_recursive: bool = False, freeze_operator: inmanta.module.ConstrainedStrValue = '~=', version: str, license: str*)

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
```

13.5 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.

13.5.1 Constants

class inmanta.const.**LogLevel**(*value*)

Bases: str, enum.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*)

Bases: str, enum.Enum

Enumeration of all resource actions.

deploy = 'deploy'

dryrun = 'dryrun'

getfact = 'getfact'

other = 'other'

pull = 'pull'

push = 'push'

store = 'store'

13.5.2 Compiler exceptions

class inmanta.ast.**CompilerException**(*msg: str*)

Bases: Exception, inmanta.ast.export.Exportable

Base class for exceptions generated by the compiler

class inmanta.parser.**ParserException**(*location: inmanta.ast.Range*, *value: object*, *msg: Optional[str] = None*)

Bases: *inmanta.ast.CompilerException*

Exception occurring during the parsing of the code

class inmanta.ast.**RuntimeException**(*stmt: Optional[inmanta.ast.Locatable]*, *msg: str*)

Bases: *inmanta.ast.CompilerException*

Baseclass for exceptions raised by the compiler after parsing is complete.

class inmanta.ast.**ExternalException**(*stmt: Optional[inmanta.ast.Locatable]*, *msg: str*, *cause: Exception*)

Bases: *inmanta.ast.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: Optional[Locatable]*, *msg: str*, *cause:* PluginException)

Bases: *inmanta.ast.ExternalException*

Base exception for wrapping an explicit *inmanta.plugins.PluginException* raised from a plugin call.

13.5.3 Plugins

class inmanta.plugins.**Context**(*resolver: inmanta.execute.runtime.Resolver*, *queue: inmanta.execute.runtime.QueueScheduler*, *owner: FunctionCall*, *plugin: Plugin*, *result: inmanta.execute.runtime.ResultVariable*)

An instance of this class is used to pass context to the plugin

emit_expression(*stmt: ExpressionStatement*) → None

Add a new statement

get_client() → inmanta.protocol.endpoints.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() → inmanta.execute.runtime.QueueScheduler

get_resolver() → inmanta.execute.runtime.Resolver

get_sync_client() → inmanta.protocol.endpoints.SyncClient

get_type(*name: inmanta.ast.LocatableString*) → *inmanta.ast.type.Type*

Get a type from the configuration model.

run_sync(*function: Callable[[...], inmanta.plugins.T]*, *timeout: int = 5*) → inmanta.plugins.T

Execute the async function and return its result. This method takes care of starting and stopping the ioloop. 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: Optional[Callable] = None*, *commands: Optional[List[str]] = 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[inmanta.plugins.Plugin]*) → None

Add a function plugin class

classmethod clear(*inmanta_module: Optional[str] = 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[inmanta.plugins.Plugin]]

Get all functions that are registered

13.5.4 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:* inmanta.resources.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*) → *inmanta.resources.Resource*

Create a clone of this resource. The given kwargs can be used to override attributes.

Returns The cloned resource

class inmanta.resources.**PurgeableResource**(*_id:* inmanta.resources.Id)

See *std::PurgeableResource* for more information.

class inmanta.resources.**ManagedResource**(*_id:* inmanta.resources.Id)

See *std::ManagedResource* for more information.

class inmanta.resources.**IgnoreResourceException**

Throw this exception when a resource should not be included by the exported.

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: Union[ResourceVersionIdStr, ResourceIdStr]*) → *inmanta.resources.Id*

Parse the resource id and return the type, the hostname and the resource identifier.

resource_str() → 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

13.5.5 Handlers

inmanta.agent.handler.**cache**(*func: Optional[inmanta.agent.handler.T_FUNC] = None*, *ignore: List[str] = []*, *timeout: int = 5000*, *for_version: bool = True*, *cache_none: bool = True*, *cacheNone: Optional[bool] = None*, *call_on_delete: Optional[Callable[[Any], None]] = None*) → Union[inmanta.agent.handler.T_FUNC, Callable[[inmanta.agent.handler.T_FUNC], inmanta.agent.handler.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 provides an implementation 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: inmanta.resources.Resource*, *dry_run: bool = False*, *action_id: Optional[uuid.UUID] = None*, *logger: Optional[logging.Logger] = None*)

Context passed to handler methods for state related "things"

add_change(*name: str*, *desired: object*, *current: Optional[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: Union[BaseModel, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.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

critical(*msg: str*, **args: object*, ***kwargs: object*) → None

Log 'msg % args' with severity 'CRITICAL'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

```
logger.critical("Houston, we have a %s", "major disaster", exc_info=1)
```

debug(*msg: str*, **args: object*, ***kwargs: object*) → None

Log 'msg % args' with severity 'DEBUG'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

Keyword arguments should be JSON serializable.

```
logger.debug("Houston, we have a %s", "thorny problem", exc_info=1)
```

error(*msg: str*, **args: object*, ***kwargs: object*) → None

Log 'msg % args' with severity 'ERROR'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

```
logger.error("Houston, we have a %s", "major problem", exc_info=1)
```

exception(*msg: str*, **args: object*, *exc_info: bool = True*, ***kwargs: object*) → None

Convenience method for logging an ERROR with exception information.

fields_updated(*fields: str*) → None

Report that fields have been updated

info(*msg: str*, **args: object*, ***kwargs: object*) → None

Log 'msg % args' with severity 'INFO'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

Keyword arguments should be JSON serializable.

```
logger.info("Houston, we have a %s", "interesting problem", exc_info=1)
```

is_dry_run() → bool

Is this a dryrun?

set_fact(*fact_id: str*, *value: str*) → None

Send a fact to the Inmanta server.

Parameters

- **fact_id** – The name of the fact.
- **value** – The actual value of the fact.

set_status(*status: inmanta.const.ResourceState*) → None

Set the status of the handler operation.

update_changes(*changes: Dict[str, inmanta.data.model.AttributeStateChange]*) → None

update_changes(*changes:* Dict*[str,* Dict*[str, Optional[SimpleTypes]]]*) → None

update_changes(*changes:* Dict*[str, Tuple[SimpleTypes, SimpleTypes]]*) → None

Update the changes list with changes

Parameters **changes** – This should be a dict with a value a dict containing "current" and "desired" keys

warning(*msg: str*, **args: object*, ***kwargs: object*) → None

Log 'msg % args' with severity 'WARNING'.

To pass exception information, use the keyword argument exc_info with a true value, e.g.

Keyword arguments should be JSON serializable.

```
logger.warning("Houston, we have a %s", "bit of a problem", exc_info=1)
```

class inmanta.agent.handler.**ResourceHandler**(*agent: inmanta.agent.agent.AgentInstance*, *io: Optional[IOBase] = None*)

A baseclass for classes that handle resources. New handler are registered with the *provider()* decorator.

The implementation of a handler should use the `self._io` instance to execute io operations. This io objects makes abstraction of local or remote operations. See *LocalIO* for the available operations.

Parameters

- **agent** – The agent that is executing this handler.
- **io** – The io object to use.

_diff(*current:* inmanta.resources.Resource, *desired:* inmanta.resources.Resource) → 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.

available(*resource:* inmanta.resources.Resource) → bool

Returns true if this handler is available for the given resource

Parameters **resource** – Is this handler available for the given resource?

Returns Available or not?

can_reload() → bool

Can this handler reload?

Returns Return true if this handler needs to reload on requires changes.

check_facts(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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 actually 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → *inmanta.resources.Resource*

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

deploy(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *requires:* *Dict[ResourceIdStr, inmanta.const.ResourceState]*) → None

This method is always be called by the agent, even when one of the requires of the given resource failed to deploy. The default implementation of this method will deploy the given resource when all its requires were deployed successfully. Override this method if a different condition determines whether the resource should deploy.

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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *changes: Dict[str, Dict[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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *dry_run: bool = False*) → None

Update the given resource. This method is called by the agent. Most handlers will not override this method and will only override *check_resource()*, optionally *list_changes()* and *do_changes()*

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to check the current state of.
- **dry_run** – True will only determine the required changes but will not execute them.

facts(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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() → inmanta.protocol.endpoints.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*) → Optional[bytes]

Retrieve a file from the fileserver identified with the given id. The convention is to use the sha1sum of the content to identify it.

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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → None

Method executed after an 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 to query facts for.

pre(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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 to query facts for.

run_sync(*func: Callable[[], Awaitable[inmanta.agent.handler.T]]*) → inmanta.agent.handler.T

Run a 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: inmanta.agent.cache.AgentCache*) → None

stat_file(*hash_id: str*) → bool

Check if a file exists on the server. This method does and async call to the server and blocks on the result.

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.

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.handler.**CRUDHandler**(*agent: inmanta.agent.agent.AgentInstance*, *io: Optional[IOBase] = 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:* inmanta.resources.Resource) → bool

Returns true if this handler is available for the given resource

Parameters **resource** – Is this handler available for the given resource?

Returns Available or not?

calculate_diff(*ctx:* inmanta.agent.handler.HandlerContext, *current:* inmanta.resources.Resource, *desired:* inmanta.resources.Resource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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 actually 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → *inmanta.resources.Resource*

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

create_resource(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.PurgeableResource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.PurgeableResource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *requires: Dict[ResourceIdStr, inmanta.const.ResourceState]*) → None

This method is always be called by the agent, even when one of the requires of the given resource failed to deploy. The default implementation of this method will deploy the given resource when all its requires were deployed successfully. Override this method if a different condition determines whether the resource should deploy.

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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *changes: Dict[str, Dict[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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *dry_run: Optional[bool] = None*) → None

Update the given resource. This method is called by the agent. Override the CRUD methods of this class.

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to check the current state of.
- **dry_run** – True will only determine the required changes but will not execute them.

facts(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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() → inmanta.protocol.endpoints.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*) → Optional[bytes]

Retrieve a file from the fileserver identified with the given id. The convention is to use the sha1sum of the content to identify it.

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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → None

Method executed after an 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 to query facts for.

pre(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource) → 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 to query facts for.

read_resource(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.PurgeableResource) → 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[inmanta.agent.handler.T]]*) → inmanta.agent.handler.T

Run a 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: inmanta.agent.cache.AgentCache*) → None

stat_file(*hash_id: str*) → bool

Check if a file exists on the server. This method does and async call to the server and blocks on the result.

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:* inmanta.agent.handler.HandlerContext, *changes: Dict[str, Dict[str, Any]]*, *resource:* inmanta.resources.PurgeableResource) → 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, Optional[str]]*)

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: Optional[str] = None*, *group: Optional[str] = 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, *Union*[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: Optional[Dict[str, str]] = None*, *cwd: Optional[str] = None*, *timeout: Optional[int] = 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

13.5.6 Export

@inmanta.export.**dependency_manager**(*function: Callable[[Dict[str, inmanta.ast.entity.Entity], Dict[*inmanta.resources.Id*, *inmanta.resources.Resource*]], None]*) → None

Register a function that manages dependencies in the configuration model that will be deployed.

13.5.7 Attributes

class inmanta.ast.attribute.**Attribute**(*entity: Entity*, *value_type: Type*, *name: str*, *location: inmanta.ast.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: inmanta.ast.Location*)

Bases: *inmanta.ast.attribute.Attribute*

An attribute that is a relation

13.5.8 Modules

class inmanta.module.**InstallMode**(*value*)

Bases: str, enum.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** prelease version.
- **For V2 modules: The python package was published on a Python package repository, the version identifier is** 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: Optional[pydantic.error_wrappers.ValidationError] = None*)

This exception is raised if the metadata file of a project or module is invalid.

class inmanta.module.**ModuleLike**(*path: str*)

Bases: abc.ABC, Generic[inmanta.module.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*) → Optional[*inmanta.module.ModuleLike*]

Get a concrete module like instance from a path. Returns None when no project or module is present at the given path.

property metadata: inmanta.module.TMetadata

class inmanta.module.**Module**(*project: Optional[*inmanta.module.Project*]*, *path: str*)

Bases: *inmanta.module.ModuleLike*[inmanta.module.TModuleMetadata], abc.ABC

This class models an inmanta configuration module

abstract classmethod from_path(*path: str*) → Optional[*inmanta.module.Module*]

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**

alias of str

class inmanta.module.**ModuleV1**(*project: Optional[*inmanta.module.Project*]*, *path: str*)

Bases: *inmanta.module.Module*[inmanta.module.ModuleV1Metadata], inmanta.module.ModuleLikeWithYmlMetadataFile

classmethod from_path(*path: str*) → Optional[inmanta.module.TModule]

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: Optional[*inmanta.module.Project*]*, *path: str*, *is_editable_install: bool = False*, *installed_version: Optional[packaging.version.Version] = None*)

Bases: *inmanta.module.Module*[inmanta.module.ModuleV2Metadata]

classmethod from_path(*path: str*) → Optional[inmanta.module.TModule]

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[inmanta.module.TModule]

get_installed_module(*project: Optional[*inmanta.module.Project*]*, *module_name: str*) → Optional[inmanta.module.TModule]

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: List[str]*)

Bases: *inmanta.module.ModuleSource*[*ModuleV2*]

inmanta.module.**Path**

alias of str

class inmanta.loader.**PluginModuleFinder**(*modulepaths: List[str]*)

Bases: importlib.abc.Finder

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: Optional[str] = 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.

13.5.9 Project

class inmanta.module.**Project**(*path: str*, *autostd: bool = True*, *main_file: str = 'main.cf'*, *venv_path: Optional[Union[str,* inmanta.env.VirtualEnv*]] = None*, *attach_cf_cache: bool = True*, *strict_deps_check: Optional[bool] = None*)

Bases: *inmanta.module.ModuleLike*[*inmanta.module.ProjectMetadata*], inmanta.module.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: Optional[bool] = None*) → *inmanta.module.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:* inmanta.module.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: *inmanta.ast.CompilerException*

This exception is raised when inmanta is unable to find a valid project

13.5.10 Python Environment

inmanta.env.**mock_process_env**(*, *python_path: Optional[str] = None*, *env_path: Optional[str] = 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.

13.5.11 Variables

class inmanta.ast.variables.**Reference**(*name: inmanta.ast.LocatableString*)

This class represents a reference to a value

Variables **name** – The name of the Reference as a string.

name

13.5.12 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() → *inmanta.ast.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() → Optional[str]

Returns the type string as expressed in the Inmanta *DSL*, if this type can be expressed in the *DSL*. Otherwise returns None.

validate(*value: Optional[object]*) → 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:* inmanta.ast.type.Type) → *inmanta.ast.type.Type*

Returns the type formed by replacing this type's base type with the supplied type.

class inmanta.ast.type.**NullableType**(*element_type:* inmanta.ast.type.Type)

Bases: *inmanta.ast.type.Type*

Represents a nullable type in the Inmanta *DSL*. For example *NullableType(Number())* represents *number?*.

class inmanta.ast.type.**Primitive**

Bases: *inmanta.ast.type.Type*

Abstract base class representing primitive types.

cast(*value: Optional[object]*) → 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: *inmanta.ast.type.Primitive*

This class represents an integer or float in the configuration model. On these numbers the following operations are supported:

+, -, /, *

class inmanta.ast.type.**Integer**

Bases: *inmanta.ast.type.Number*

An instance of this class represents the int type in the configuration model.

class inmanta.ast.type.**Bool**

Bases: *inmanta.ast.type.Primitive*

This class represents a simple boolean that can hold true or false.

class inmanta.ast.type.**String**

Bases: *inmanta.ast.type.Primitive*

This class represents a string type in the configuration model.

class inmanta.ast.type.**Union**(*types: List[*inmanta.ast.type.Type*]*)

Bases: *inmanta.ast.type.Type*

Instances of this class represent a union of multiple types.

class inmanta.ast.type.**Literal**

Bases: *inmanta.ast.type.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: *inmanta.ast.type.Type*

Instances of this class represent a list type containing any types of values.

class inmanta.ast.type.**TypedList**(*element_type:* inmanta.ast.type.Type)

Bases: *inmanta.ast.type.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: *inmanta.ast.type.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: *inmanta.ast.type.Type*

Instances of this class represent a dict type with any types of values.

class inmanta.ast.type.**TypedDict**(*element_type:* inmanta.ast.type.Type)

Bases: *inmanta.ast.type.Dict*

Instances of this class represent a dict type containing only values of type element_type.

class inmanta.ast.type.**LiteralDict**

Bases: *inmanta.ast.type.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: inmanta.ast.Namespace*, *name: str*)

Bases: inmanta.ast.type.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*.

13.5.13 Protocol

class inmanta.protocol.common.**Result**(*code: int = 0*, *result: Optional[Dict[str, Any]] = None*)

A result of a method call

code

The result code of the method call.

property result: Optional[*Dict*[str, Any]]

13.5.14 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.UUID*, *connection: Optional[asyncpg.connection.Connection] = None*) → Optional[inmanta.data.TBaseDocument]

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: Optional[str] = None*, *order: Optional[str] = None*, *limit: Optional[int] = None*, *offset: Optional[int] = None*, *no_obj: Optional[bool] = None*, *lock: Optional[inmanta.data.RowLockMode] = None*, *connection: Optional[asyncpg.connection.Connection] = None*, ***query: object*) → List[inmanta.data.TBaseDocument]

Get a list of documents matching the filter args

class inmanta.data.**Compile**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: *inmanta.data.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
- **environment_variables** – environment variables to be passed to the compiler
- **succes** – 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

async classmethod get_substitute_by_id(*compile_id: uuid.UUID*) → Optional[*inmanta.data.Compile*]

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() → inmanta.data.model.CompileRun

class inmanta.data.**ConfigurationModel**(***kwargs: object*)

Bases: *inmanta.data.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

async classmethod **get_versions**(*environment: uuid.UUID*, *start: int = 0*, *limit: int = 100000*) → List[*inmanta.data.ConfigurationModel*]

Get all versions for an environment ordered descending

class inmanta.data.**Environment**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: *inmanta.data.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
- **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: *inmanta.data.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: *inmanta.data.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
- **last_non_deploying_status** – The last status of this resource that is not the 'deploying' status.

async classmethod get_resources_for_version(*environment: uuid.UUID*, *version: int*, *agent: Optional[str] = None*, *no_obj: bool = False*, ***, *connection: Optional[asyncpg.connection.Connection] = None*) → List[*inmanta.data.Resource*]

class inmanta.data.**ResourceAction**(*from_postgres: bool = False*, ***kwargs: object*)

Bases: *inmanta.data.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.UUID*, *version: int*, *action: Optional[str] = None*, *limit: int = 0*) → List[*inmanta.data.ResourceAction*]

class inmanta.data.model.**BaseModel**

Bases: pydantic.main.BaseModel

Base class for all data objects in Inmanta

class Config

Pydantic config.

use_enum_values = True

inmanta.data.model.**ResourceIdStr**

The resource id without the version

alias of str

inmanta.data.model.**ResourceVersionIdStr**

The resource id with the version included.

alias of `str`

13.5.15 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*) → Union[None, str, Tuple[object, ...], int, float, bool, *inmanta.execute.proxy.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.

13.5.16 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.UUID*)

Clear all data from this environment.

Parameters **id** – The uuid of the environment.

Raises

- ***NotFound*** – The given environment doesn't exist.
- ***Forbidden*** – The given environment is protected.

inmanta.protocol.methods.**create_environment**(*project_id: uuid.UUID*, *name: str*, *repository: Optional[str] = None*, *branch: Optional[str] = None*, *environment_id: Optional[uuid.UUID] = None*)

Create a new environment

Parameters

- **project_id** – The id of the project this environment belongs to
- **name** – The name of the environment
- **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

inmanta.protocol.methods.**create_project**(*name: str*, *project_id: Optional[uuid.UUID] = None*)

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.**create_token**(*tid: uuid.UUID*, *client_types: list*, *idempotent: bool = True*)

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.**decomission_environment**(*id: uuid.UUID*, *metadata: Optional[dict] = None*)

Decommision an environment. This is done by uploading an empty model to the server and let purge_on_delete handle removal.

Parameters

- **id** – The uuid of the environment.
- **metadata** – Optional metadata associated with the decommissioning

Raises

- ***NotFound*** – The given environment doesn't exist.
- ***Forbidden*** – The given environment is protected.

inmanta.protocol.methods.**delete_environment**(*id: uuid.UUID*)

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.**delete_param**(*tid: uuid.UUID*, *id: str*, *resource_id: Optional[str] = None*)

Delete a parameter on the server

Parameters

- **tid** – The id of the environment
- **id** – The name of the parameter
- **resource_id** – The resource id of the parameter

inmanta.protocol.methods.**delete_project**(*id: uuid.UUID*)

Delete the given project and all related data

inmanta.protocol.methods.**delete_setting**(*tid: uuid.UUID*, *id: str*)

Delete a value

inmanta.protocol.methods.**delete_version**(*tid: uuid.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.UUID*, *agent_trigger_method: inmanta.const.AgentTriggerMethod = AgentTriggerMethod.push_full_deploy*, *agents: Optional[list] = 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**(*a: str*, *b: str*)

Returns the diff of the files with the two given ids

inmanta.protocol.methods.**do_dryrun**(*tid: uuid.UUID*, *id: uuid.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.UUID*, *version: Optional[int] = None*)

Create a list of dry runs

Parameters

- **tid** – The id of the environment
- **version** – Only for this version

inmanta.protocol.methods.**dryrun_report**(*tid: uuid.UUID*, *id: uuid.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.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.UUID*, *id: uuid.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.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.UUID*, *id: int*, *resource: str*)

Get the code for a given version of the configuration model

Parameters

- **tid** – The environment the code belongs to
- **id** – The id (version) of the configuration model

inmanta.protocol.methods.**get_compile_queue**(*tid: uuid.UUID*) → List[inmanta.data.model.CompileRun]

Get the current compiler queue on the server

inmanta.protocol.methods.**get_environment**(*id: uuid.UUID*, *versions: Optional[int] = None*, *resources: Optional[int] = None*)

Get an environment and all versions associated

Parameters

- **id** – The id of the environment to return
- **versions** – Include this many available version for this environment.
- **resources** – Include this many available resources for this 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.UUID*, *id: str*, *resource_id: Optional[str] = None*)

Get a parameter from the server.

Parameters

- **tid** – The id of the environment
- **id** – The name of the parameter
- **resource_id** – Optionally, 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.UUID*, *agent: str*, *resource: dict*)

Get all parameters/facts known by the agents for the given resource

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.UUID*)

Get a project and a list of the ids of all environments

inmanta.protocol.methods.**get_report**(*id: uuid.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.UUID*, *start: Optional[str] = None*, *end: Optional[str] = None*, *limit: Optional[int] = None*)

Return compile reports newer then start

Parameters

- **tid** – The id of the environment to get a report from
- **start** – Reports after start
- **end** – Reports before end
- **limit** – Maximum number of results, up to a maximum of 1000

inmanta.protocol.methods.**get_resource**(*tid: uuid.UUID*, *id: str*, *logs: Optional[bool] = None*, *status: Optional[bool] = None*, *log_action: Optional[inmanta.const.ResourceAction] = 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 id
- **logs** – Include the logs in the response
- **status** – Only return the status of the resource
- **log_action** – The log action to include, leave empty/none for all actions. Valid actions are one of the action strings in const.ResourceAction
- **log_limit** – 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()*)

inmanta.protocol.methods.**get_resources_for_agent**(*tid: uuid.UUID*, *agent: str*, *sid: Optional[uuid.UUID] = None*, *version: Optional[int] = None*, *incremental_deploy: bool = False*)

Return the most recent state for the resources associated with agent, or the version requested

Parameters

- **tid** – The id of the environment this resource belongs to
- **agent** – The agent
- **sid** – Session id of the agent (transparently added by agent client)
- **version** – 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** – Indicates whether the server should only return the resources that changed since the previous deployment.

inmanta.protocol.methods.**get_server_status**() → inmanta.data.model.StatusResponse

Get the status of the server

inmanta.protocol.methods.**get_setting**(*tid: uuid.UUID*, *id: str*)

Get a value

inmanta.protocol.methods.**get_state**(*tid: uuid.UUID*, *sid: uuid.UUID*, *agent: str*)

Get the state for this agent.

returns a map {

enabled: bool

}

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.UUID*, *id: int*, *include_logs: Optional[bool] = None*, *log_filter: Optional[str] = None*, *limit: Optional[int] = 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** – If true, a log of all operations on all resources is included
- **log_filter** – Filter log to only include actions of the specified type
- **limit** – 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()*)

inmanta.protocol.methods.**heartbeat**(*sid: uuid.UUID*, *tid: uuid.UUID*, *endpoint_names: list*, *nodename: str*, *no_hang: bool = False*)

Send a heartbeat to the server

Paran sid The session ID used by this agent at this moment

Parameters

- **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** – 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.UUID*, *reply_id: uuid.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.UUID*)

Is a compiler running for the given environment

Parameters **id** – The environment id

inmanta.protocol.methods.**list_agent_processes**(*environment: Optional[uuid.UUID] = None*, *expired: bool = True*, *start: Optional[uuid.UUID] = None*, *end: Optional[uuid.UUID] = None*, *limit: Optional[int] = None*)

Return a list of all nodes and the agents for these nodes

Parameters

- **environment** – An optional environment. If set, only the agents that belong to this environment are returned
- **expired** – Optional, also show expired processes, otherwise only living processes are shown.
- **start** – Agent processes after start (sorted by sid in ASC)
- **end** – Agent processes before end (sorted by sid in ASC)
- **limit** – Maximum number of results, up to a maximum of 1000

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.UUID*, *start: Optional[str] = None*, *end: Optional[str] = None*, *limit: Optional[int] = None*)

List all agent for an environment

Parameters

- **tid** – The environment the agents are defined in
- **start** – Agent after start (sorted by name in ASC)
- **end** – Agent before end (sorted by name in ASC)
- **limit** – Maximum number of results, up to a maximum of 1000

Raises

- *BadRequest* – limit parameter can not exceed 1000
- *NotFound* – The given environment id does not exist!

inmanta.protocol.methods.**list_environments**()

Create a list of environments

inmanta.protocol.methods.**list_params**(*tid: uuid.UUID*, *query: dict = {}*)

List/query parameters in this environment

Parameters

- **tid** – The id of the environment
- **query** – A query to match against metadata

inmanta.protocol.methods.**list_projects**()

Create a list of projects

inmanta.protocol.methods.**list_settings**(*tid: uuid.UUID*)

List the settings in the current environment

inmanta.protocol.methods.**list_versions**(*tid: uuid.UUID*, *start: Optional[int] = None*, *limit: Optional[int] = None*)

Returns a list of all available versions

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.

inmanta.protocol.methods.**modify_environment**(*id: uuid.UUID*, *name: str*, *repository: Optional[str] = None*, *branch: Optional[str] = None*)

Modify the given environment

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

inmanta.protocol.methods.**modify_project**(*id: uuid.UUID*, *name: str*)

Modify the given project

inmanta.protocol.methods.**notify_change**(*id: uuid.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** – Update the model code and modules. Default value is true
- **metadata** – The metadata that indicates the source of the compilation trigger.

inmanta.protocol.methods.**notify_change_get**(*id: uuid.UUID*, *update: bool = True*)

Simplified GET version of the POST method

inmanta.protocol.methods.**put_version**(*tid: uuid.UUID*, *version: int*, *resources: list*, *resource_state: dict = {}*, *unknowns: Optional[list] = None*, *version_info: Optional[dict] = None*, *compiler_version: Optional[str] = None*, *resource_sets: Dict[ResourceIdStr, Optional[str]] = {}*)

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** – A list of unknown parameters that caused the model to be incomplete
- **version_info** – Module version information
- **compiler_version** – version of the compiler, if not provided, this call will return an error
- **resource_sets** – a dictionary describing which resource belongs to which resource set

inmanta.protocol.methods.**release_version**(*tid: uuid.UUID*, *id: int*, *push: bool = False*, *agent_trigger_method: Optional[inmanta.const.AgentTriggerMethod] = 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** – Indicates whether the agents should perform a full or an incremental deploy when push is true.

inmanta.protocol.methods.**resource_action_update**(*tid: uuid.UUID*, *resource_ids: list*, *action_id: uuid.UUID*, *action:* inmanta.const.ResourceAction, *started: Optional[datetime.datetime] = None*, *finished: Optional[datetime.datetime] = None*, *status: Optional[Union[inmanta.const.ResourceState, inmanta.const.DeprecatedResourceState]] = None*, *messages: list = []*, *changes: dict = {}*, *change: Optional[inmanta.const.Change] = 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** – The timestamp when this action was started. When this action (action_id) has not been saved yet, started has to be defined.
- **finished** – 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** – The current status of the resource (if known)
- **messages** – A list of log entries to add to this entry.

:param change:s 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.

Parameters

- **change** – The result of the changes
- **send_events** – [DEPRECATED] The value of this field is not used anymore.

inmanta.protocol.methods.**resource_event**(*tid: uuid.UUID*, *id: str*, *resource: str*, *send_events: bool*, *state: inmanta.const.ResourceState*, *change: inmanta.const.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** – The changes made to the resource

inmanta.protocol.methods.**set_param**(*tid: uuid.UUID*, *id: str*, *source: inmanta.const.ParameterSource*, *value: str*, *resource_id: Optional[str] = None*, *metadata: dict = {}*, *recompile: bool = False*)

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
- **id** – The name of the parameter
- **resource_id** – Optionally, scope the parameter to resource (fact)
- **source** – The source of the parameter.
- **value** – The value of the parameter
- **metadata** – metadata about the parameter
- **recompile** – Whether to trigger a recompile

inmanta.protocol.methods.**set_parameters**(*tid: uuid.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) - metadata metadata about the parameter

inmanta.protocol.methods.**set_setting**(*tid: uuid.UUID*, *id: str*, *value: Union[uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str, Dict[str, Any]]*)

Set a value

inmanta.protocol.methods.**set_state**(*agent: str*, *enabled: bool*)

Set the state of the agent.

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 id to check

Returns A list of files that do not exist.

inmanta.protocol.methods.**trigger**(*tid: uuid.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.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.UUID*, *id: int*, *resources: dict*)

Upload the supporting code to the server

Parameters

- **tid** – The environment the code belongs to
- **id** – The id (version) of the configuration model
- **resource** – a dict mapping resources to dicts mapping file names to file hashes

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.**agent_action**(*tid: uuid.UUID*, *name: str*, *action: inmanta.const.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.UUID*, *action: inmanta.const.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.UUID*, *id: uuid.UUID*) → inmanta.data.model.CompileDetails

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.**dryrun_trigger**(*tid: uuid.UUID*, *version: int*) → uuid.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.UUID*) → None

Clear all data from this environment.

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.UUID*, *name: str*, *repository: Optional[str] = None*, *branch: Optional[str] = None*, *environment_id: Optional[uuid.UUID] = None*, *description: str = ''*, *icon: str = ''*) → inmanta.data.model.Environment

Create a new environment

Parameters

- **project_id** – The id of the project this environment belongs to
- **name** – The name of the environment
- **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.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_decommission**(*id: uuid.UUID*, *metadata: Optional[inmanta.data.model.ModelMetadata] = None*) → int

Decommission an environment. This is done by uploading an empty model to the server and let purge_on_delete handle removal.

Parameters

- **id** – The uuid of the environment.
- **metadata** – Optional metadata associated with the decommissioning

Raises

- *NotFound* – The given environment doesn't exist.
- *Forbidden* – The given environment is protected.

inmanta.protocol.methods_v2.**environment_delete**(*id: uuid.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.UUID*, *details: bool = False*) → inmanta.data.model.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[inmanta.data.model.Environment]

Returns a list of environments :param details: Whether to include the icon and description of the environments in the results

inmanta.protocol.methods_v2.**environment_modify**(*id: uuid.UUID*, *name: str*, *repository: Optional[str] = None*, *branch: Optional[str] = None*, *project_id: Optional[uuid.UUID] = None*, *description: Optional[str] = None*, *icon: Optional[str] = None*) → inmanta.data.model.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.UUID*, *id: str*) → inmanta.protocol.common.ReturnValue[None]

Delete a value

inmanta.protocol.methods_v2.**environment_setting_get**(*tid: uuid.UUID*, *id: str*) → inmanta.data.model.EnvironmentSettingsReponse

Get a value

inmanta.protocol.methods_v2.**environment_settings_list**(*tid: uuid.UUID*) → inmanta.data.model.EnvironmentSettingsReponse

List the settings in the current environment

inmanta.protocol.methods_v2.**environment_settings_set**(*tid: uuid.UUID*, *id: str*, *value: Union[inmanta.types.StrictNonIntBool, int, float, str, Dict[str, Union[str, int, inmanta.types.StrictNonIntBool]]]*) → inmanta.protocol.common.ReturnValue[None]

Set a value

inmanta.protocol.methods_v2.**get_agent_process_details**(*tid: uuid.UUID*, *id: uuid.UUID*, *report: bool = False*) → inmanta.data.model.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.UUID*, *limit: Optional[int] = None*, *start: Optional[Union[datetime.datetime, bool, str]] = None*, *end: Optional[Union[datetime.datetime, bool, str]] = None*, *first_id: Optional[str] = None*, *last_id: Optional[str] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'name.asc'*) → List[inmanta.data.model.Agent]

Get all of the agents in the given environment :param tid: The id of the environment the agents should belong to :param limit: Limit the number of agents that are returned :param start: The lower limit for the order by column (exclusive). :param 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

Parameters

- **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.UUID*, *limit: Optional[int] = None*, *first_id: Optional[uuid.UUID] = None*, *last_id: Optional[uuid.UUID] = None*, *start: Optional[str] = None*, *end: Optional[str] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'name.asc'*) → List[inmanta.data.model.Fact]

List the facts in an environment :param tid: The id of the environment :param limit: Limit the number of facts that are returned :param 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

Parameters

- **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
- **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: Optional[inmanta.const.ApiDocsFormat] = ApiDocsFormat.swagger*) → inmanta.protocol.common.ReturnValue[Union[inmanta.protocol.openapi.model.Op str]]

Get the OpenAPI definition of the API :param 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.UUID*) → Optional[*inmanta.data.model.CompileData*]

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.UUID*, *limit: Optional[int] = None*, *first_id: Optional[uuid.UUID] = None*, *last_id: Optional[uuid.UUID] = None*, *start: Optional[datetime.datetime] = None*, *end: Optional[datetime.datetime] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'requested.desc'*) → List[inmanta.data.model.CompileReport]

Get the compile reports from an environment :param tid: The id of the environment :param limit: Limit the number of instances that are returned :param 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

Parameters

- **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.UUID*, *from_version: int*, *to_version: int*) → List[inmanta.data.model.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.UUID*, *version: int*, *report_id: uuid.UUID*) → inmanta.data.model.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_fact**(*tid: uuid.UUID*, *rid: ResourceIdStr*, *id: uuid.UUID*) → inmanta.data.model.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.UUID*, *rid: ResourceIdStr*) → List[inmanta.data.model.Fact]

Get the facts related to a specific resource :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.UUID*, *notification_id: uuid.UUID*) → inmanta.data.model.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.UUID*, *limit: Optional[int] = None*, *first_id: Optional[uuid.UUID] = None*, *last_id: Optional[uuid.UUID] = None*, *start: Optional[Union[datetime.datetime, str]] = None*, *end: Optional[Union[datetime.datetime, str]] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'name.asc'*) → List[inmanta.data.model.Parameter]

List the parameters in an environment :param tid: The id of the environment :param limit: Limit the number of parameters that are returned :param 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

Parameters

- **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_resource_actions**(*tid: uuid.UUID*, *resource_type: Optional[str] = None*, *agent: Optional[str] = None*, *attribute: Optional[str] = None*, *attribute_value: Optional[str] = None*, *log_severity: Optional[str] = None*, *limit: Optional[int] = 0*, *action_id: Optional[uuid.UUID] = None*, *first_timestamp: Optional[datetime.datetime] = None*, *last_timestamp: Optional[datetime.datetime] = None*) → inmanta.protocol.common.ReturnValue[List[inmanta.data.model.Resou

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

Returns the list of matching Resource Actions in a descending order according to the 'started' timestamp. If a limit was specified, also return the links to the next and previous pages. The "next"

page always refers to the 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.UUID*, *rvid: ResourceVersionIdStr*) → Dict[ResourceIdStr, List[inmanta.data.model.ResourceAction]]

Return relevant events for a resource, i.e. all deploy actions for each of its dependencies since this resources' last 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.

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.

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.UUID*, *version: int*, *limit: Optional[int] = None*, *first_id: Optional[ResourceVersionIdStr] = None*, *last_id: Optional[ResourceVersionIdStr] = None*, *start: Optional[str] = None*, *end: Optional[str] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'resource_type.desc'*) → List[inmanta.data.model.VersionedResource]

Get the resources that belong to a specific version :param tid: The id of the environment :param version: The version number :param limit: Limit the number of resources that are returned :param 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

Parameters

- **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.UUID*, *version: int*, *resource_type: str*) → List[inmanta.data.model.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.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.UUID*, *limit: Optional[int] = None*, *start: Optional[int] = None*, *end: Optional[int] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'version.desc'*) → List[inmanta.data.model.DesiredStateVersion]

Get the desired state versions from an environment :param tid: The id of the environment :param limit: Limit the number of versions that are returned :param start: The lower limit for the order by column (exclusive).

Only one of 'start' and 'end' should be specified at the same time.

Parameters

- **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.UUID*, *version: int*) → List[inmanta.data.model.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.UUID*, *limit: Optional[int] = None*, *first_id: Optional[uuid.UUID] = None*, *last_id: Optional[uuid.UUID] = None*, *start: Optional[datetime.datetime] = None*, *end: Optional[datetime.datetime] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'created.desc'*) → List[inmanta.data.model.Notification]

List the notifications in an environment :param tid: The id of the environment :param limit: Limit the number of notifications that are returned :param 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

Parameters

- **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.**project_create**(*name: str*, *project_id: Optional[uuid.UUID] = None*) → inmanta.data.model.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.UUID*) → None

Delete the given project and all related data

inmanta.protocol.methods_v2.**project_get**(*id: uuid.UUID*, *environment_details: bool = False*) → inmanta.data.model.Project

Get a project and a list of the environments under this project :param 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[inmanta.data.model.Project]

Returns a list of projects :param environment_details: Whether to include the icon and description of the environments in the results

inmanta.protocol.methods_v2.**project_modify**(*id: uuid.UUID*, *name: str*) → inmanta.data.model.Project

Modify the given project

inmanta.protocol.methods_v2.**promote_desired_state_version**(*tid: uuid.UUID*, *version: int*, *trigger_method: Optional[inmanta.data.model.PromoteTriggerMethod] = None*) → None

Promote a desired state version, making it the active version in the environment :param tid: The id of the environment :param version: The number of the version to promote :param 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.UUID*, *resource_state: typing.Optional[typing.Dict[ResourceIdStr, typing.Literal[<ResourceState.available: 'available'>, <ResourceState.undefined: 'undefined'>]]] = None*, *unknowns: typing.Optional[typing.List[typing.Dict[str, typing.Union[uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]]]] = None*, *resource_sets: typing.Optional[typing.Dict[ResourceIdStr, typing.Optional[str]]] = None*, *removed_resource_sets: typing.Optional[typing.List[str]] = None*, ***kwargs: object*) → 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

Returns The newly stored version number.

inmanta.protocol.methods_v2.**reserve_version**(*tid: uuid.UUID*) → int

Reserve a version number in this environment.

inmanta.protocol.methods_v2.**resource_deploy_done**(*tid: uuid.UUID*, *rvid: ResourceVersionIdStr*, *action_id: uuid.UUID*, *status: inmanta.const.ResourceState*, *messages: List[inmanta.data.model.LogLine] = []*, *changes: Dict[str, inmanta.data.model.AttributeStateChange] = {}*, *change: Optional[inmanta.const.Change] = 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.UUID*, *rvid: ResourceVersionIdStr*, *action_id: uuid.UUID*) → Dict[ResourceVersionIdStr, inmanta.const.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.UUID*, *rid: ResourceIdStr*) → inmanta.data.model.ReleasedResourceDetails

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.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.UUID*, *rid: ResourceIdStr*, *limit: Optional[int] = None*, *first_id: Optional[str] = None*, *last_id: Optional[str] = None*, *start: Optional[datetime.datetime] = None*, *end: Optional[datetime.datetime] = None*, *sort: str = 'date.desc'*) → List[inmanta.data.model.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.UUID*, *limit: Optional[int] = None*, *first_id: Optional[ResourceVersionIdStr] = None*, *last_id: Optional[ResourceVersionIdStr] = None*, *start: Optional[str] = None*, *end: Optional[str] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'resource_type.desc'*, *deploy_summary: bool = False*) → List[inmanta.data.model.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.UUID*, *rid: ResourceIdStr*, *limit: Optional[int] = None*, *start: Optional[datetime.datetime] = None*, *end: Optional[datetime.datetime] = None*, *filter: Optional[Dict[str, List[str]]] = None*, *sort: str = 'timestamp.desc'*) → List[inmanta.data.model.ResourceLog]

Get the logs of a specific resource :param tid: The id of the environment this resource belongs to :param rid: The id of the resource :param limit: Limit the number of instances that are returned :param start: The lower limit for the order by column (exclusive).

Only one of 'start' and 'end' should be specified at the same time.

Parameters

- **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). The 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.**resume_environment**(*tid: uuid.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.**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.UUID*, *notification_id: uuid.UUID*, *read: Optional[bool] = None*, *cleared: Optional[bool] = None*) → inmanta.data.model.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.UUID*, *version: int*, *rid: ResourceIdStr*) → inmanta.data.model.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

13.6 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[*inmanta.ast.export.Error*]*)

Bases: *inmanta.data.model.BaseModel*

Top level structure of compiler data to be exported.

errors: *List*[*inmanta.ast.export.Error*]

All errors occurred while trying to compile.

class inmanta.ast.export.**Error**(*, *category:* inmanta.ast.export.ErrorCategory = *ErrorCategory.runtime*, *type: str*, *message: str*, *location: Optional[*inmanta.ast.export.Location*]* = *None*, ***extra_data: Any*)

Bases: `pydantic.main.BaseModel`

Error occurred while trying to compile.

category: *inmanta.ast.export.ErrorCategory*

Category of this error.

location: Optional[*inmanta.ast.export.Location*]

Location where this error occurred.

message: str

Error message.

type: str

Fully qualified name of the actual exception.

class inmanta.ast.export.**ErrorCategory**(*value*)

Bases: str, enum.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:* inmanta.ast.export.Range)

Bases: pydantic.main.BaseModel

Location in a file. Based on the LSP spec 3.15

range: *inmanta.ast.export.Range*

uri: str

class inmanta.ast.export.**Range**(*, *start:* inmanta.ast.export.Position, *end:* inmanta.ast.export.Position)

Bases: pydantic.main.BaseModel

Range in a file. Based on the LSP spec 3.15

end: *inmanta.ast.export.Position*

start: *inmanta.ast.export.Position*

class inmanta.ast.export.**Position**(*, *line: int*, *character: int*)

Bases: pydantic.main.BaseModel

Position in a file. Based on the LSP spec 3.15

character: int

line: int

13.7 Inmanta modules

13.7.1 Module apache

- License: Apache 2.0
- Version: 0.5.8
- Upstream project: https://github.com/inmanta/apache.git

Entities

entity apache::Server

Parents: *web::ApplicationContainer*

An apache server

The following implementations are defined for this entity:

- *apache::apacheServerRPM*
- *apache::apacheServerDEB*
- *apache::patchhttp2*

The following implements statements select implementations for this entity:

- *apache::apacheServerRPM*, *apache::patchhttp2* constraint `(std::familyof(host.os, 'fedora') and (host.os.version == 23))`
- *apache::apacheServerRPM* constraint `(std::familyof(host.os,'rhel') or (std::familyof(host.os,'fedora') and ((not host.os.version) == 23)))`
- *apache::apacheServerDEB* constraint `std::familyof(host.os,'ubuntu')`

Implementations

implementation apache::apacheServerDEB

implementation apache::apacheServerRPM

implementation apache::appImplDEB

implementation apache::appImplRPM

implementation apache::patchhttp2

13.7.2 Module apt

- License: Apache 2.0
- Version: 0.4.15
- 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

TODO: add latest support

- Handler name apt
- Handler for entity *std::Package*

13.7.3 Module aws

- License: Apache 2.0
- Version: 3.2.3
- 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)$/

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 number listen_port=80

attribute number 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::Host

Parents: *aws::VMAttributes*, *ip::Host*

A subclass of ip::Host that creates a virtual machine on AWS.

attribute bool install_agent=false

relation aws::VirtualMachine vm [1]

relation aws::Provider provider [1]

relation ssh::Key public_key [1]

relation ip::IP public_ip [0:1]

relation ip::IP private_ip [1]

relation aws::Subnet subnet [0:1]

relation aws::SecurityGroup security_groups [0:*]

The following implementations are defined for this entity:

- *aws::awsHost*

The following implements statements select implementations for this entity:

- *std::hostDefaults*, *aws::awsHost* constraint `true`
- *aws::userData* constraint `install_agent`

entity `aws::IPrule`

Parents: *aws::SecurityRule*

attribute ip::cidr `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 ip::cidr `destination`

The destination route

attribute ip::ip `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 number `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 number `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 ip::protocol `ip_protocol`

The type of ip protocol to allow. Currently this support tcp/udp/icmp/sctp or all

attribute ip::port `port_min`=0

attribute ip::port `port_max`=0

attribute ip::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 ip::cidr 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: *platform::UserdataVM*

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 install_agent=false

attribute bool ignore_extra_volumes=false

attribute bool ignore_wrong_image=false

attribute number 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 ip::cidr 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`=Dict()

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`
- *aws::userData* constraint `install_agent`

entity `aws::Volume`

Parents: *aws::AWSResource*

attribute string `name`

attribute string attachmentpoint='/dev/sdb'

attribute string availability_zone

attribute bool encrypted=false

attribute number size=10

attribute string volume_type='gp2'

attribute dict tags=Dict()

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 ip::port port=3306

attribute bool public=false

attribute dict tags=Dict()

The following implements statements select implementations for this entity:

- *std::none* constraint true

Implementations

implementation aws::agentConfig

implementation aws::awsHost

implementation aws::req

implementation aws::userData

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*

13.7.4 Module cron

- License: Apache 2.0
- Version: 1.0.6
- Upstream project: https://github.com/inmanta/cron.git

Typedefs

typedef cron::cronjob_name

- Base type string
- Type constraint /^[a-zA-Z0-9]+$/

Entities

entity cron::Cronjob

Parents: *std::PurgeableResource*

attribute cron::cronjob_name **name**

The name of the cronjob.

attribute string **user**

Command will be executed with this user.

attribute string **schedule**

A cron expression indicating when the command should be executed.

attribute string **command**

The command executed when the cronjob fires.

attribute dict **env_vars**=Dict()

The environment variables that should be available to the command being executed.

relation std::Host host [1]

other end: std::Host.cronjobs [0:*]

The following implementations are defined for this entity:

- *cron::cronjob*

The following implements statements select implementations for this entity:

- *cron::cronjob* constraint true

Implementations

implementation cron::cronjob

13.7.5 Module drupal

- License: Apache 2.0
- Version: 0.7.9
- Upstream project: https://github.com/inmanta/drupal.git

Entities

entity drupal::Application

Parents: *php::Application*

A single drupal application.

attribute string admin_user

attribute string admin_password

attribute string admin_email

attribute string site_name

attribute bool run_install=true

relation mysql::Database database [1]

relation exec::Run _exec [1]

The following implementations are defined for this entity:

- *drupal::installer*
- *drupal::noInstaller*
- *drupal::drupalSiteRPM*
- *drupal::drupalSiteDEB*

The following implements statements select implementations for this entity:

- *drupal::installer* constraint run_install
- *drupal::noInstaller* constraint (not run_install)

- *drupal::drupalSiteRPM*, *php::phpApacheRPM*, *apache::appImplRPM* constraint (std::familyof(container.host.os,'rhel') or std::familyof(container.host.os, 'fedora'))
- *drupal::drupalSiteDEB*, *php::phpApacheDEB*, *apache::appImplDEB* constraint std::familyof(container.host.os,'ubuntu')

Implementations

implementation drupal::drupalSiteDEB

implementation drupal::drupalSiteRPM

implementation drupal::installer

implementation drupal::noInstaller

13.7.6 Module exec

- License: Apache 2.0
- Version: 1.1.12
- 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=Dict()

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*)

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*

13.7.7 Module graph

- License: Apache V2
- Version: 0.8.9
- Upstream project: https://github.com/inmanta/graph.git

Entities

entity graph::ClassDiagram

Parents: *std::Entity*

Create a class diagram of a given module expression

attribute string name

The name of the graph, this is used to determine the name of the resulting image file

attribute string[] moduleexpression
: List of regexes matching module names

attribute string header=''
: file header for plantuml file

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity graph::Graph
: Parents: *std::Entity*

Create a graph with the given name and the grap definition in config

attribute string name
: The name of the graph, this is used to determine the name of the resulting image file

attribute string config
: The definition used to generate the graph

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

13.7.8 Module ip

- License: Apache 2.0
- Version: 1.2.11
- This module requires compiler version 2016.5 or higher
- Upstream project: https://github.com/inmanta/ip.git

Typedefs

typedef ip::cidr

- Base type `string`
- Type constraint `(ip::is_valid_cidr(self) == true)`

typedef ip::cidr_v10

- Base type `string`
- Type constraint `(ip::is_valid_cidr_v10(self) == true)`

typedef ip::cidr_v6

- Base type `string`
- Type constraint `(ip::is_valid_cidr_v6(self) == true)`

typedef ip::ip

- Base type `string`
- Type constraint `(ip::is_valid_ip(self) == true)`

typedef ip::ip_v10

- Base type string
- Type constraint (ip::is_valid_ip_v10(self) == true)

typedef ip::ip_v6

- Base type string
- Type constraint (ip::is_valid_ip_v6(self) == true)

typedef ip::mask

- Base type string
- Type constraint (ip::is_valid_netmask(self) == true)

typedef ip::port

- Base type number
- Type constraint ((self >= 0) and (self < 65536))

typedef ip::protocol

- Base type string
- Type constraint (((((self == 'tcp') or (self == 'udp')) or (self == 'icmp')) or (self == 'sctp')) or (self == 'all'))

Entities

entity ip::Address

Parents: *ip::Alias*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::Alias

Parents: *ip::IP*

attribute ip::ip netmask='0.0.0.0'

attribute number alias=0

attribute bool dhcp=false

relation ip::services::Server server [0:*]

other end: *ip::services::Server.ips [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::DstService

Parents: *ip::Service*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::Host

Parents: *std::Host*

A host that has an ip attribute for easy ip address access in the configuration model.

attribute ip::ip ip

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 ip::port remote_port=22

The remote port for this remote agent to use.

relation ip::services::Server servers [0:*]

other end: *ip::services::Server.host [1]*

relation ip::services::Client clients [0:*]

other end: *ip::services::Client.host [1]*

The following implements statements select implementations for this entity:

- *std::hostDefaults* constraint true

entity ip::IP

Parents: *std::Entity*

Base class for all ip addresses

attribute ip::ip v4='0.0.0.0'

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::Network

Parents: *std::Entity*

A network in this infrastructure.

attribute string network

attribute string netmask

attribute string name

attribute bool dhcp

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::Port

Parents: *ip::PortRange*

attribute ip::port high=0

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::PortRange

Parents: *std::Entity*

attribute ip::port low

attribute ip::port high

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::Service

Parents: *std::Entity*

Define a service as a protocol and a source and destination port range

attribute ip::protocol proto

relation ip::PortRange dst_range [0:*]

relation ip::PortRange src_range [0:*]

relation ip::services::BaseServer listening_servers [0:*]

other end: *ip::services::BaseServer.services [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::BaseClient

Parents: *std::Entity*

Base client class that connects to a server

relation ip::services::BaseServer servers [0:*]

other end: *ip::services::BaseServer.clients [0:*]*

entity ip::services::BaseServer

Parents: *std::Entity*

Base class for servers that accept connections from clients

relation ip::Service services [0:*]

other end: *ip::Service.listening_servers [0:*]*

relation ip::services::BaseClient clients [0:*]

other end: *ip::services::BaseClient.servers [0:*]*

entity ip::services::Client

Parents: *ip::services::BaseClient*

This interface models a client that is linked to a host

relation ip::Host host [1]

other end: *ip::Host.clients [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::Server

Parents: *ip::services::BaseServer*

This interface models a server that accepts connections from a client

relation ip::Host host [1]

other end: *ip::Host.servers [0:*]*

relation ip::Alias ips [0:*]

other end: *ip::Alias.server [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::VirtualClient

Parents: *ip::services::BaseClient*, *ip::services::VirtualSide*

This interface models a virtual client. It can for example represent all clients that exist on the internet.

attribute string name

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::VirtualHost

Parents: *ip::services::VirtualScope*

An address represented by a hostname

attribute std::hoststring hostname

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::VirtualIp

Parents: *ip::services::VirtualScope*

Only one ip

attribute ip::ip address

entity ip::services::VirtualNetwork

Parents: *ip::services::VirtualScope*

Define a virtual network segment

attribute ip::ip network

attribute ip::ip netmask

entity ip::services::VirtualRange

Parents: *ip::services::VirtualScope*

A range defined by from/to

attribute ip::ip from

attribute ip::ip to

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity ip::services::VirtualScope

Parents: *std::Entity*

This interface represents a scope to determine what a virtual client or server is.

relation ip::services::VirtualSide side [0:*]

other end: *ip::services::VirtualSide.scope [0:*]*

entity ip::services::VirtualServer

Parents: *ip::services::BaseServer*, *ip::services::VirtualSide*

Same as VirtualClient but then for a server

attribute string name

entity ip::services::VirtualSide

Parents: *std::Entity*

A base class for a virtual server or client

relation ip::services::VirtualScope scope [0:*]

other end: *ip::services::VirtualScope.side [0:*]*

Implementations

implementation ip::agentConfig

Plugins

ip.**add**(*addr: ip::ip_v10*, *n: number*) → ip::ip_v10

Add a number to the given ip.

ip.**cidr_to_network**(*cidr: string*) → string

Given cidr return the network address

ip.**concat**(*host: std::hoststring*, *domain: std::hoststring*) → std::hoststring

Concat host and domain

ip.**hostname**(*fqdn: string*) → string

Return the hostname part of the fqdn

ip.**ipindex**(*addr: ip::cidr_v10*, *position: number*) → string

Return the address at position in the network.

ip.**ipnet**(*addr: ip::cidr_v10*, *what: string*) → string

Return the ip, prefixlen, netmask or network address of the CIDR

Parameters

- **addr** – CIDR
- **what** – The required result:

- ip: The IP address of *addr* object.
- prefixlen: The prefix length of *addr* object.
- netmask: The subnet mask of *addr* object.
- network: The network address of *addr* object.

For instance:

std::print(ipnet("192.168.1.100/24", "ip")) –> 192.168.1.100
std::print(ipnet("192.168.1.100/24", "prefixlen")) –> 24
std::print(ipnet("192.168.1.100/24", "netmask")) –> 255.255.255.0
std::print(ipnet("192.168.1.100/24", "network")) –> 192.168.1.0

ip.**is_valid_cidr**(*addr: string*) → bool

ip.**is_valid_cidr_v10**(*addr: string*) → bool

Validate if the string matches a v6 or a v4 network in CIDR notation

ip.**is_valid_cidr_v6**(*addr: string*) → bool

ip.**is_valid_ip**(*addr: string*) → bool

ip.**is_valid_ip_v10**(*addr: string*) → bool

Validate if the string matches a v6 or v4 address

ip.**is_valid_ip_v6**(*addr: string*) → bool

ip.**is_valid_netmask**(*netmask: string*) → bool

Validate if the string matches a netmask

ip.**net_to_nm**(*network_addr: string*) → string

ip.**netmask**(*cidr: number*) → ip::ip

Given the cidr, return the netmask

ip.**network**(*ip: ip::ip*, *cidr: string*) → string

Given the ip and the cidr, return the network address

13.7.9 Module mysql

- License: Apache 2.0
- Version: 0.6.8
- This module requires compiler version 2017.2 or higher
- Upstream project: https://github.com/inmanta/mysql.git

Entities

entity mysql::DBMS

Parents: *std::Entity*

A DB management system (a service on a machina, DBaaS, ...)

attribute string hostref

reference to host, e.g. ip or hostname

attribute ip::port port=3306

relation mysql::Database databases [0:*]

other end: *mysql::Database.server [1]*

entity mysql::Database

Parents: *std::Entity*

attribute string name

attribute string user

attribute string password

attribute string encoding='utf8'

attribute string collation='utf8_general_ci'

relation mysql::DBMS server [1]

other end: *mysql::DBMS.databases [0:*]*

The following implementations are defined for this entity:

- *mysql::dBDependsOnServer*

The following implements statements select implementations for this entity:

- *mysql::dBDependsOnServer* constraint true

entity mysql::ManagedMysql

Parents: *mysql::DBMS*

attribute string user

attribute string password

relation ip::Host agenthost [1]

The following implementations are defined for this entity:

- *mysql::manageManaged*

The following implements statements select implementations for this entity:

- *mysql::manageManaged* constraint true

entity mysql::Server

Parents: *ip::services::Server*, *mysql::DBMS*

Mysql server configuration

attribute bool remove_anon_users=false

Required when trying to connect to a mysql database on the same host over it' external IP.

relation std::Service _svc [1]

The following implementations are defined for this entity:

- *mysql::removeAnonUsers*
- *mysql::ports*
- *mysql::mysqlRedhat*
- *mysql::mysqlMariaDB*
- *mysql::ubuntuMysql*

The following implements statements select implementations for this entity:

- *mysql::removeAnonUsers* constraint `(remove_anon_users == true)`
- *mysql::ports* constraint `true`
- *mysql::mysqlRedhat* constraint `(std::familyof(host.os,'rhel') and (host.os.version <= 6))`
- *mysql::mysqlMariaDB* constraint `((std::familyof(host.os,'rhel') and (host.os.version >= 7)) or std::familyof(host.os,'fedora'))`
- *mysql::ubuntuMysql* constraint `std::familyof(host.os,'ubuntu')`

Implementations

implementation mysql::dBDependsOnServer

implementation mysql::manageManaged

implementation mysql::mysqlMariaDB

implementation mysql::mysqlRedhat

implementation mysql::ports

implementation mysql::removeAnonUsers

implementation mysql::ubuntuMysql

13.7.10 Module net

- License: Apache 2.0
- Version: 1.0.13
- This module requires compiler version 2020.1 or higher
- Upstream project: https://github.com/inmanta/net.git

Typedefs

typedef net::mac_addr

- Base type string
- Type constraint (std::validate_type('pydantic.constr',self,Dict()) == true)

typedef net::vlan_id

- Base type int
- Type constraint (std::validate_type('pydantic.conint',self,Dict()) == true)

Entities

entity net::Interface

Parents: *std::Entity*

This interface models an ethernet network interface.

attribute net::mac_addr mac=''

attribute string name

attribute number mtu=1500

attribute bool vlan=false

relation std::Host host [1]

Host ethernet interface are always placed inside a host

other end: *std::Host.ifaces [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

13.7.11 Module openstack

- License: Apache 2.0
- Version: 3.8.1
- This module requires compiler version 2020.2 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::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 ip::cidr address

The address range that is allowed on this port (network interface)

attribute net::mac_addr? mac

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 number ram

Memory in MB for the flavor

attribute number vcpus

Number of VCPUs for the flavor

attribute number 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 number ephemeral=0

Ephemeral disk size in GB

attribute number swap=0

Swap space in MB

attribute number rxtx_factor=1.0

RX/TX factor

attribute bool is_public=true

Whether the flavor is publicly visible

attribute dict extra_specs=Dict()

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 ip::ip 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: *ip::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`

- *openstack::userData* constraint `install_agent`

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 number 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 number 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 number 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 ip::cidr 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`=Dict()

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 number `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 ip::ip 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 ip::cidr destination

attribute ip::ip 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 number 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 number 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 ip::protocol ip_protocol

The type of ip protocol to allow. Currently this support tcp/udp/icmp/sctp or all

attribute ip::port port_min=0

attribute ip::port port_max=0

attribute ip::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 ip::cidr `network_address`

attribute bool `dhcp`

attribute string `name`

attribute string `allocation_start`=''

attribute string `allocation_end`=''

attribute ip::ip[] `dns_servers`=List()

attribute ip::ip? `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: *platform::UserdataVM*

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=Dict()

A dict of metadata items

attribute dict personality=Dict()

A dict of files (personality)

attribute bool config_drive=false

Attach a configuration drive to the vm

attribute bool install_agent=false

Create a script and pass it as user_data to install the inmanta agent at boot time.

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
- *openstack::userData* constraint install_agent

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

implementation openstack::userData

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*

13.7.12 Module param

- License: Apache 2.0
- Version: 0.6.13
- This module requires compiler version 2018.2 or higher
- Upstream project: https://github.com/inmanta/param.git

Typedefs

typedef `param::email`

- Base type `string`
- Type constraint `/[^@]+@[^@]+\.[^@]+/`

Plugins

`param.`**report**(*name: string*, *value: string*)

This plugin reports a parameter to the server from the compile process. This can be used for *output* like parameter like in HEAT or TOSCA templates.

The dashboard will explicitly show these values as well.

Parameters

- **name** – The name/label of the value
- **value** – The value to report.

13.7.13 Module php

- License: Apache 2.0
- Version: 0.3.7
- Upstream project: https://github.com/inmanta/php.git

Entities

entity `php::Application`

Parents: *web::Application*

A web application that requires PHP

attribute bool `php55w`=false

The following implementations are defined for this entity:

- *php::phpApacheRPM*
- *php::php55el*
- *php::phpApacheDEB*

The following implements statements select implementations for this entity:

- *php::phpApacheRPM* constraint `(std::familyof(host.os,'redhat') and (php55w == false))`

- *php::phpApacheDEB* constraint `std::familyof(host.os,'ubuntu')`
- *php::php55el* constraint `(std::familyof(host.os,'redhat') and (php55w == true))`

Implementations

implementation php::php55el

This modules installs a common set of php modules and support for webservers either through a plugin or a cgi like interface.

implementation php::phpApacheDEB

This modules installs a common set of php modules and support for webservers either through a plugin or a cgi like interface.

implementation php::phpApacheRPM

This modules installs a common set of php modules and support for webservers either through a plugin or a cgi like interface.

13.7.14 Module platform

- License: ASL 2.0
- Version: 1.1.6
- This module requires compiler version 2019.1 or higher
- Upstream project: https://github.com/inmanta/platform.git

Entities

entity platform::UserdataBootstrap

Parents: *std::Entity*

Bootstrap an inmanta agent on the host by passing a shell script to the virtual machine user data. Setting the INMANTA_RELEASE environment variable to dev will install the agent from development snapshots.

The user script will force the correct hostname and setenforce 0 to disable enforcing selinux.

> **Warning:** Currently this script only support centos 7 or equivalent (rhel7, aws linux, sl7, ...), Ubuntu and Fedora.

relation platform::UserdataVM vm [1]

The following implementations are defined for this entity:

- *platform::userdataBootstrap*

The following implements statements select implementations for this entity:

- *platform::userdataBootstrap* constraint `true`

entity platform::UserdataVM

Parents: *std::Entity*

Base class for virtual machines that provide a user_data attribute through which a shell script can be injected at first boot of the virtual machine.

attribute string user_data
A shell script that is executed at first boot.

Implementations

implementation platform::userdataBootstrap

13.7.15 Module postgresql

- License: Apache 2.0
- Version: 0.3.1
- Upstream project: https://github.com/inmanta/postgresql.git

Typedefs

typedef postgresql::username_t

- Base type string
- Type constraint /[a-z0-9]*/

Entities

entity postgresql::Database
Parents: *std::PurgeableResource*

attribute string db_name

relation postgresql::PostgresqlServer server [1]
other end: *postgresql::PostgresqlServer.databases [0:*]*

relation postgresql::User owner [1]
other end: *postgresql::User.databases [0:*]*

The following implementations are defined for this entity:

- *postgresql::db_requires*

The following implements statements select implementations for this entity:

- *postgresql::db_requires* constraint true

entity postgresql::PostgresqlServer
Parents: *ip::services::Server*

attribute bool managed=true

attribute int log_min_duration_statement=-1

attribute bool pg_stat_statements=false

relation postgresql::Database databases [0:*]
other end: *postgresql::Database.server [1]*

relation postgresql::User users [0:*]

other end: *postgresql::User.server [1]*

relation std::Entity _packages [0:*]

internal Wait point: do Something(requires=_packages) to wait for all packages to be installed

The following implementations are defined for this entity:

- *postgresql::install*
- *postgresql::postgresqlServer*

The following implements statements select implementations for this entity:

- *postgresql::postgresqlServer*, *postgresql::install* constraint `managed`
- *std::none* constraint `(not managed)`

entity postgresql::PostgresqlTools

Parents: *std::Entity*

Install the postgresql client tools on a host.

relation std::Host host [1]

The following implementations are defined for this entity:

- *postgresql::install_tools*

The following implements statements select implementations for this entity:

- *postgresql::install_tools* constraint `true`

entity postgresql::User

Parents: *std::PurgeableResource*

attribute postgresql::username_t username

attribute string password

attribute string[] from=List()

network location this user is allowed to connect from

relation postgresql::PostgresqlServer server [1]

other end: *postgresql::PostgresqlServer.users [0:*]*

relation postgresql::Database databases [0:*]

other end: *postgresql::Database.owner [1]*

The following implementations are defined for this entity:

- *postgresql::user_requires*

The following implements statements select implementations for this entity:

- *postgresql::user_requires* constraint `true`

entity postgresql::ha::Master

Parents: *postgresql::PostgresqlServer*

attribute string synchronous_standby_names='inmanta'

attribute string replication_slot_name='replication'

attribute string replication_user='replication'

attribute string replication_user_password

attribute string synchronous_commit='off'

relation postgresql::ha::Standby standby [1]
other end: *postgresql::ha::Standby.master [1]*

relation postgresql::ha::ReplicationSlot replication_slot [1]
other end: *postgresql::ha::ReplicationSlot.server [1]*

The following implementations are defined for this entity:

- *postgresql::ha::postgresqlMaster*

The following implements statements select implementations for this entity:

- *postgresql::install*, *postgresql::ha::postgresqlMaster* constraint true

entity postgresql::ha::ReplicationSlot
Parents: *std::PurgeableResource*

relation postgresql::ha::Master server [1]
other end: *postgresql::ha::Master.replication_slot [1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity postgresql::ha::Standby
Parents: *postgresql::PostgresqlServer*

relation postgresql::ha::Master master [1]
other end: *postgresql::ha::Master.standby [1]*

The following implementations are defined for this entity:

- *postgresql::ha::postgresqlStandby*

The following implements statements select implementations for this entity:

- *postgresql::install*, *postgresql::ha::postgresqlStandby* constraint true

Implementations

implementation postgresql::db_requires

implementation postgresql::install

implementation postgresql::install_tools

implementation postgresql::postgresqlServer

implementation postgresql::user_requires

implementation postgresql::ha::postgresqlMaster

implementation postgresql::ha::postgresqlStandby

Resources

class postgresql.resources.**Database**

- Resource for entity *postgresql::Database*
- Id attribute db_name
- Agent name server.host.name
- Handlers *postgresql.resources.DatabaseProvider*

class postgresql.resources.**User**

- Resource for entity *postgresql::User*
- Id attribute username
- Agent name server.host.name
- Handlers *postgresql.resources.UserProvider*

class postgresql.resources.**ReplicationSlot**

- Resource for entity *postgresql::ha::ReplicationSlot*
- Id attribute replication_user
- Agent name server.host.name
- Handlers *postgresql.resources.ReplicationSlotProvider*

Handlers

class postgresql.resources.**DatabaseProvider**

- Handler name postgresql-database
- Handler for entity *postgresql::Database*

class postgresql.resources.**UserProvider**

- Handler name postgresql-user
- Handler for entity *postgresql::User*

class postgresql.resources.**ReplicationSlotProvider**

- Handler name postgresql-user
- Handler for entity *postgresql::ha::ReplicationSlot*

13.7.16 Module redhat

- License: Apache 2.0
- Version: 0.9.13
- Upstream project: https://github.com/inmanta/redhat.git

Implementations

implementation redhat::epel::epel7

implementation redhat::network::aliasImpl

This module implements the configuration for network interfaces on rhel

implementation redhat::network::config

This is the configuration each redhat based operating system should have

implementation redhat::network::ifaceImpl

implementation redhat::scl::epel7

13.7.17 Module rest

- License: Apache 2.0
- Version: 0.2.11
- 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=Dict()

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*

13.7.18 Module ssh

- License: Apache 2.0
- Version: 0.6.11
- 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

entity ssh::Server

Parents: *ip::services::Server*

A ssh server

The following implementations are defined for this entity:

- *ssh::sshServer*

The following implements statements select implementations for this entity:

- *ssh::sshServer* constraint `true`

Implementations

implementation ssh::sshServer

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

13.7.19 Module std

- License: Apache 2.0
- Version: 3.1.5
- This module requires compiler version 2020.8 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,Dict()) == true)`

typedef std::any_http_url

- Base type `string`
- Type constraint `(std::validate_type('pydantic.AnyHttpUrl',self) == true)`

typedef std::any_url

- Base type `string`
- Type constraint `(std::validate_type('pydantic.AnyUrl',self) == true)`

typedef std::ascii_word

- Base type string
- Type constraint (std::validate_type('pydantic.constr',self,Dict()) == true)

typedef std::base64

- Base type string
- Type constraint (std::is_base64_encoded(self) == true)

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) == true)

typedef std::datetime

- Base type string
- Type constraint (std::validate_type('datetime.datetime',self) == true)

typedef std::email_str

- Base type string
- Type constraint (std::validate_type('pydantic.EmailStr',self) == true)

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) == true)

typedef std::ipv4_address

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv4Address',self) == true)

typedef std::ipv4_interface

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv4Interface',self) == true)

typedef std::ipv4_network

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv4Network',self) == true)

typedef std::ipv6_address

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv6Address',self) == true)

typedef std::ipv6_interface

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv6Interface',self) == true)

typedef std::ipv6_network

- Base type string
- Type constraint (std::validate_type('ipaddress.IPv6Network',self) == true)

typedef std::ipv_any_address

- Base type string
- Type constraint (std::validate_type('pydantic.IPvAnyAddress',self) == true)

typedef std::ipv_any_interface

- Base type string
- Type constraint (std::validate_type('pydantic.IPvAnyInterface',self) == true)

typedef std::ipv_any_network

- Base type string
- Type constraint (std::validate_type('pydantic.IPvAnyNetwork',self) == true)

typedef std::name_email

- Base type string
- Type constraint (std::validate_type('pydantic.NameEmail',self) == true)

typedef std::negative_float

- Base type number
- Type constraint (std::validate_type('pydantic.NegativeFloat',self) == true)

typedef std::negative_int

- Base type int
- Type constraint (std::validate_type('pydantic.NegativeInt',self) == true)

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::positive_float`

- Base type `number`
- Type constraint `(std::validate_type('pydantic.PositiveFloat',self) == true)`

typedef `std::positive_int`

- Base type `int`
- Type constraint `(std::validate_type('pydantic.PositiveInt',self) == true)`

typedef `std::printable_ascii`

- Base type `string`
- Type constraint `(std::validate_type('pydantic.constr',self,Dict()) == 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) == true)`

typedef `std::uuid`

- Base type `string`
- Type constraint `(std::validate_type('uuid.UUID',self) == true)`

Entities

entity `std::AgentConfig`

Parents: *std::PurgeableResource*

Control agent settings. Currently these settings are only applied to autostarted agents

attribute bool? `autostart`

When this flag is set to true, the resource will be exported and set the agent map on the orchestrator. When false (or not set), this instance is ignore 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. Altough 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 number `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 number `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 number `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::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 number 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 of computer in the managed infrastructure

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:*]*

relation net::Interface ifaces [0:*]

Host ethernet interface are always placed inside a host

other end: *net::Interface.host [1]*

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:

- *redhat::scl::epel7*
- *redhat::network::config*
- *redhat::epel::epel7*
- *ip::agentConfig*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`
- *redhat::scl::epel7* constraint `(std::familyof(host.os,'rhel') and (host.os.version >= 7))`
- *redhat::network::config* constraint `std::familyof(host.os,'redhat')`
- *redhat::epel::epel7* constraint `(std::familyof(host.os,'rhel') and (host.os.version >= 7))`
- *ip::agentConfig* constraint `(host.ip is defined 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 mutuable set of strings.

Example

```
attr_a = std::MutableString()
if some_condition:
    attr_a.value = "a"
```

(continues on next page)

(continued from previous page)

```
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

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 wether 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`

Implementations

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.**assert**(*expression: bool*, *message: string*=)

Raise assertion error if expression is false

std.**at**(*objects: list*, *index: number*) → 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*) → number

Returns the number of elements in this list

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*)

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: number = 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: number = None*) → number

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.**inlineif**(*conditional: bool*, *a: any*, *b: any*) → any

An inline if

std.**invert**(*value: bool*) → bool

Invert a boolean value

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_set**(*obj: any*, *attribute: 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: number*) → 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.**length**(*value: string*) → number

Return the length of the string

std.**list_files**(*path: string*) → list

List files in a directory

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.**print**(*message: 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: number*, *start: number = 0*, *offset: number = 0*) → list

Return a sequence of i numbers, starting from zero or start if supplied.

std.**server_ca**() → string

std.**server_port**() → number

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*)

Execute the template in path in the current context. This function will generate a new statement that has dependencies on the used variables.

std.**timestamp**(*dummy: any = None*) → number

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: number = 20*) → string

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())
- **pydantic.stricturl:** min_length: int = 1 max_length: int = 2 ** 16 tld_required: bool = True allowed_schemes: Optional[Set[str]] = None

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*

Handlers

class std.resources.**YumPackage**

A Package handler that uses yum

- Handler name `yum`
- Handler for entity *std::Package*

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

TODO: add recursive operations

- 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*

13.7.20 Module terraform

- License: ASL 2.0
- Version: 1.3.0
- This module requires compiler version 2019.3 or higher
- Upstream project: https://github.com/inmanta/terraform.git

Typedefs

typedef terraform::config::nesting_mode_t

- Base type string
- Type constraint (self in ['set','list','dict','single'])

Entities

entity terraform::Provider

Parents: *std::Entity*

A Terraform provider

attribute string namespace

Organization in the terraform registry, packaging this provider

attribute string type

The provider type (e.g. "aws", "vsphere", "local")

attribute string version='latest'

The version of the provider to use, setting it to null will get the latest one

attribute string alias=''

An alias to differentiate this provider from other providers with the same binary but different config

attribute dict config

The config to apply to this provider

attribute bool manual_config=true

Whether the user wishes to provide the config as a dict, if false the config should be provided as a config block entity tree via the root_config relation.

attribute bool auto_agent=true

Whether to start an agent automatically or not. If set to false the relation agent_config should be set manually. :rel agent_config: This needs to be set only if auto_agent=false

relation std::AgentConfig agent_config [1]

Relation to the agent config

relation terraform::config::Block root_config [0:1]

Relation to the root configuration, or null if manual_config is true

The following implementations are defined for this entity:

- *terraform::agentConfig*
- *terraform::providerManualConfig*
- *terraform::providerBlockConfig*

The following implements statements select implementations for this entity:

- *terraform::providerManualConfig* constraint self.manual_config
- *terraform::providerBlockConfig* constraint (not self.manual_config)
- *terraform::agentConfig* constraint auto_agent

entity terraform::Resource

Parents: *std::PurgeableResource*

A Terraform resource

attribute string type

The type of resource this is

attribute string name

An arbitrary name to identify this resource

attribute string? terraform_id=null

If this is set, and the resource state is not stored in parameter yet, the handler will first try to import it, using the provided value as terraform id. :rel provider: The terraform provider for this resource

attribute dict config

The configuration for this resource

attribute bool manual_config=true

Whether the user wishes to provide the config as a dict, if false the config should be provided as a config block entity tree via the root_config relation.

attribute bool purge_on_delete=false

relation terraform::Provider provider [1]

Relation to the resource provider

relation terraform::config::Block root_config [0:1]

Relation to the root configuration, or null if manual_config is true

The following implementations are defined for this entity:

- *terraform::resourceManualConfig*
- *terraform::resourceBlockConfig*

The following implements statements select implementations for this entity:

- *terraform::resourceManualConfig* constraint self.manual_config
- *terraform::resourceBlockConfig* constraint (not self.manual_config)

entity terraform::config::Block

Parents: *std::Entity*

This entity represents a block of attributes in a terraform module. It can be used for anyone using the module to build a config. It is being used by the generator to generate the config of a resource or provider.

Source for the schema: https://github.com/inmanta/inmanta-tfplugin/blob/7269bc7d28d751b5dc110161dae29a6209c3fb63/docs/tf_grpc_plugin/proto/inmanta_tfplugin/tfplugin5.proto

tfplugin5.proto: L81: message Block { L82: int64 version = 1; L83: repeated Attribute attributes = 2; L84: repeated NestedBlock block_types = 3; L85: string description = 4; L86: StringKind description_kind = 5; L87: bool deprecated = 6; L88: }

attribute string? name=null

The name of this config section in the parent config block. Should be left null for the root config block.

attribute dict attributes

A dictionary of attributes. The key is the attribute name as specified in the terraform provider schema. The value is the value assigned to this attribute in the corresponding inmanta entity.

attribute bool deprecated=false

If true, will raise a warning everytime the configuration block is used.

attribute terraform::config::nesting_mode_t nesting_mode='single'

attribute string key

The key, required for list and dict nesting mode, automatically set otherwise.

attribute dict _config

Generated, the serialized version of this config.

attribute dict _state

Generated, the current state for the resource attached to this config block. The state here matches the same element as this config block. (The config should be a subset of the state if there is not change between last compile config and current compile config, in which case the config will still be unknown). i.e. The following configuration structure can be constructed with entities. .. code-block:: terraform::config::Block(name=null, attributes={"name": "Albert"}, children=[terraform::config::Block(name="children", attributes={"name": "Bob", "age": 12}, nesting_mode="set",), terraform::config::Block(name="children", attributes={"name": "Alice", "age": 14}, nesting_mode="set",), terraform::config::Block(name="pets", attributes={"type": "dog"}, nesting_mode="dict", key="Brutus",), terraform::config::Block(name="favorite_dishes", attributes={"name": "Pizza"}, nesting_mode="list", key="1",), terraform::config::Block(name="favorite_dishes", attributes={"name": "Pasta"}, nesting_mode="list", key="2",)], parent=null, state=get_resource_attribute(terraform_resource, []),) It will be serialized as follows (the order of the children list might differ): .. code-block:: { "name": "Albert", "children": [{ "name": "Alice", "age": 14, }, { "name": "Bob", "age": 12, },], "pets": { "Brutus": { "type": "dog", }, }, "favorite_dishes": [{"name": "Pizza"}, {"name": "Pasta"},], }

relation terraform::config::Block children [0:*]

other end: *terraform::config::Block.parent [0:1]*

relation terraform::config::Block parent [0:1]

other end: *terraform::config::Block.children [0:*]*

The following implementations are defined for this entity:

- *terraform::config::generate_key*
- *terraform::config::serialize*
- *terraform::config::build_state*
- *terraform::config::deprecation_warning*

The following implements statements select implementations for this entity:

- *terraform::config::generate_key* constraint `(not (self.nesting_mode in ['list', 'dict']))`
- *terraform::config::serialize* constraint `true`
- *terraform::config::build_state* constraint `self.name is defined`
- *terraform::config::deprecation_warning* constraint `self.deprecated`

Implementations

implementation terraform::agentConfig

implementation terraform::providerBlockConfig

If self.manual_config is false, the user should provide the root_config relation, the Block entity will be serialized and attached to this entity config attribute.

implementation terraform::providerManualConfig

If self.manual_config is true, the user should provide the config as a dict directly to the entity. The root_config relation should then be set to null.

implementation terraform::resourceBlockConfig

If self.manual_config is false, the user should provide the root_config relation, the Block entity will be serialized and attached to this entity config attribute.

implementation terraform::resourceManualConfig

If self.manual_config is true, the user should provide the config as a dict directly to the entity. The root_config relation should then be set to null.

implementation terraform::config::build_state

Extract the state matching this block from the parent state. This should only be called on non-root blocks. The root block should get the config from the resource parameter storing it.

implementation terraform::config::deprecation_warning

Send a warning that the usage of this block is deprecated

implementation terraform::config::generate_key

Automatically generate the key for blocks that don't require it to be set. This key will be a hash of the block's config and can then be used to order the block and generate a consistent config even with unordered sets of children blocks. (e.a. nesting_mode=set)

implementation terraform::config::serialize

Serialize this block into a config dict.

Plugins

terraform.**deprecated_config_block**(*config_block: terraform::config::Block*)

Log a warning for the usage of a deprecated config block

terraform.**dict_hash**(*input: dict*) → string

terraform.**extract_state**(*parent_state: dict*, *config: terraform::config::Block*) → dict

Extract the state corresponding to the provided config block from the parent state. This method should only be used with a state originating from the safe_resource_state plugin.

Parameters

- **state** – The parent state dict, it should include our config at key config.name
- **config** – The config block we want to find the matching config for.

terraform.**get_resource_attribute**(*resource: terraform::Resource*, *attribute_path: any*) → any

Get a resource attribute from the saved parameters (facts).

Disclaimer: Whatever comes out of this method might not be very safe to use, as it might be out of sync with the current state of the model. i.e. If you access here the id of a file, which is modified in the same

model, the id you will receive will be the one of the previous file not the one deployed in this model.

It is safer to use safe_resource_state plugin.

Parameters

- **resource** – The resource we which to get an attribute from.
- **attribute_path** – The path, in the resource state dict, to the desired value.

terraform.**get_resource_attribute_ref**(*resource: terraform::Resource*, *attribute_path: any*) → dict

Get a resource attribute reference. The difference with get_resource_attribute is that the value is not resolved at compile time but during the handler execution. This means that:

1. The value can not be manipulated in the model.
2. We save some time during the compile as we don't need to make api calls.
3. We avoid multiple recompile due to unknown values.
4. **If the targeted value changes, but none of the other attributes of this** resource, we will need a full deploy to have our value up to date.

Parameters

- **resource** – The resource we which to get an attribute from.
- **attribute_path** – The path, in the resource state dict, to the desired value.

terraform.**safe_resource_state**(*resource: terraform::Resource*) → dict

Get the state dict of a resource and check whether the current config of the resource has changed since the state was published. If this is the case, raise an Unknown value, as the state is out of sync and is dangerous to use.

terraform.**serialize_config**(*config_block: terraform::config::Block*) → dict

Serialize a config block into a dictionnary.

terraform.**sorted_list**(*input_list: list*) → list

Resources

class terraform.terraform_resource.**TerraformResource**

- Resource for entity *terraform::Resource*
- Id attribute `id`
- Agent name `provider.agent_config.agentname`
- Handlers *terraform.terraform_resource.TerraformResourceHandler*

Handlers

class terraform.terraform_resource.**TerraformResourceHandler**

- Handler name `terraform-resource`
- Handler for entity *terraform::Resource*

13.7.21 Module ubuntu

- License: Apache 2.0
- Version: 0.4.12
- 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*

13.7.22 Module user

- License: ASL 2
- Version: 0.1.14
- 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

13.7.23 Module vyos

- License: ASL2.0
- Version: 2.0.3
- Upstream project: https://github.com/inmanta/vyos.git

Typedefs

typedef vyos::abrtype_t

- Base type string
- Type constraint (self in ['cisco','ibm','shortcut','standard'])

typedef vyos::area

- Base type number
- Type constraint ((self >= 0) and (self < 4294967296))

typedef vyos::duplex

- Base type string
- Type constraint (((self == 'auto') or (self == 'half')) or (self == 'full'))

typedef vyos::ospf_metric_t

- Base type number
- Type constraint ((self > 0) and (self <= 16))

typedef vyos::ospf_metric_type_t

- Base type number
- Type constraint (self in [1,2])

typedef vyos::redistribute_t

- Base type string
- Type constraint (self in ['bgp','connected','kernel','rip','static'])

typedef vyos::speed

- Base type string
- Type constraint (self in ['10','100','1000','2500','10000','auto'])

typedef vyos::tunnel_encap_t

- Base type string
- Type constraint (self in ['gre','gre-bridge','ipip','sit','ipip6','ip6ip6'])

typedef vyos::tunnel_key_t

- Base type number
- Type constraint ((self >= 0) and (self <= 99999))

typedef vyos::tunnel_mtu_t

- Base type number
- Type constraint ((self >= 64) and (self <= 8024))

typedef vyos::vlan_id

- Base type int
- Type constraint ((self >= 0) and (self < 4095))

typedef vyos::firewall::action_t

- Base type string
- Type constraint (self in ['accept','drop','reject'])

typedef vyos::firewall::protocol_t

- Base type string
- Type constraint (self in ['tcp_udp','all','icmp','tcp','udp'])

typedef vyos::routemap::rm_action_t

- Base type string
- Type constraint (self in ['permit','deny'])

typedef vyos::vpn::auth_mode_t

- Base type string
- Type constraint (self in ['pre-shared-secret','rsa','x509'])

typedef vyos::vpn::conn_type_t

- Base type string
- Type constraint (self in ['initiate','respond'])

typedef vyos::vpn::dh_group_t

- Base type string
- Type constraint (self in [2,5,14,15,16,17,18,19,20,21,22,23,24,25,26])

typedef vyos::vpn::encryption_t

- Base type string
- Type constraint (self in ['aes128','aes256','3des'])

typedef vyos::vpn::esp_mode_t

- Base type string
- Type constraint (self in ['tunnel','transport'])

typedef vyos::vpn::hash_t

- Base type string
- Type constraint (self in ['md5','sha1','sha256','sha384','sha512'])

typedef vyos::vpn::kex_t

- Base type string
- Type constraint (self in ['ikev1','ikev2'])

typedef vyos::vpn::local_address_t

- Base type string
- Type constraint (ip::is_valid_ip_v10(self) or (self == 'any'))

Entities

entity vyos::Address

Parents: *std::Entity*

An address entity to add multiple addresses to an interface

attribute ip::cidr_v10 ip

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::BaseHost

Parents: *ip::Host*

A vyos (or derivative) based host.

attribute string user='inmanta'

attribute string password='inmanta'

attribute number port=22

attribute bool skip_on_connect_error=false

When true, vyos resources deployed on this host will be skipped when the handler fails to connect to the host. Otherwise the resource will be marked as failed.

relation vyos::Credential credential [1]

The following implementations are defined for this entity:

- *vyos::vyosConfig*
- *vyos::commonConfig*

The following implements statements select implementations for this entity:

- *vyos::vyosConfig* constraint true
- constraint true

entity vyos::BaseInterface

Parents: *vyos::ConfigNode*

attribute string name

attribute ip::cidr_v10? address=null

attribute bool dhcp=false

relation vyos::Address addresses [0:*]

relation vyos::PolicyRoute policy_route [0:1]

Set a policy route for this interface.

relation vyos::Shaper traffic_policy_out [0:1]

other end: *vyos::Shaper.interfaces_in [0:*]*

relation vyos::Shaper traffic_policy_in [0:1]

other end: *vyos::Shaper.interfaces_out [0:*]*

relation vyos::Bridge bridge_group [0:1]

other end: *vyos::Bridge.members [0:*]*

The following implementations are defined for this entity:

- *vyos::ifacePolicyRoute*

The following implements statements select implementations for this entity:

- *vyos::ifacePolicyRoute* constraint policy_route is defined

entity vyos::Bridge

Parents: *vyos::BaseInterface*

attribute string type='bridge'

relation vyos::BaseInterface `members` [0:*]

other end: *vyos::BaseInterface.bridge_group [0:1]*

The following implementations are defined for this entity:

- *vyos::bridge*

The following implements statements select implementations for this entity:

- *vyos::bridge* constraint `true`

entity `vyos::Config`

Parents: *vyos::ConfigItem*, *std::PurgeableResource*

VYOS config block resource

This is the central resource, that is used to deploy specific configlets.

attribute string `device`

attribute string `node`

attribute bool `never_delete`=false

attribute bool `save`=true

attribute bool `send_event`=true

attribute string[] `keys_only`=List()

Only compare these keys, ignore all other keys that are in the current state

attribute string[] `ignore_keys`=List()

Ignore these keys in the current state

attribute bool `facts`=false

When set to true the config is never executed. The value under node is exposed as a fact

attribute bool `skip_on_connect_error`

relation vyos::Credential `credential` [1]

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `vyos::ConfigItem`

Parents: *std::Entity*

attribute string `config`

relation vyos::ExtraConfig `extra` [0:*]

other end: *vyos::ExtraConfig.parent [1]*

entity `vyos::ConfigNode`

Parents: *std::Entity*

attribute string `node_name`

attribute bool `purged`=false

attribute bool `purge_on_delete`=false

relation vyos::ConfigItem config [0:1]

relation vyos::BaseHost host [1]

The following implementations are defined for this entity:

- *vyos::vpn::ipsecOptions*

entity vyos::Credential

Parents: *std::Entity*

attribute string address

attribute string user

attribute string password

attribute number port

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::DhcpServer

Parents: *vyos::ConfigNode*

attribute string name

attribute ip::cidr subnet

attribute ip::ip default_router

attribute ip::ip[] dns_servers

attribute ip::ip range_start

attribute ip::ip range_end

The following implementations are defined for this entity:

- *vyos::dhcpServer*

The following implements statements select implementations for this entity:

- *vyos::dhcpServer* constraint true

entity vyos::ExtraConfig

Parents: *vyos::ConfigItem*

relation vyos::ConfigItem parent [1]

other end: *vyos::ConfigItem.extra [0:*]*

The following implementations are defined for this entity:

- *vyos::extraconfig_depends*

The following implements statements select implementations for this entity:

- *vyos::extraconfig_depends* constraint true

entity vyos::Host

Parents: *vyos::BaseHost*

The following implements statements select implementations for this entity:

- constraint `true`
- *vyos::commonConfig* constraint `true`

entity vyos::Hostname

Parents: *vyos::ConfigNode*

attribute string `name`

The following implementations are defined for this entity:

- *vyos::hostname*

The following implements statements select implementations for this entity:

- *vyos::hostname* constraint `true`

entity vyos::Interface

Parents: *vyos::BaseInterface*

attribute bool `never_delete`=false

attribute vyos::duplex `duplex`='auto'

attribute vyos::speed `speed`='auto'

relation vyos::firewall::RuleSet `inbound_ruleset` [0:1]

relation vyos::firewall::RuleSet `local_ruleset` [0:1]

relation vyos::firewall::RuleSet `outbound_ruleset` [0:1]

The following implementations are defined for this entity:

- *vyos::iface*

The following implements statements select implementations for this entity:

- *vyos::iface* constraint `true`

entity vyos::IpFact

Parents: *std::PurgeableResource*

Discover interface IP

attribute string `id`

attribute string `device`

relation vyos::BaseHost `host` [1]

relation vyos::Credential `credential` [1]

relation vyos::Interface `interface` [1]

The following implementations are defined for this entity:

- *vyos::wireup_ipfact*

The following implements statements select implementations for this entity:

- *vyos::wireup_ipfact* constraint `true`

entity vyos::Loopback

Parents: *vyos::ConfigNode*

attribute ip::cidr `address`

The following implementations are defined for this entity:

- *vyos::loopback*

The following implements statements select implementations for this entity:

- *vyos::loopback* constraint `true`

entity vyos::Masquerade

Parents: *vyos::ConfigNode*

attribute string `outbound_interface`

attribute string `source_address`

attribute number `rule`

The following implementations are defined for this entity:

- *vyos::masq*

The following implements statements select implementations for this entity:

- *vyos::masq* constraint `true`

entity vyos::Ospf

Parents: *vyos::ConfigNode*

attribute vyos::area `area=0`

attribute ip::cidr[] `network`

attribute ip::ip `router_id`

attribute string[]? `passive_interfaces`

attribute string[]? `passive_interface_excludes`

attribute vyos::abrtype_t `abrtype='cisco'`

relation vyos::OspfRedistribute `redistributes` [0:*]

other end: *vyos::OspfRedistribute.ospf [1]*

The following implementations are defined for this entity:

- *vyos::ospf*

The following implements statements select implementations for this entity:

- *vyos::ospf* constraint `true`

entity vyos::OspfRedistribute

Parents: *std::Entity*

attribute vyos::redistribute_t `type`

attribute vyos::ospf_metric_t? `metric`

attribute vyos::ospf_metric_type_t `metric_type`=2

attribute string? `route_map`=null

relation vyos::Ospf `ospf` [1]

other end: *vyos::Ospf.redistributes [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `vyos::PolicyRoute`

Parents: *vyos::ConfigNode*

Route policy for Vyos Polciy Based Routing.

attribute std::alfanum `name`

The name for this policy route

relation vyos::PolicyRouteRule `rules` [1:*]

other end: *vyos::PolicyRouteRule.policy [1]*

The following implementations are defined for this entity:

- *vyos::policyRoute*

The following implements statements select implementations for this entity:

- *vyos::policyRoute* constraint `true`

entity `vyos::PolicyRouteRule`

Parents: *vyos::ConfigNode*

Rule in a route policy for Vyos Polciy Based Routing.

attribute number `id`

The rule number

attribute number `table`

Routing table for traffic matching this rule

attribute std::alfanum? `description`=null

Description for this rule

attribute ip::cidr? `match_source_address`=null

The source address to match traffic on

attribute ip::cidr? `match_destination_address`=null

The destination address to match traffic on. Can only be specified if match_protocol is set

attribute ip::port? `match_source_port`=null

The source port to match traffic on. Can only be specified if match_protocol in [“tcp”, “udp”]

attribute ip::port? `match_destination_port`=null

The destination port to match traffic on

attribute std::alfanum? `match_protocol`=null

The protocol to match traffic on

relation vyos::PolicyRoute `policy` [1]

other end: *vyos::PolicyRoute.rules [1:*]*

The following implementations are defined for this entity:

- *vyos::policyRouteRule*

The following implements statements select implementations for this entity:

- *vyos::policyRouteRule* constraint `true`

entity `vyos::RouteMap`

Parents: *vyos::ConfigNode*

attribute string `name`

attribute string? `description`=null

relation vyos::routemap::Rule `rules` [0:*]

The following implementations are defined for this entity:

- *vyos::routeMap*

The following implements statements select implementations for this entity:

- *vyos::routeMap* constraint `true`

entity `vyos::Shaper`

Parents: *vyos::ConfigNode*

attribute string `name`

attribute string `bandwidth`

attribute string `default_bandwidth`='50%'

attribute string `default_ceiling`='100%'

attribute string `default_queue_type`='fair-queue'

relation vyos::BaseInterface `interfaces_in` [0:*]

other end: *vyos::BaseInterface.traffic_policy_out [0:1]*

relation vyos::BaseInterface `interfaces_out` [0:*]

other end: *vyos::BaseInterface.traffic_policy_in [0:1]*

The following implementations are defined for this entity:

- *vyos::shaper*

The following implements statements select implementations for this entity:

- *vyos::shaper* constraint `true`

entity `vyos::StaticRoute`

Parents: *vyos::ConfigNode*

attribute ip::cidr `destination`

attribute ip::ip `next_hop`

attribute number table=0

The following implementations are defined for this entity:

- *vyos::staticRouteDefault*
- *vyos::staticRouteTable*

The following implements statements select implementations for this entity:

- *vyos::staticRouteDefault* constraint (table == 0)
- *vyos::staticRouteTable* constraint (table > 0)

entity vyos::Tunnel

Parents: *vyos::BaseInterface*

attribute string? description=null

attribute vyos::tunnel_mtu_t mtu=1476

attribute vyos::tunnel_encap_t encapsulation

attribute ip::ip_v10 local_ip

attribute ip::ip_v10? remote_ip=null

attribute vyos::tunnel_key_t? key=null

The following implementations are defined for this entity:

- *vyos::tunnel*

The following implements statements select implementations for this entity:

- *vyos::tunnel* constraint true

entity vyos::Vif

Parents: *vyos::BaseInterface*

attribute vyos::vlan_id vlan

attribute string type='vif'

attribute string name=''

relation vyos::Interface parent [1]

The following implementations are defined for this entity:

- *vyos::vif*

The following implements statements select implementations for this entity:

- *vyos::vif* constraint true

entity vyos::firewall::AddressGroup

Parents: *vyos::firewall::Group*

attribute string[] addresses

string vyos::firewall::AddressGroup.description='inmanta managed address-group'

The following implementations are defined for this entity:

- *vyos::firewall::addressGroup*

The following implements statements select implementations for this entity:

- *vyos::firewall::addressGroup* constraint `true`

entity vyos::firewall::Group

Parents: *vyos::ConfigNode*

attribute string `name`

attribute string `group_type`

entity vyos::firewall::NetworkGroup

Parents: *vyos::firewall::Group*

attribute ip::cidr[] `networks`

string vyos::firewall::NetworkGroup.description='inmanta managed network-group'

The following implementations are defined for this entity:

- *vyos::firewall::networkGroup*

The following implements statements select implementations for this entity:

- *vyos::firewall::networkGroup* constraint `true`

entity vyos::firewall::PortGroup

Parents: *vyos::firewall::Group*

attribute string[] `ports`

string vyos::firewall::PortGroup.description='inmanta managed port-group'

The following implementations are defined for this entity:

- *vyos::firewall::portGroup*

The following implements statements select implementations for this entity:

- *vyos::firewall::portGroup* constraint `true`

entity vyos::firewall::Rule

Parents: *std::Entity*

attribute number `id`

attribute vyos::firewall::action_t `action`

attribute vyos::firewall::protocol_t `protocol`

string vyos::firewall::Rule.description='inmanta managed rule'

relation vyos::firewall::Group `source` [0:*]

relation vyos::firewall::Group `destination` [0:*]

relation vyos::firewall::RuleSet ruleset [1]
other end: *vyos::firewall::RuleSet.rules [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::firewall::RuleSet

Parents: *vyos::ConfigNode*

attribute string name

attribute vyos::firewall::action_t default_action

string vyos::firewall::RuleSet.description='inmanta managed ruleset'

relation vyos::firewall::Rule rules [0:*]
other end: *vyos::firewall::Rule.ruleset [1]*

The following implementations are defined for this entity:

- *vyos::firewall::ruleSet*

The following implements statements select implementations for this entity:

- *vyos::firewall::ruleSet* constraint true

entity vyos::openstackext::OpenstackHost

Parents: *vyos::BaseHost*, *openstack::Host*

A vyos based host for Openstack

attribute string? floatingIP

The following implementations are defined for this entity:

- *vyos::openstackext::openstackConfig*
- *vyos::openstackext::withFip*

The following implements statements select implementations for this entity:

- *vyos::openstackext::withFip* constraint floatingIP is defined
- *vyos::commonConfig* constraint (not floatingIP is defined)
- constraint true
- *vyos::openstackext::openstackConfig* constraint true

entity vyos::routemap::Match

Parents: *std::Entity*

attribute string? interface=null

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::routemap::Rule

Parents: *std::Entity*

attribute number id

attribute vyos::routemap::rm_action_t action

relation vyos::routemap::Match match [1]

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::vpn::Authentication

Parents: *std::Entity*

attribute string id

attribute vyos::vpn::auth_mode_t mode

attribute string? pre_shared_key=null

attribute string? remote_id=null

attribute string? rsa_key_name=null

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::vpn::ESPGroup

Parents: *vyos::ConfigNode*

attribute string name

attribute bool compression

attribute number lifetime

attribute vyos::vpn::esp_mode_t mode

attribute bool pfs

relation vyos::vpn::ESPProposal proposals [1:*]

The following implementations are defined for this entity:

- *vyos::vpn::espGroup*

The following implements statements select implementations for this entity:

- *vyos::vpn::espGroup* constraint true

entity vyos::vpn::ESPProposal

Parents: *std::Entity*

attribute number id

attribute vyos::vpn::encryption_t encryption

attribute vyos::vpn::hash_t hash='sha1'

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::vpn::IKEGroup

Parents: *vyos::ConfigNode*

attribute string name

attribute vyos::vpn::kex_t key_exchange='ikev1'

attribute number lifetime

relation vyos::vpn::IKEProposal proposals [1:*]

The following implementations are defined for this entity:

- *vyos::vpn::ikeGroup*

The following implements statements select implementations for this entity:

- *vyos::vpn::ikeGroup* constraint true

entity vyos::vpn::IKEProposal

Parents: *std::Entity*

attribute number id

attribute vyos::vpn::dh_group_t? dh_group=null

attribute vyos::vpn::encryption_t encryption

attribute vyos::vpn::hash_t hash='sha1'

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity vyos::vpn::IPSECOptions

Parents: *vyos::ConfigNode*

attribute string[] ipsec_interfaces=List()

attribute string[] log_modes=List()

attribute bool nat_traversal=false

attribute ip::cidr[] allowed_nat_networks=List()

The following implements statements select implementations for this entity:

- *vyos::vpn::ipsecOptions* constraint true

entity vyos::vpn::KeyGen

Parents: *std::PurgeableResource*

Ensure an RSA key has been generated

attribute string id='keygen'

attribute string device

relation vyos::BaseHost host [1]

relation vyos::Credential credential [1]

The following implementations are defined for this entity:

- *vyos::vpn::wireup*

The following implements statements select implementations for this entity:

- *vyos::vpn::wireup* constraint true

entity vyos::vpn::RSAKey

Parents: *vyos::ConfigNode*

attribute string name

attribute string rsa_key

The following implementations are defined for this entity:

- *vyos::vpn::rsaKey*

The following implements statements select implementations for this entity:

- *vyos::vpn::rsaKey* constraint true

entity vyos::vpn::SiteToSite

Parents: *vyos::ConfigNode*

attribute string peer

attribute vyos::vpn::conn_type_t connection_type

attribute vyos::vpn::local_address_t local_address

relation vyos::vpn::Authentication authentication [1]

relation vyos::vpn::IKEGroup ike_group [1]

relation vyos::vpn::ESPGroup default_esp_group [0:1]

relation vyos::vpn::Tunnel tunnels [0:*]

The following implementations are defined for this entity:

- *vyos::vpn::siteToSite*

The following implements statements select implementations for this entity:

- *vyos::vpn::siteToSite* constraint true

entity vyos::vpn::Tunnel

Parents: *std::Entity*

attribute number id

attribute ip::cidr_v10 local_prefix

attribute ip::cidr_v10 remote_prefix

The following implements statements select implementations for this entity:

- *std::none* constraint true

Implementations

implementation vyos::bridge

implementation vyos::commonConfig

implementation vyos::dhcpServer

implementation vyos::extraconfig_depends

implementation vyos::hostname

implementation vyos::iface

implementation vyos::ifacePolicyRoute

implementation vyos::loopback

implementation vyos::masq

implementation vyos::ospf

implementation vyos::policyRoute

implementation vyos::policyRouteRule

implementation vyos::routeMap

implementation vyos::shaper

implementation vyos::staticRouteDefault

implementation vyos::staticRouteTable

implementation vyos::tunnel

implementation vyos::vif

implementation vyos::vyosConfig

implementation vyos::wireup_ipfact

implementation vyos::firewall::addressGroup

implementation vyos::firewall::networkGroup

implementation vyos::firewall::portGroup

implementation vyos::firewall::ruleSet

implementation vyos::openstackext::openstackConfig

implementation vyos::openstackext::withFip

implementation vyos::vpn::espGroup

implementation vyos::vpn::ikeGroup

implementation vyos::vpn::ipsecOptions

implementation vyos::vpn::rsaKey

implementation vyos::vpn::siteToSite

implementation vyos::vpn::wireup

Resources

class vyos.**Config**

- Resource for entity *vyos::Config*
- Id attribute nodeid
- Agent name device
- Handlers *vyos.VyosHandler*

class vyos.**IpFact**

- Resource for entity *vyos::IpFact*
- Id attribute id
- Agent name device
- Handlers *vyos.IpFactHandler*

class vyos.**KeyGen**

- Resource for entity *vyos::vpn::KeyGen*
- Id attribute id
- Agent name device
- Handlers *vyos.KeyGenHandler*

Handlers

class vyos.**VyosHandler**

- Handler name sshconfig
- Handler for entity *vyos::Config*

class vyos.**KeyGenHandler**

- Handler name keygen
- Handler for entity *vyos::vpn::KeyGen*

class vyos.**IpFactHandler**

- Handler name IpFact
- Handler for entity *vyos::IpFact*

13.7.24 Module web

- License: Apache 2.0
- Version: 0.3.12
- Upstream project: https://github.com/inmanta/web.git

Entities

entity web::Alias

Parents: *std::Entity*

An alias (hostname) for a web application

attribute std::hoststring hostname

relation web::Application application [0:*]

other end: *web::Application.name [1]*

relation web::Application application_alias [0:*]

other end: *web::Application.aliases [0:*]*

relation web::Cluster cluster [0:1]

other end: *web::Cluster.name [1]*

relation web::Cluster cluster_alias [0:1]

other end: *web::Cluster.aliases [0:*]*

relation web::LoadBalancedApplication loadbalancer [0:1]

other end: *web::LoadBalancedApplication.name [1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity web::Application

Parents: *std::Entity*

This entity models a webapplication

attribute string document_root

relation web::Alias name [1]

other end: *web::Alias.application [0:*]*

relation web::Alias aliases [0:*]

other end: *web::Alias.application_alias [0:*]*

relation web::ApplicationContainer container [1]

other end: *web::ApplicationContainer.application [0:*]*

relation web::LoadBalancedApplication lb_app [0:1]

other end: *web::LoadBalancedApplication.app_instances [1:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity web::ApplicationContainer

Parents: *ip::services::Server*

A container that hosts webapplications

attribute string user

The group name of the group as which the process of this container runs

attribute string group

attribute number port=80

relation web::Application application [0:*]

other end: *web::Application.container [1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity web::Cluster

Parents: *std::Entity*

A webapplication that is hosted as a cluster

attribute number cluster_size

relation web::Alias name [1]

other end: *web::Alias.cluster [0:1]*

relation web::Alias aliases [0:*]

other end: *web::Alias.cluster_alias [0:1]*

relation web::LoadBalancedApplication loadbalancer [1:*]

other end: *web::LoadBalancedApplication.web_cluster [0:*]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity web::HostedLoadBalancer

Parents: *web::LoadBalancer*, *ip::services::Server*

entity web::LoadBalancedApplication

Parents: *std::Entity*

attribute bool nameonly=true

relation web::Cluster web_cluster [0:*]

other end: *web::Cluster.loadbalancer [1:*]*

relation web::LoadBalancer loadbalancer [1:*]

other end: *web::LoadBalancer.applications [0:*]*

relation web::Application app_instances [1:*]

other end: *web::Application.lb_app [0:1]*

relation web::Alias name [1]

other end: *web::Alias.loadbalancer [0:1]*

The following implements statements select implementations for this entity:

- *std::none* constraint true

entity web::LoadBalancer

Parents: *ip::services::BaseServer*

A loadbalancer for web applications

relation web::LoadBalancedApplication applications [0:*]

other end: *web::LoadBalancedApplication.loadbalancer [1:*]*

13.7.25 Module yaml

A module to help with handling yaml files

- License: ASL 2.0
- Version: 0.1.0
- This module requires compiler version 2019.3 or higher
- Upstream project: https://github.com/inmanta/yaml.git

Plugins

yaml.**load**(*path: string*) → dict

Parse the yaml found in the file at 'path' into a dictionary

The path is according to the convention for std::source and std::file

yaml.**loads**(*content: string*) → dict

Parse the yaml found in *content* into a dictionary

13.7.26 Module yum

- License: Apache 2.0
- Version: 0.6.11
- Upstream project: https://github.com/inmanta/yum.git

Entities

entity yum::Repository

Parents: *std::Entity*

A yum repositoy

attribute string name

attribute bool gpgcheck=false

attribute bool enabled=true

attribute string baseurl

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::redhatRepo*

The following implements statements select implementations for this entity:

- *yum::redhatRepo* constraint std::familyof(host.os,'redhat')

Implementations

implementation yum::redhatRepo

CHAPTER

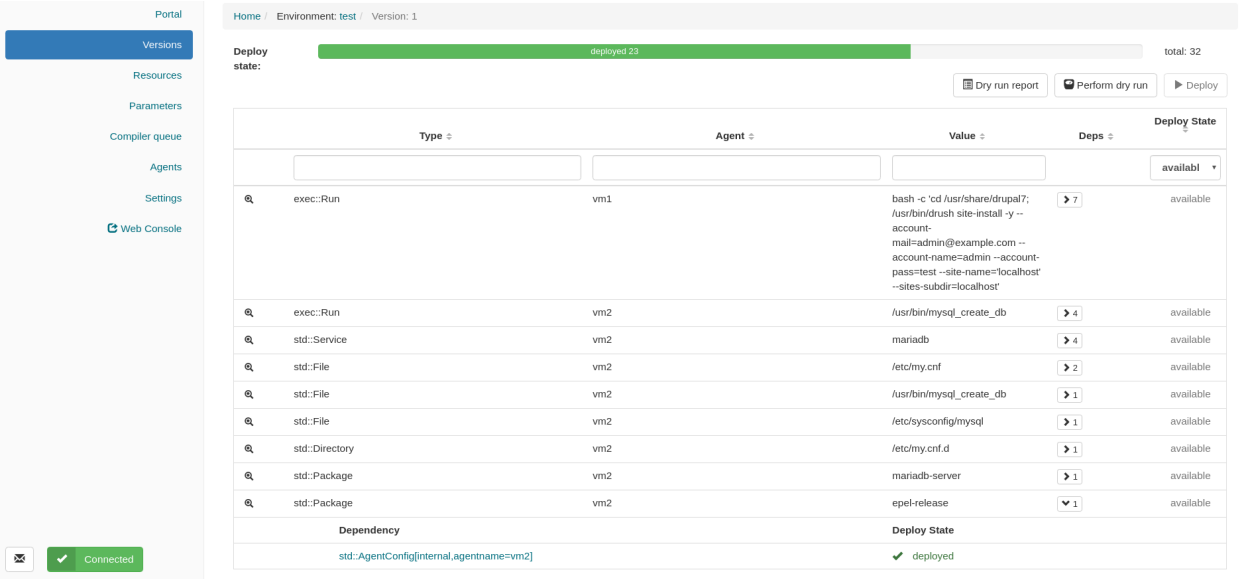
FOURTEEN

TROUBLESHOOTING

This page describes typical failure scenario's and provides a guideline on how to troubleshoot them.

14.1 A resources 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 version of the configuration model, shown in the versions tab of the Inmanta dashboard, 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 epel-release package should be ready to deploy on agent vm2

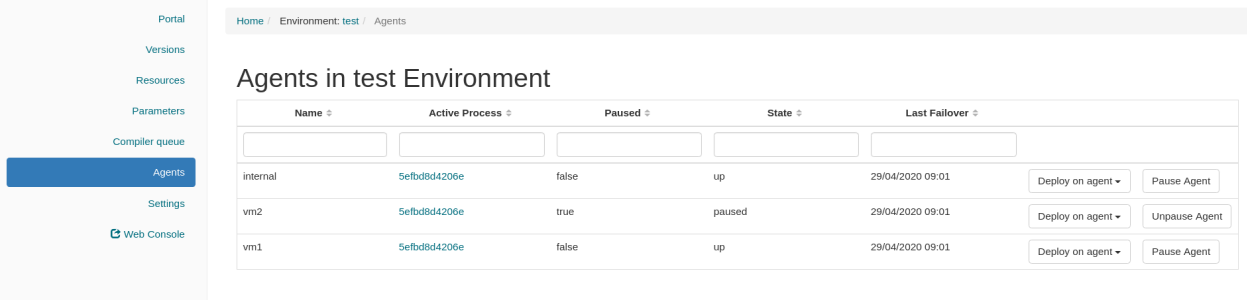


Next, go to the agents tab of the dashboard to verify the state of that agent.

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.

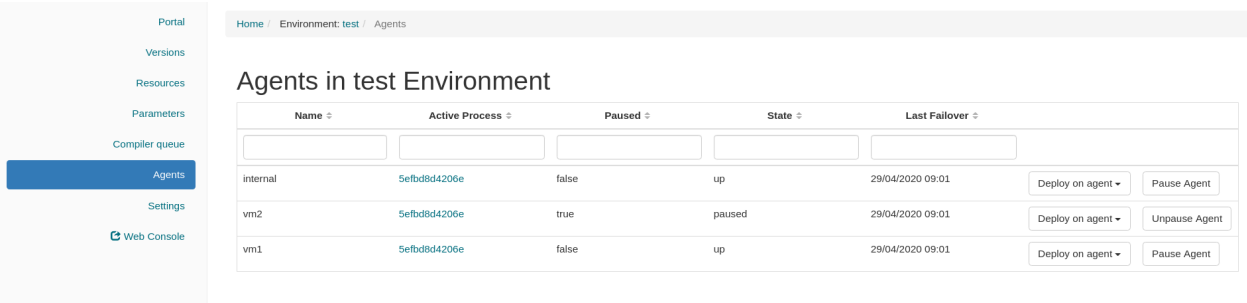


14.1.1 The agent is down

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.

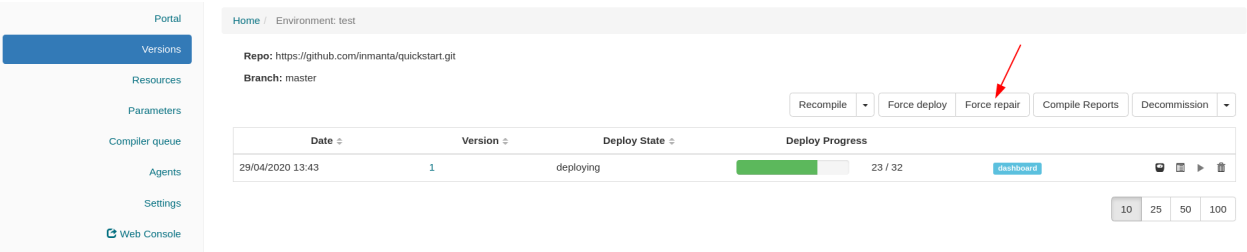
14.1.2 The agent is paused

Unpause the agent by clicking the `Unpause agent` button in the agents tab of the dashboard.



14.1.3 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 dashboard. 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 agents to execute a deployment by clicking on the `Force Repair` button in the versions tab of the dashboard, as shown in the figure below:



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`.

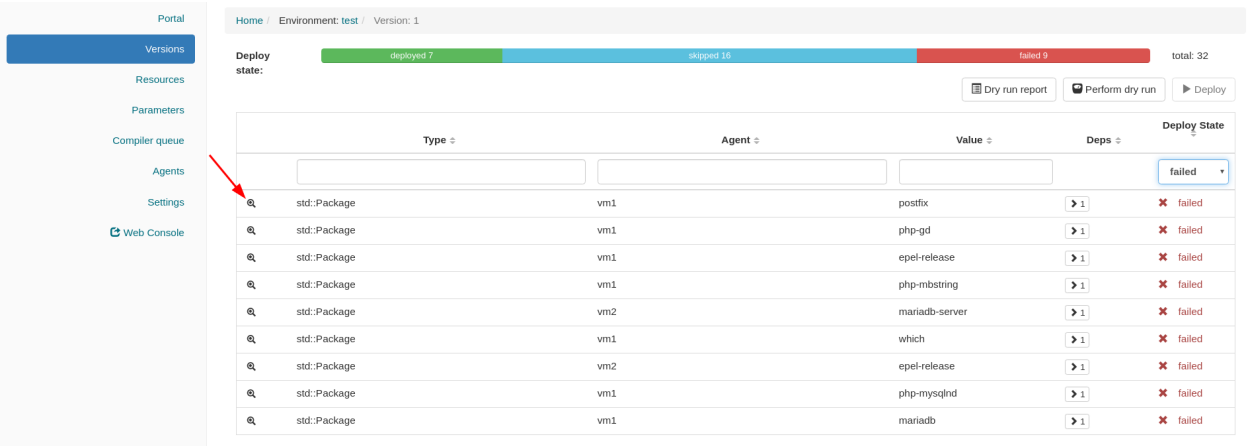
14.2 The deployment of a resource fails

When a resource cannot be deployed, it ends up in one of the following deployment states:

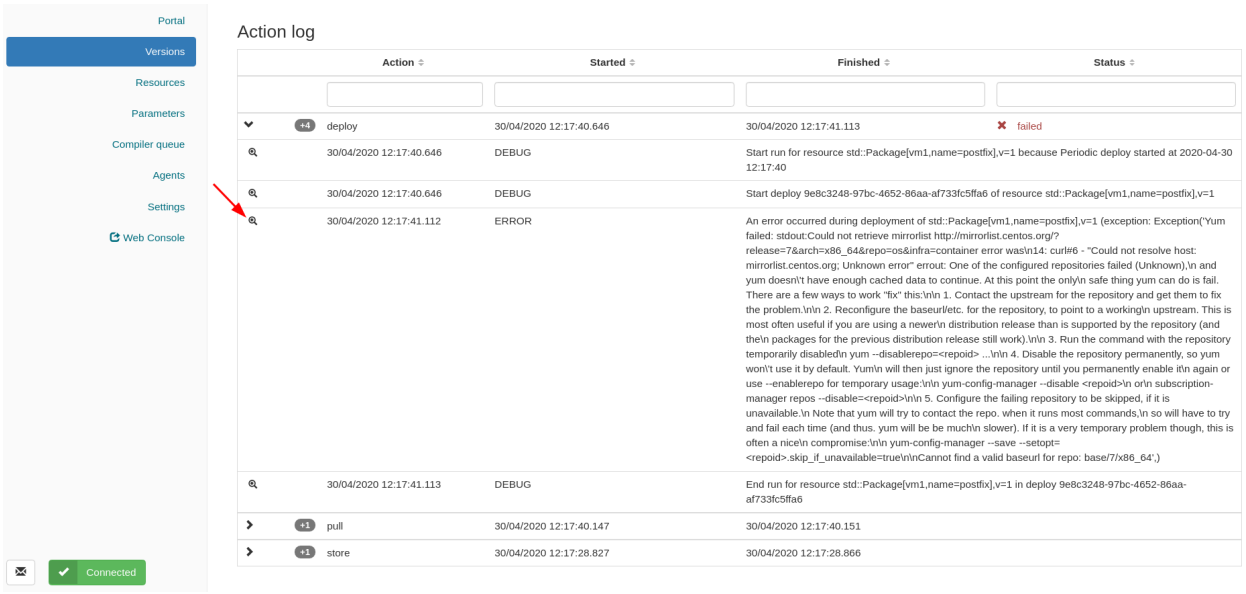
- **failed:** A resource ends up in the `failed` state when the handler of that resource raises an uncaught exception. *Check the log of the resource* to get more details about the issue.
- **unavailable:** A resource ends up in the `unavailable` state when no handler could be found to deploy that resource. *Check the log of the resource* to get more details about the issue.
- **undefined:** A resource ends up in the `undefined` state when a fact, required by that resource didn't yet resolve to a value. Read Section *Check which facts are not yet resolved* to find out which fact is still unknown.
- **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 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.

14.2.1 Read the logs of a resource

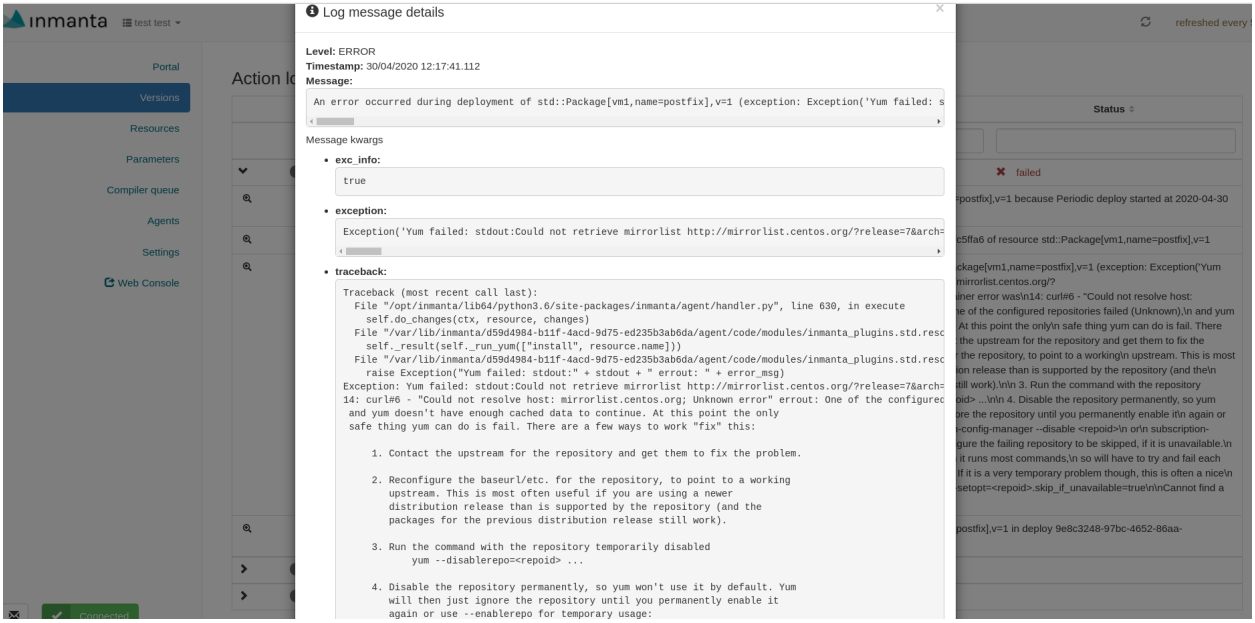
This section describes how to obtain the logs for a specific resource. In the versions tab of the dashboard, click on the version of the configuration model being deployed to get a list of all the resource in that configuration model. Next, click on the magnifier in front of a resource, as shown in the figure below, to get the logs for that specific resource. The log messages for the different stages of the deployment are grouped together.



The magnifier in front of each log message can be used to get a more structured output for that specific log message.



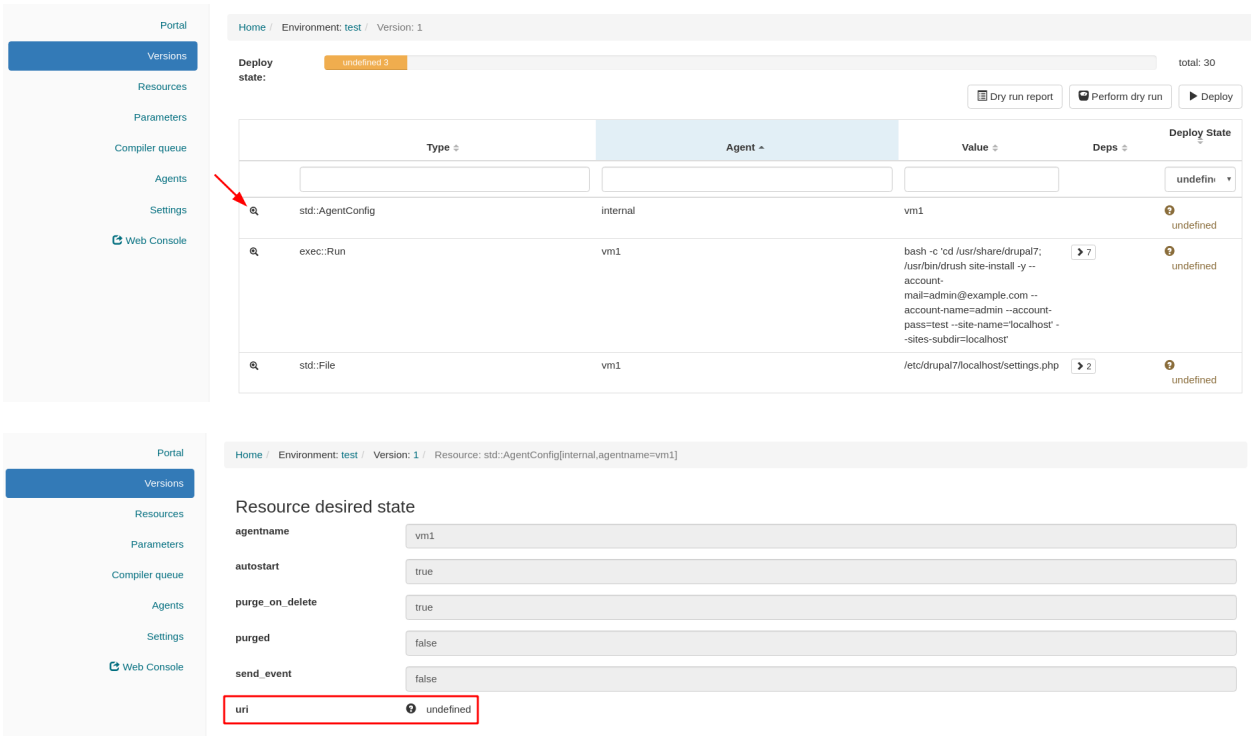
In the figure below, the traceback of the exception is shown.



14.2.2 Check which facts are not yet resolved

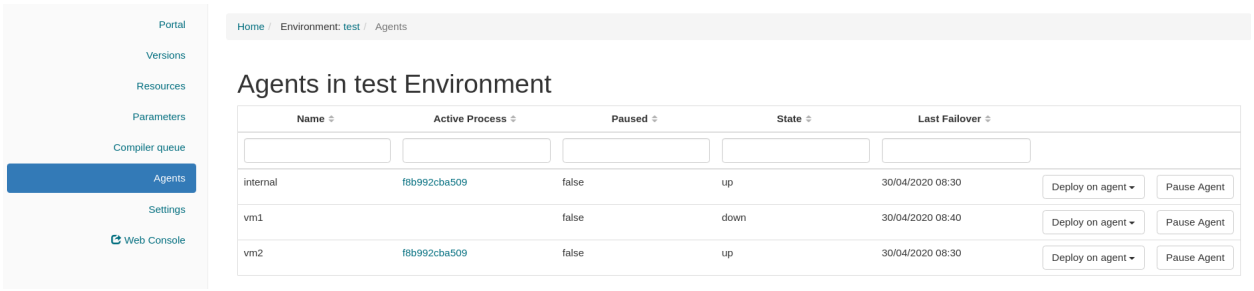
To find out which fact of a certain resource is not yet resolved, click on the magnifier in front of the resource in the `undefined` state, as shown in the figure below.

The list of attributes of that resource, will contain one attribute which is marked as undefined (See figure below). This is the attribute that wasn't resolved yet. Track the source of this attribute down within the configuration model to find out why this attribute is undefined.



14.3 Agent doesn't come up

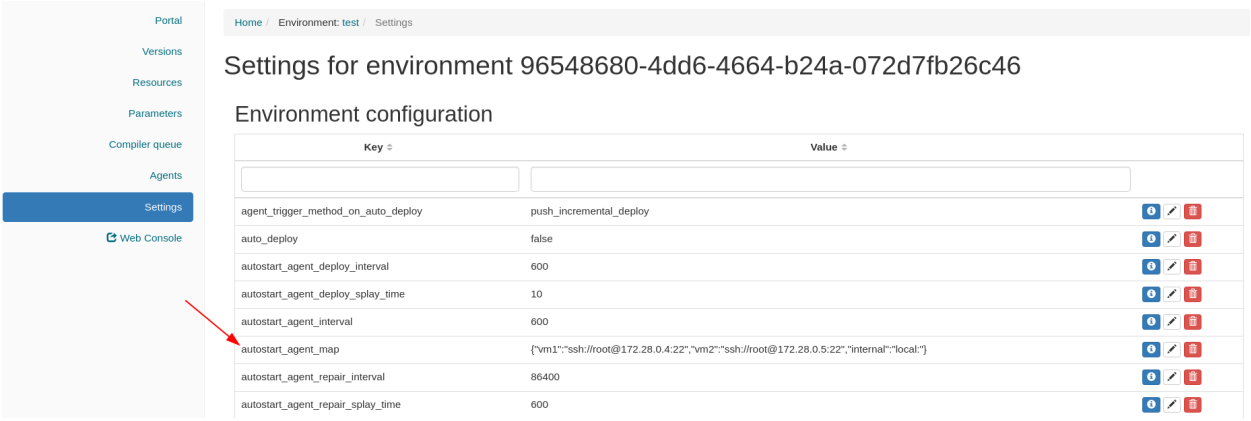
This section explains how to troubleshoot the problem where an agent is in the down state while it should be up. In the figure shown below, the agent vm1 is down.



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.

14.3.1 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 via the settings tab of the inmanta dashboard as shown in the figure below.



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 dashboard. 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.

14.3.2 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
```

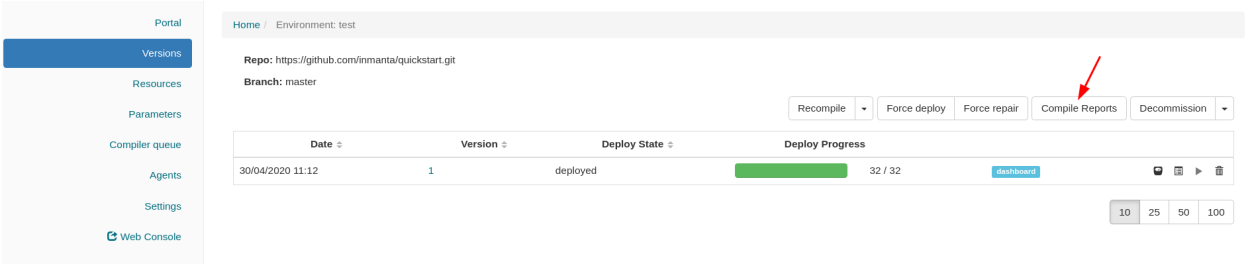
14.3.3 Potential reasons why an agent doesn't start

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.

14.4 No version appears after recompile trigger

After clicking the `Recompile` button of the dashboard, a new version of the configuration model should appear in the list of versions. If this doesn't happen, the compilation has failed. Click on the `Compile Reports` button, as shown in the figure below, to get the compile report of the latest compilation. This report will give more information about the exact problem.

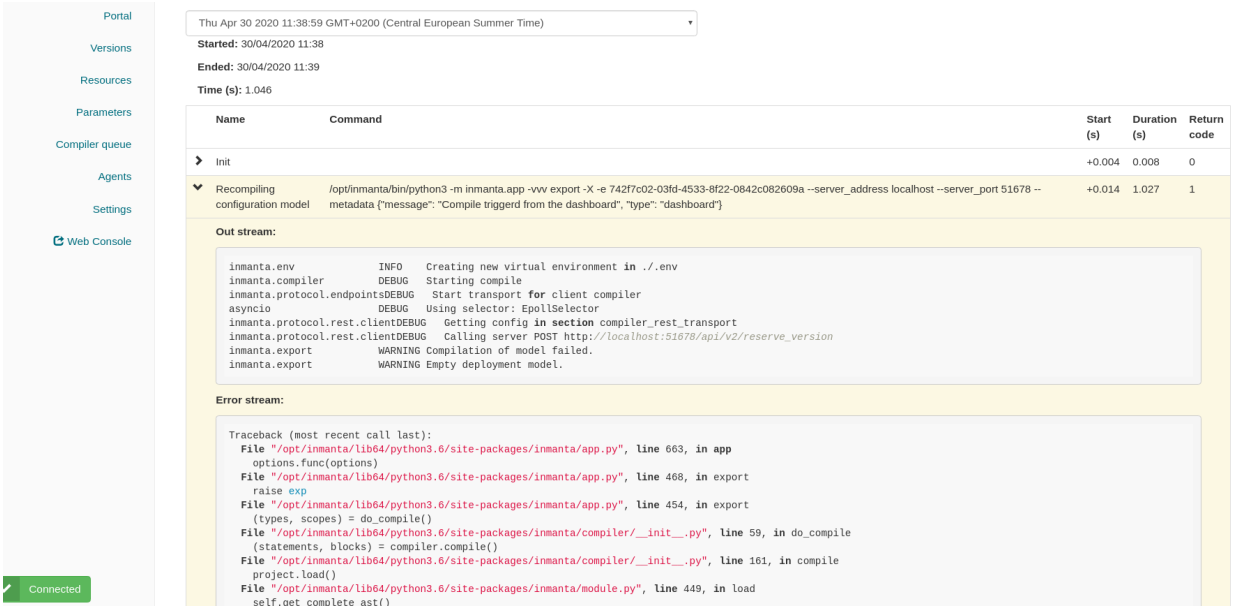


Each step of the compile process is shown, together with the output produced by that step and the return code. Verify that the timestamp of the compile report corresponds to the time the compilation was triggered in the dashboard. If no compile report was generated or the compile report doesn't show any errors, check the server logs as well. By default the server log is present in `<config.log-dir>/server.log`.

14.5 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
```



14.6 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.

14.6.1 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.

14.6.2 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>
```

14.6.3 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
```

(continues on next page)

(continued from previous page)

```
(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.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.

14.7 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

FIFTEEN

CHANGELOG

15.1 Release 2022.3 (2022-09-29)

15.1.1 General changes

Upgrade notes

- Ensure the database is backed up before executing an upgrade.

Bug fixes

- Add a signal handler to the entrypoint of the Inmanta container to correctly handle the termination of the container

15.1.2 Inmanta-core: release 7.1.0 (2022-09-29)

New features

- Add option to bytecompile all python source in a v2 module wheel (Issue inmanta/irt#1190)
- Replace Drupal model of quickstart with SR Linux. (Issue #4333)
- Added partial compile feature

Improvements

- When the AutostartedAgentManager starts a new agent process, it now uses a dynamic timeout on the time to wait until all agents are active. The AutostartedAgentManager raises a timeout as soon as no new agent has become active in the past five seconds. (Issue inmanta/inmanta-core#4398)
- Improved logging on the agent manager when restarting agents
- Performance improvements for the resource_did_dependency_change endpoint (Issue #4402)
- The `put_partial` endpoint and `inmanta export --partial` now dynamically allocate a new version.
- Add support for extras on Python dependencies (Issue inmanta/inmanta-core#4497)
- Improve logging on module installation. (Issue #4500)
- Reject v1tov2 module conversion when a setup.py is present
- Fix issue where the v1tov2 command removes the requirements.txt file (Issue #4684)

- Fix a bug in the typing of the new influxdb metrics (Issue #4688)
- Don't set PYTHONPATH environment variable on venv activation: fixes editable install compatibility with setuptools<64 (Issue #4713)
- Add argument to compilerservice to allow exporting with the specified exporter plugin
- Added options to compiler service to configure notification behavior (Issue #4803)
- Reduce compiler log level for iterations and cache log lines to debug
- For v1tov2 conversion, split tag from version and put it in tag_build field
- Improved editable v2 module compatibility with latest setuptools and PEP660 in edge case scenarios.
- Set the startup/shutdown order between the Inmanta server and the database in the docker-compose file

Upgrade notes

- It's required to update-and-recompile on each Inmanta project on the server after an upgrade (Issue inmanta/inmanta-core#4718)

Deprecation notes

- The internal upload_code endpoint has been removed, deprecated since core release 2018.2 (Issue inmanta/irt#1190)
- The `put_partial` endpoint (previously marked experimental) no longer accepts a version argument.

Bug fixes

- Fix rare deadlock in the database locking mechanism when tasks are cancelled, mostly affects test environments. (Issue #4384)
- Fix issue that causes an agent restart storm for all agents on an agent process when an agent on that process is paused. (Issue inmanta/inmanta-core#4398)
- make sure that the index present in PIP_INDEX_URL or PIP_EXTRA_INDEX_URL is not leaked to pip when using install_from_index (Issue inmanta/inmanta-core#4723)
- Fix issue where the pip consistency check is too strict (Issue #4761)
- The compiler service now logs the requested time of a recompile using a consistent timezone
- Fixed minor backwards incompatibility of the resource action database schema and `resource_action_update` endpoint
- Fix bugs in the merge logic of a partial compile. 1) Ensure that the version numbers present in the new version of the configuration model are set correctly. 2) Ensure that the resource states and unknowns, that belongs to the partial model, are sent to the server and merged correctly with the old configuration model.

15.1.3 Inmanta-ui: release 3.0.2 (2022-09-29)

No changelog entries.

15.1.4 inmanta-dashboard: release 3.8.1

This component has had no new releases since the last product version.

15.1.5 Web-console: release 1.11.2 (2022-09-29)

Improvements

- add delete button for desired state version with test coverage, bump test coverage for sibiling components (Issue #3957)
- replace KeyCloakInstance as it is depraceted (Issue #4002)

Upgrade notes

- Improve test coverage for conditionals (Issue #4000)

Bug fixes

- Scroll into view when new lines are being added to the report while it is compiling. (Issue #3855)
- Fix the overflow disapearing outside the window for the facts table. (Issue #3909)
- Add error handling for uncaught errors. (Issue #3924)
- replace instance uuid with instance identity when possible in action modals(Delete and set State Action)
- bump dependencies versions to resolve vulnerabilities (Issue #4001)
- Fixed issue where web-console would crash when failing to format xml

Other notes

- The editorconfig file now ensures that the codebase stays LF, the package.json will contain Windows specific commands for linting/prettier. Single quotes for paths are not supported by Windows. (Issue #3909)
- updated the jenkins scripts for tests to be slightly more performant (Issue #3924)

15.2 Release 2022.2.1 (2022-08-16)

15.2.1 Upgrade notes

- Ensure the database is backed up before executing an upgrade.

15.2.2 inmanta-core: release 7.0.0

This component has had no new releases since the last product version.

15.2.3 inmanta-ui: release 3.0.1

This component has had no new releases since the last product version.

15.2.4 inmanta-dashboard: release 3.8.1

This component has had no new releases since the last product version.

15.2.5 Web-console: release 1.11.1 (2022-08-16)

Bug fixes

- Fixed error on settings page resulting in blank page
- Fixed behavior of instance creation and update for services with inter-service relations

15.3 Release 2022.2 (2022-08-08)

15.3.1 Upgrade notes

- Ensure the database is backed up before executing an upgrade.

15.3.2 Inmanta-core: release 7.0.0 (2022-08-05)

New features

- Change the relation deprecation warning to be more accurate. (Issue #2443)
- Add support for the elif keyword to the compiler
- Improved tracking of potential future relation assignments within conditional statements.
- Improved error reporting for invalid namespace access (Issue #2818)
- Expressions are now treated as statements (Issue #3367)
- Add environment setting to set the number of stored versions. (Issue #3505)
- Ensure processes forked by Inmanta commands load the same config folder as their parent process (Issue #3765)
- Add notification service (Issue #3981)
- Create a notification when a git pull fails during compile (Issue #4021)
- Add ‘inmanta-cli environment recompile’ command (Issue #4052)
- Added auto_full_compile environment option to schedule regular full compiles (Issue #4274)
- Add support to pass type precedence hints to the compiler (Issue #3098)
- Added support to create development builds of V2 modules (Issue inmanta/irt#1184)

- Added documentation for primitive type casts to the language reference

Improvements

- Improve batching of code loading in the agent (Issue #4217)
- inmanta module v1tov2 and inmanta module build will now merge setup.cfg and pyproject.toml (Issue #4372)
- Add py.typed file to packages build using inmanta module build (Issue #4374)
- The compiler cache (.cfc) files are now stored in the .cfcache directory in the root of the inmanta project instead of in the **cfcache** directory in the inmanta modules. (Issue inmanta/inmanta-core#4407)
- More precise cache invalidation for the compiler cache (Issue #4408)
- Add support to enable/disable strict dependency checking in the compiler and in the module tools using the –strict-deps-check and –no-strict-deps-check options. (Issue #4516)
- Improve exception messages on version conflicts (Issue #4524)
- Improve documentation of agent configuration
- Make `python -m inmanta` work
- Add database connection metrics to the influxdb reporter

Upgrade notes

- The default log level of the `inmanta` commandline tool was changed from ERROR to WARNING (Issue #3911)
- The compiler and the module tools now by default check all dependencies transitively for version conflicts. When a version conflict is found, an error is raised. A fallback to the old behavior is possible by providing the `--no-strict-deps-check` option. (Issue #4516)
- `inmanta project install` and `inmanta project update` now always take into account the `requirements.txt` of the project to provide additional version constraints to pip (Issue #4410)

Deprecation notes

- Unicode characters are no longer escaped in multi-line strings. (Issue #2582)
- The available-versions-to-keep option in the server configuration file is now deprecated in favor of the environment setting (Issue #3505)
- Writing a string over multiple lines is now only supported for strings within triple quotes. This was previously allowed for strings within single quotes due to a bug.
- An exception is now raised when trying to interpolate a string in a dictionary key
- The auto-recompile-wait option in the server configuration is now deprecated in favor of the recompile_backoff environment setting (Issue #4332)

Bug fixes

- The logger now correctly reports the endpoints that will be removed from a session
- Fixed an instance of nondeterministic behavior in the compiler
- Fix memory leak caused by lru-cache keeping strong references to cached items
- Optimize resource list query
- Fix installing extras of module dependencies (Issue #3443)
- Fix bug that fails the CRUDHandler when a changed attribute is of type set. (Issue #3470)
- Wrap any exception that occurs during resource export so that it is more useful to the end user (Issue #3787)
- Writing a string over multiple lines is now only supported for strings within triple quotes.
- An error message is now shown if a wrong repo path is used
- An exception is now raised when there is a mismatch between the python version of the compiler venv and the python version of the active process (Issue #3829)
- Improve the compiler error message that is given when an index attribute is missing in the constructor call. (Issue #3902)
- Fix bug where the user is suggested to run the `inmanta module update` command when the execution of the same command failed. (Issue #3911)
- Fixed bug that makes the `inmanta deploy` command fail when the database and server sections of the inmanta configuration files contain non-default values. (Issue #3927)
- Fix bug that makes every inmanta warning end with an empty line. (Issue #3951)
- Improve syntax error reporting when defining an attribute starting with a capital letter.
- Fix handling of '_' in resource_logs and get_resource_events api endpoints (Issue #4043)
- Fix bug where `inmanta project install` and `inmanta project update` always invokes pip, even when all dependencies are already met. (Issue #4055)
- Limit included namespace packages to inmanta_plugins for v1tov2 module conversion. (Issue #4130)
- Enforce inmanta package requirements so that modules can't overwrite those. (Issue #4200)
- Make sure that the `inmanta project install` command doesn't protect the inmanta-dev-dependencies package (Issue #4249)
- Fix syntax error when calling "is defined" on dictionary lookup
- The set_setting endpoint now correctly returns a 400 status code when an invalid value is provided. (Issue #4361)
- Fix bug where the setup.cfg file, generated by the v1tov2 command, contains a dependency to the module itself when the module contains an import for a namespace in its own module. (Issue inmanta/inmanta-core#4373)
- Fix bug on value lookup in an unknown dict and on lookup with an unknown key. (Issue #4475)
- Fix failing test case.
- Fix order of stages in compile report details (Issue inmanta/web-console#3082)
- Fixed incorrect top level module loading for nested imports when v2 module is present in venv but not in explicit requires
- Fix performance impacting race condition in deploy handler method (Issue inmanta/lsm#433)
- Fix issue with get_resources_in_latest_version call not taking into account versions without resources (Issue inmanta/inmanta-lsm#739)

- Fix issue where the deployment of resources takes a long time, due a high rate limiter backoff. (Issue #4084)
- Fixed type cast behavior for `null` and unknown values

15.3.3 Inmanta-ui: release 3.0.1 (2022-08-05)

No changelog entries.

15.3.4 inmanta-dashboard: release 3.8.1

This component has had no new releases since the last product version.

15.3.5 Web-console: release 1.11.0 (2022-08-05)

New features

- Add support for inter-service relations in the service inventory (Issue #3040)

15.4 Release 2022.1.1 (2022-04-19)

15.4.1 Upgrade notes

- Ensure the database is backed up before executing an upgrade.

15.4.2 Inmanta-core: release 6.0.2 (2022-04-19)

Bug fixes

- Fix bug that crashes the agent when a cross-agent dependency doesn't have any changes (Issue #4116)
- Constrained click dependency to known compatible range because of backwards incompatible minor

15.4.3 Inmanta-core: release 6.0.1 (2022-02-11)

Bug fixes

- Fix bug in incremental deploy where event processing can be delayed (Issue #3789)

15.4.4 inmanta-ui: release 3.0.0

This component has had no new releases since the last product version.

15.4.5 inmanta-dashboard: release 3.8.1

This component has had no new releases since the last product version.

15.4.6 Web-console: release 1.10.0 (2022-04-12)

New features

- Add the Compliance Check page (Issue #2558)
- Add notification drawer (Issue #3056)
- Add notification center page (Issue #3067)

15.4.7 Web-console: release 1.9.1 (2022-02-11)

New features

- Add Desired State Compare page (Issue #2374)

15.5 Release 2022.1 (2022-02-03)

15.5.1 General changes

New features

- Added the web console as the default front-end, replacing the dashboard (Issue #65)
- Introduced the v2 module format. V2 modules offer better integration with the Python ecosystem with regards to distribution, dependency resolution and plugin loading. For more information on v2 modules, see how to add a v2 module source, use a v2 module in your project, and install v2 modules.
- Added support for Python 3.9
- Added deploy method to handlers for increased flexibility in responding to events (Issue inmanta/inmanta-core#2940)
- Added raw strings (r-strings) to the inmanta language (https://docs.inmanta.com/community/latest/language.html#literals-values)
- Added support for Jinja 3 to std module.
- Added terraform module. Allows to use native terraform providers without having to use terraform directly by using the included model generator. (https://docs.inmanta.com/community/latest/reference/modules/terraform.html)
- VSCode extension interacts with the Python extension to allow venv selection.
- Extended web console functionality and made it the default front-end.

Upgrade notes

- Compiling a project no longer installs modules on the fly. Run `inmanta project install` to install modules. For more details see setting up a project.
- The compiler venv (`.env`) is no longer used. The compiler uses the active venv.
- The supported PostgreSQL version is now 13
- The supported Python version is now 3.9
- This release requires RHEL 8
- Jinja templates are required to be compatible with Jinja 3.
- An update of the VSCode extension is required for compatibility with this release.
- Clear your browser cache after upgrading to remove the old redirection rule. If the cache is not cleared the '/' route will keep redirecting to '/dashboard'.
- The compiler and agent venv's with a Python version older than the Python version of the Inmanta server will be moved to an `.rpmsave` directory at installation time. (Issue inmanta/inmanta-service-orchestrator#234)
- Ensure the database is backed up before executing an upgrade.

Deprecation notes

- `inmanta module install` no longer installs all modules for a project. This has moved to `inmanta project install`.
- The inmanta dashboard is now deprecated in favor of the web console. It will be removed in a future major release.

15.5.2 Inmanta-core: release 6.0.0 (2022-02-02)

New features

- Added `resource_deploy_start` endpoint (Issue #2928)
- Added `resource_deploy_done` endpoint (Issue #2931)
- Added helper method for reliable event processing (Issue #2941)
- Improved south bound integration documentation (Issue #2954)
- Compiler improvement: made `is defined` gradually executable
- Added `resource_list` endpoint (Issue #3045)
- Added `resource_details` endpoint (Issue #3046)
- Added support to build V2 modules into a Python package. (Issue #3047)
- Added `resource_history` endpoint (Issue #3048)
- Added the ability to package V1 modules as V2 modules (Issue #3049)
- Added `inmanta module v1tov2` command. (Issue #3050)
- Added V2 package loader (Issue #3051)
- Updated `inmanta module install` to install v2 modules from source.
- Added the `inmanta module add` command. (Issue #3089)

- Added `resource_logs` endpoint (Issue #3109)
- Added endpoint to list compile reports (Issue #3131)
- Added endpoint to get compile details (Issue #3132)
- `inmanta project update` now updates modules' Python dependencies to the latest compatible version. The same goes for triggering an update and recompile from the dashboard. (Issue #3623)
- Enable the UI extension by default (Issue #3653)
- Added version diff api endpoint (Issue #3659)
- Added raw strings to the inmanta language.
- Restructured module developer guide
- added operational procedures documentation
- added instructions about passwordless sudo to remote agent setup
- Added documentation regarding modules V2. (Issue #3023)
- Port the agent to the new `deploy` handler method. (Issue #2940)
- Added support for PostgreSQL 13 (Issue #2893)

Upgrade notes

- On newly created environments, the environment setting `purge_on_delete` will be set to false by default instead of true. This overrides any purge_on_delete settings on individual resources. You need to explicitly set it to true to enable the old behavior again. (Issue #2958)
- `inmanta compile` no longer installs any modules. Run `inmanta project install` before compiling the first time.
- "The compiler venv has been phased out. The compiler will now use the venv used to execute the `inmanta compile` command." (Issue #3096)
- Compiler no longer installs modules on the fly, `inmanta project install` needs to be run to install required modules
- Clear your browser cache after upgrading to remove the old redirection rule. If the cache is not cleared the '/' route will keep redirecting to '/dashboard'. (Issue #3497)
- `Project.load()` no longer installs Project dependencies. Pass `install=True` for the old behavior.
- NOTSET is no longer accepted as a log level by the agent's context logger. It was not a valid log level before, but it was accepted by the agent.
- After upgrading the Inmanta server, all virtual environments used by the compiler and the agents have to be removed. Use the following procedure to achive this:
  - Stop the Inmanta server
  - Remove all `/var/lib/inmanta/server/environments/<environment-id>/.env` directories
  - Remove all `/var/lib/inmanta/<environment-id>` directories
  - Start the Inmanta server again

Deprecation notes

- `inmanta module install` no longer installs all modules for a project. This has moved to `inmanta project install`.
- The `inmanta module list -r` command has been deprecated in favor of `inmanta project freeze`
- `inmanta modules update` has been replaced by `inmanta project update`. The old command has been deprecated and will be removed in a future release. (Issue #3623)

Bug fixes

- Fixed docstring-parser compatibility after non-backwards compatible changes and constrained dependency to semi-safe range.
- Ensure that special characters in the resource action log are not escaped. (Issue inmanta/inmanta-lsm#699)
- Fixed agent cache behavior when `cache_none` is provided
- Fix dollar sign escaping issue in installation documentation
- Fix bug where the listeners of the environment clear action are not notified when files of that environment cannot be deleted from the filesystem. (Issue #3637)
- The tests folder is no longer included into the sdist package
- Removed NOTSET loglevel from all API's

15.5.3 inmanta-ui: release 3.0.0

This component has had no new releases since the last product version.

15.5.4 Inmanta-dashboard: release 3.8.1 (2022-01-25)

No changelog entries.

15.5.5 Inmanta-dashboard: release 3.8.0 (2021-10-18)

New features

- Extend proxy support (Issue #130)

15.5.6 web-console: release 1.9.0

This component has had no new releases since the last product version.

15.6 Release 2021.2.1 (2021-06-01)

15.6.1 Inmanta-core: release 5.1.1 (2021-06-01)

Bug fixes

- Add upperbound to docstring-parser dependency so that pip install does not fail

15.6.2 Inmanta-dashboard: release 3.7.0 (2021-06-01)

No changelog entries.

15.7 Release 2021.2 (2021-05-05)

15.7.1 Inmanta-core: release 5.1.0 (2021-05-05)

New features

- Mark the stable API using a decorator (Issue #2414)
- More strictly validate the schema of the project.yml and module.yml file (Issue #2723)
- Updated db schema update mechanism to track all installed versions (Issue #2724)
- Add partial support for collection type parameters for GET methods (Issue #2775)
- Add changelog section to the documentation (Issue inmanta/irt#417)
- Added developer getting started guide
- Added experimental caching support to the compiler
- Improved Inmanta install guide for Debian
- Extended stable API documentation (Issue inmanta/inmanta-lsm#408)
- Added built-in performance micro-benchmark, to help diagnose performance issues
- Added ability to do `pip install inmanta-core[pytest-inmanta-extension]`

Deprecation notes

- Deprecated yaml dictionary syntax for module requires

Bug fixes

- Correctly describe in the documentation how version constraints can be set on module dependencies in the module.yml file (Issue #2723)
- Ensure that an error at agent startup time is properly logged. (Issue #2777)
- Fixed compiler issue on rescheduling of plugins breaking the cycle breaking (Issue #2787)
- Fixed compiler issue on cycle breaking (Issue #2811)
- Fixed typos in language.rst file
- Changed python versions in install doc

Other notes

- To enable caching on the compiler, either set the config value `compiler.cache` in the `.inmanta` file or pass the option `--experimental-cache` to `inmanta compile`

15.7.2 Inmanta-dashboard: release 3.7.0 (2021-05-05)

No changelog entries.

15.8 Release 2021.1 (2021-02-25)

15.8.1 inmanta-core: 5.0.0 (2021-02-25)

Bug fixes

- Fix broken order by (#2638)
- Report the Inmanta OSS product version correctly (#2622)
- Set PYTHONPATH so that all subprocesses also see packages in parent venv (#2650, #2747)
- Create virtual environments without pip and use the pip of the parent venv
- Correctly set `[:n]` as syntactic sugar for `[0:n]` instead of leaving lower unbound (#2689)

New features

- Add installation procedure for el8 to installation documentation

15.9 Release 2020.6 (2020-12-23)

15.9.1 inmanta-core: 4.0.0 (2020-12-23)

New features

- Add support to use a custom venv path in the Project class (#2466)
- Added more specific location information for attributes (#2481)
- Added plugin call anchors to support ctrl-clicking a plugin call (#1954)
- Added rpdb signal handler (#2170)
- Added pagination support on api calls for agent and agentproc (#2500)
- Added support to build RPMs for a python version different from Python3.6 (#1857)
- Added support for assigning `null` to relations with lower arity 0 (#2459)
- Added documentation on the core dashboard (inmanta/dashboard#63)
- Decouple the compiler version from the OSS product version (#2573)
- Show versions of all installed components when running `inmanta --version` (#2574)

Bug fixes

- Fix broken links in the documentation (#2495)
- Fixed bug in serialization of Resource with Unknowns in collections (#2603)
- Fixed documentation of `install_mode`
- Ensure all running compilations are stopped when the server is stopped (#2508)
- Cleanup old entries in the agentprocess and agentinstance database tables (#2499)
- Ensure the compiler service takes into account the environment variables set on the system (#2413)
- Fix `--server_address` option on `inmanta export` (#2514)
- Handle failure in an event handler consistently for local and non-local agents (#2509)
- Fix for cross agent dependencies responding to unavailable resources (#2501)
- Handle JSON serialization errors in handler log messages (#1875)
- Fixed too restrictive typing (and coercing) of AttributeStateChange (#2540)
- Export command should raise exception on failure (#2487)

Upgrade notes

- Ensure the database is backed up before executing an upgrade.

Other notes

- The inmanta core package is renamed from `inmanta` to `inmanta-core` to allow for true semantic versioning starting at `4.0.0`. A new `inmanta` package is provided that includes `inmanta-core` and continues the `<year>.<minor>[.<patch>]` version schema.

15.10 Release 2020.5 (2020-10-27)

15.10.1 New features

- Added support for environment markers as described in PEP 508 to module requirements parsing (#2359)
- Added design guide to the documentation
- Improved error message when plugin loading fails to include original exception and location (#2385)
- Improved duplicate attributes error message (#2386)

15.10.2 Bug fixes

- Fixed import loop when using `inmanta.execute.proxy` as entry point (#2341)
- Fixed import loop when using `inmanta.resources` as entry point (#2342)
- Clearing an environment with merged compile requests no longer fails (#2350)
- Fixed compiler bug (#2378)
- Fix “compile_data_json_file” referenced before assignment (#2361)
- Fix server-autorecompile-wait config option (#2262)
- Specify the supported values of the ‘format’ parameter of the OpenAPI endpoint explicitly (#2369)
- Fix version cli argument conflict (#2358)
- Don’t remove resource independent parameters on version deletion (#2370)
- Enhance installation documentation (#2241, #2356, #2357)
- Ensure that a protected environment can’t be decommissioned (#2376)
- Don’t load all code on agent start (#2343)
- Allow empty body in else branch for if-else statement (#2375)
- Fixed export failure with null in dict (#2437)
- Fixed small documentation issues
- Only store single agent instance in database for each distinct instance to prevent database overflow when agent rapidly and repeatadly dis- and reconnects (#2394)

15.11 Release 2020.4 (2020-09-08)

15.11.1 New features

- Added merging of similar compile requests to the compile queue (#2137)
- Export all handler's / resource's module's plugin source files so helper functions can be used from sibling modules (#2162, #2312)
- Added documentation on how a string is matched against a regex defined in a regex-based typedef (#2214)
- Added API to query ResourceActions
- Added support to query the resource action log of a resource via the CLI (#2253)
- Added conditional expression to the language with syntax condition ? x: y (#1987)
- Add support for inmanta-cli click plugins
- Added link to the PDF version of the documentation
- Added environment setting for agent_trigger_method (#2025)
- Expose compile data as exported by `inmanta compile --export-compile-data` via API (inmanta/inmanta-telco#54, #2317)
- Added `typedmethod` decorator `strict_typing` parameter to allow `Any` types for those few cases where it's required (#2301)
- Added API method for halting all environment operations (#2228)

15.11.2 Upgrade notes

- Ensure the database is backed up before executing an upgrade.
- Option `inmanta compile --json` is renamed to `inmanta compile --export-compile-data`
- `DynamicProxy.__getattr__` now raises an `AttributeError` instead of a plain `NotFoundException` when an attribute can not be found, for compatibility with Python's builtin `hasattr`. This change is backwards compatible, though it is recommended to except on `AttributeError` over `NotFoundException`. (#2991)

15.11.3 Bug fixes

- Restore support to pass mocking information to the compiler
- Disallow parameters mapped to a header to be passed via the body instead (#2151)
- Handle skipped and unavailable as failures when calculating increments (#2184)
- Constrain agent name to string values (#2172)
- Fix for allowing comments in the requirements.txt file of modules (#2206)
- Allow equality checks between types to support optional value overrides (#2243)
- Don't add path params as query params to the url in the client (#2246)
- Allow Optional as return type for typedmethods (#2277)
- Made Dict- and SequenceProxy serializable to allow exporter to wrap dict and list attributes in other data structures (#2121)

- Improved reporting of `PluginException` (#2304)

15.12 Release 2020.3 (2020-07-02)

15.12.1 New features

- Added cleanup mechanism of old compile reports (#2054)
- Added `compiler.json` option and `--json` compile flag to export structured compile data such as occurred errors (#1206)
- Added troubleshooting documentation (#1211)
- Documentation on compiler API and JSON (#2060)
- Documentation on valid client types (#2015)
- Improved documentation on handler development (#1278)
- Added further documentation to inmanta-cli command (#2057)
- Documentation of config option types (#2072)
- Added method names as Operation Id to OpenApi definition (#2053)
- Added documentation of exceptions to the platform developers guide (#1210)
- Extended documentation of autostarted agent settings (#2040)
- Typing Improvements
- Redirect stdout and stderr to /var/log/inmanta/agent.{out,err} for agent service (#2091)
- Added resource name to log lines in agent log.
- Better reporting of json decoding errors on requests (#2107)
- Faster recovery of agent sessions
- Add compiler entrypoint to get types and scopes (#2114)
- Add support to push facts via the handler context (#593)

15.12.2 Upgrade notes

- Ensure the database is backed up before executing an upgrade.
- Updated Attribute.get_type() to return the full type instead of just the base type (inmanta/inmanta-sphinx#29)
- Overriding parent attribute type with the same base type but different modifiers (e.g. override `number` with `number[]`) is no longer allowed. This was previously possible due to bug (#2132)

15.12.3 Bug fixes

- Various small issues (#2134)
- Fixed issue of autostarted agents not being restarted on environment setting change (#2049)
- Log primary for agent correctly in the database when pausing/unpausing agents (#2079)
- Cancel scheduled deploy operations of an agent when that agent is paused (#2077)
- Fix agent-names config type (#2071)
- Ensure the internal agent is always present in the autostart_agent_map of auto-started agents (#2101)
- Cancel scheduled ResourceActions when AgentInstance is stopped (#2106)
- Decoding of REST return value for content type html with utf-8 charset (#2074)
- Empty list option in config no longer interpreted as list of empty string (#2097)
- Correct closing of agentcache
- Agent cross environment communication bug (#2163)
- Fixed an issue where an argument missing from a request would result in a http-500 error instead of 400 (#2152)
- Ensure agent is in proper state after URI change (#2138)
- Removed warning about collecting requirements for project that has not been loaded completely on initial compile (#2125)

15.13 v 2020.2 (2020-04-24) Changes in this release:

15.13.1 Breaking changes

- Non-boolean arguments to boolean operators are no longer allowed, this was previously possible due to bug (#1808)
- Server will no longer start if the database schema is for a newer version (#1878)
- The environment setting autostart_agent_map should always contain an entry for the agent “internal” (#1839)

15.13.2 Deprecated

- Leaving a nullable attribute unassigned now produces a deprecation warning. Explicitly assign null instead. (#1775)
- Default constructors (typedef MyType as SomeEntityType(some_field = “some_value”)). Use inheritance instead. (#402)
- Old relation syntax (A aa [0:] – [0:] B bb) (#2000)

15.13.3 Fixed

- Various compiler error reporting improvements (#1810, #1920)
- Fixed cache leak in agent when deployments are canceled (#1883)
- Improved robustness of modules update (#1885)
- Removed environmental variables from agent report (#1891)
- Use asyncio subprocess instead of tornado subprocess (#1792)
- Added warning for incorrect database migration script names (#1912)
- Agent manager remains consistent when the database connection is lost (#1893)
- Ensure correct version is used in api docs (#1994)
- Fixed double assignment error resulting from combining constructor kwargs with default values (#2003)
- Fixed recursive unwrapping of dict return values from plugins (#2004)
- Resource action update is now performed in a single transaction, eliminating the possibility of inconsistent state (#1944)
- Type.type_string is now defined as returning the representation of the type in the inmanta DSL (inmanta/lsm#75)

15.13.4 Added

- Experimental data trace, root cause and graphic data flow visualization applications (#1820, #1831, #1821, #1822)
- Warning when shadowing variable (#1366, #1918)
- Added support for compiler warnings (#1779, #1905, #1906)
- Added support for DISABLED flag for database migration scripts (#1913)
- Added v5 database migration script (#1914)
- Added support for declaring implement using parents together with normal implement declaration list (#1971)
- Resource Action Log now includes timestamps (#1496)
- Added support to pause an agent (#1128)
- Added –no-tag option to module tool (#1939)
- Added base exception for plugins and corresponding documentation (#1205)
- Added tags to openapi definition (#1751)
- Added support to pause an agent (#1128, #1982)
- Plugins are now imported in the inmanta_plugins package to allow importing submodules (#507)
- Added event listener to Environment Service (#1996)
- Autostarted agents can load a new value for the autostart_agent_map setting without agent restart (#1839)
- Added protected environment option (#1997)
- Added warning when trying to override a built-in type with a typedef (#81)
- Added inmanta-cli documentation to the docs (#1992)

15.14 v 2020.1 (2020-02-19) Changes in this release:

15.14.1 Fixed

- Added support for conditions as expressions and vice versa (#1815)

15.14.2 Breaking changes

- Entity instances are no longer allowed in list and dict attributes, this was previously possible due to bug (#1435)

15.14.3 Fixed

- Fixed incorrect parsing of booleans as conditions (#1804)
- Added support for nullable types in plugins (#674)
- Inmanta type module cleanup and type coverage
- Various compiler error reporting improvements (#1584, #1341, #1600, #1292, #1652, #1221, #1707, #1480, #1767, #1766, #1762, #1575)
- CRUDHandler bugfix, ensure update is not called on purged resources
- Changes in default values: AUTO_DEPLOY, PUSH_ON_AUTO_DEPLOY are enabled by default, AGENT_TRIGGER_METHOD_ON_AUTO_DEPLOY is set to incremental deployment
- Fixed deadlock triggered by std::AgenConfigHandler (#1662)
- Removed the resourceversionid table from the database (#1627)
- Remote machines not being available or not having a python interpreter now results in a clearer error.
- Parse comments and urls correctly from the requirements.txt file of an Inmanta module (#1764)

15.14.4 Added

- Added support for dict lookup in conditions (#1573)
- Added support for type casts for primitive types (#1798)
- Added support for multiline string interpolations (#1568)
- Added int type to the language (#1568)
- Add get_environment_id to exporter (#1683)
- Added inmanta-cli environment save command (#1666)
- Added finalizer support to @cache annotation
- Added support to parse the docstring of an entity
- Added support for **dict as kwargs for constructor calls and index lookups (#620, #1702)
- Added support for kwargs in plugin calls, as named arguments as well as using **dict (#1143)

15.14.5 Removed

- Removed the inmanta module validate command. Use pytest-inmanta fixtures to test your modules instead.
- Removed Forms functionality (#1667)

15.15 v 2019.5 (2019-12-05) Changes in this release:

15.15.1 Fixed

- Compiler bugfix, ensure done nodes are correctly removed from zerowaiters
- Fixed memory leak in database layer
- Fixed lexing of strings ending in an escaped backslash (#1601)
- Fixed bug where `module freeze` results in empty module.yml (#1598)
- Fixed inconsistent behavior of `export` and `export -j` (#1595)

IMPORTANT CHANGES:

- Added environment variables for config, env variables overwrite all other forms of config (#1507)

v 2019.4 (2019-10-30) Changes in this release:

- Various bugfixes (#1367,#1398,#736, #1454)
- Added if statement (#1325)
- Added CORS Access-Control-Allow-Origin header configuration (#1306)
- Added –version option (#1291)
- Added retry to moduletool update, to allow updating of corrupt projects (#177)
- RPM-based installations on Fedora are not supported anymore
- Added option to configure asyncpg pool (#1304)
- Split out the main service into many smaller services (#1388)
- Use python3 from the core OS in Dockerfile
- Introduce v2 protocol and implement project and environment api in v2 (#1412)
- Improve agent documentation (#1389)
- Improve language reference documentation (#1419)
- Change autostart_agent_deploy_splay_time from 600 to 10 (#1447)
- Introduce the bind-address and bind-port config option (#1442)
- Switch to sequential version numbers instead of timestamps (#1011)
- Fixed memory leak in TaskHandler
- Don’t install packages inherited from the parent virtualenv
- Added logging to CRUD methods of handler and a diff method with context
- HTTP errors are logged at DEBUG level only (#1282)
- Verify hashes when serving a file (#532)

- Mark resource as failed when code loading fails (#1520)
- Print extra env variables in init log and only store those in database (#1482)
- Add feature manager for enabling and disabling orchestrator features (#1530)
- Add get_environment_id to plugin context (#1331)
- Log server bind address and bind port on startup (#1475)
- Fix warning about transport config (#1203)
- Add setting to environment to disable purge on delete (#1546)

IMPORTANT CHANGES:

- Older compiler versions are no longer supported with this server
- The Inmanta server now listens on 127.0.0.1:8888 by default, while this was 0.0.0.0:8888 in previous versions. This behavior is configurable with the `bind-address` config option.

DEPRECATIONS:

- The `server_rest_transport.port` config option is deprecated in favor of the `server.bind-port` option.

v 2019.3 (2019-09-05) Changes in this release:

- Various bugfixes (#1148, #1157, #1163, #1167, #1188)
- Abort server startup if the database can not be reached (#1153)
- Use native coroutines everywhere (async def)
- Updated dockerfile and docker-compose to use postgres and centos
- Added extensions mechanism (#565, #1185)
- Add /serverstatus api call to get version info, loaded slices and extensions (#1184)
- Support to set environment variables on the Inmanta server and its agents
- Split of server recompile into separate server slice (#1183)
- Add API to inspect compiler service queue (#1252)
- Define explicit path in protocol methods
- Added support for schema management for multiple slices in the same database (#1207)
- Marked pypi package as typed
- Create pytest-inmanta-extensions package for extensions testing
- Added support for /etc/inmanta/inmanta.d style configuration files (#183)
- Increased the iteration limit to 10000. This value is controlled with INMANTA_MAX_ITERATIONS environment variable.
- Added support for custom resource deserialization by adding the ‘populate’ method
- Improve compiler scaling by using more efficient data structures
- Added the –export-plugin option to the export command (#1277)
- Only one of set_created, set_updated or set_purged may be called now from a handler
- Remove facts when the resource is no longer present in any version (#1027)
- Successful exports without resources or unknowns will now be exported
- Export plugins will not run when the compile has failed

- Documentation updates and improvements (#1209)

DEPRECATIONS:

- The files /etc/inmanta/agent.cfg and /etc/inmanta/server.cfg are not used anymore. More information about the available configuration files can be found in the documentation pages under `Administrator Documentation -> Configuration files`.

v 2019.2 (2019-04-30) Changes in this release:

- Various bugfixes (#1046, #968, #1045)
- Migration from mongodb to postgres (#1023, #1024, #1025, #1030)
- Added metering using pyformance
- Added influxdb reporter for protocol endpoint metrics
- Remove the configuration option agent-run-at-start (#1055)
- Add project id and environment id as optional parameters to API call (#1001)
- Fixed an issue which cleared the environment on remote python 2 interpreters
- Improve deploy command resilience and added option to work with dashboard
- Added API endpoint to trigger agents deploy (#1052)
- Documentation updates and improvements (#905)

v 2019.1 (2019-03-06) Changes in this release:

- Various bugfixes and performance enhancements (#873, #772, #958, #959, #955)
- Dependency updates
- Introduce incremental deploy (#791, #794, #793, #792, #932, #795)
- Introduce deploying resource state (#931)
- Introduce request_timeout option for transport settings
- Add support to run the compiler on windows
- Add exception explainer to compiler for 'modified after freeze' (#876)
- Improve log format, added replace file name with logger name
- Split out logs, stdout and stderr in autostarted agents (#824, #234)
- Add logging of resource actions on the server and purging of resource actions in the database (#533)
- Improve agent logging
- Replace virtualenv by python standard venv (#783)
- Update to Tornado 5, moving from tornado ioloop to the standard python async framework (#765)
- Use urllib client for fetching jwks public keys
- Remove all io_loop references and only use current ioloop (#847)
- Remove environment directory from server when environment is removed (#838)
- Catch various silent test failures
- Extend mypy type annotations
- Port unit tests to pytest-asyncio and fix deprecation warnings (#743)
- Raise exception on bad export to make inmanta export fail with exit status > 0

- Refactor protocol
- Improve lazy execution for attributes
- Update autogenerated config file for agents with correct server hostname (#892)

DEPRECATIONS:

- Minimal python version is now python 3.6
- Removal of snapshot and restore functionality from the server (#789)
- Removed the non-version api (#526)
- The config option agent-interval, agent-splay, autostart_agent_interval and autostart_splay are deprecated in favour of agent-deploy-interval, agent-deploy-splay-time, autostart_agent_deploy_interval and autostart_agent_deploy_splay_time respectively. The deprecated options will be removed in release 2019.2

v 2018.3 (2018-12-07) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- Added improved error reporting to CLI (#814)
- Fixed missing re-raise on pip install (#810)
- Add pytest plugins (#786)
- Extra test cases for the data module + two bugfixes (#805)
- Fix deprecation warnings (#785)
- Reorganized test case in more modules to reduce the number of merge conflicts (#764)
- Prevent purge_on_delete due to failed compile (#780)
- Add mypy to tox and improve typing annotations (no enforcement yet) (#763)
- Removed incorrect uninitialize of subprocess signal handler (#778, #777)
- Fix modules do command (#760)
- Changed process_events so that it is called even when processing a skip. (#761)
- Track all locations where an instance has been created. (fixes #747)
- Add start to the index for the get_log query (#758)
- Improved reporting of nested exceptions (#746)
- Added compiler check on index attributes so an index on a nullable attribute now raises a compiler error. (#745)
- Added support for lazy attribute execution in constructors (#729)
- Big update to module and project version freeze. See documentation for more details (#106)
- Added argument to @plugin to allow unknown objects as arguments (#754)
- Fix for deploy of undefined resource (#627)
- Improved handling ofr dryrun failures (#631)
- Correctly store and report empty facts (#731)
- Allow get facts from undeployed or undefined resources (#726)
- Minor changes for ide alpha release (#607)
- Added uniqueness check to indices (#715)

- Bugfixes in handling of optional attributes (#724)
- Transport cleanup (added bootloader, split off session management) (#564)
- Reserved keywords in resources (#645)
- Fix a bug in option definition
- Use own mongobox implementation that works with mongo >= 4
- Fixed reporting on undefined list attributes (#657)
- Improved list freeze for gradual execution (#643)
- Fixed bug in bounds check (#671)
- Improved error reporting on bad assignment (#670)
- Improved error reporting on missing type (#672)
- Added in operator for dicts (#673)

v 2018.2 (2018-07-30) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- The internal storage format for code is optimized. This introduces API and schema changes. This release supports both storage versions. The old version will be removed in the next release.
- Support formatter in repo url
- Make export of complete model configurable
- Use id of loopvar instead of hash to support iteration over list returned by plugins
- Fix error in default args for list attribute (#633)
- Add multi level map lookup (#622 and #632)
- Improved deploy, make deploy sync
- Added improved error message for lower bound violations on relations (#610)
- Fixes for empty optionals (#609)
- Added improved logging to context handler (#602)
- Added fix for string representation (#552)
- Added support for single quotes (#589)
- Fix in operator in typedefs (#596)
- Fixed line numbers on MLS (#601)
- Added += operator for assignment to lists (#587)
- Add a synchronous protocol client
- Fix error message for wrong type in ctor
- Improve index error reporting
- Fix validate on modules with no commited version
- Set purged=false on clone in CRUDHandler (#582)
- Add gzip encoding support to protocol (#576)

- added anchormap functions to compiler
- Improved error reporting on for loops (#553)

v 2018.1 (2018-02-09) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- Ubuntu 14.04 mongo (2.4) is no longer supported. Version 2.6 or higher is required.
- The inmanta API endpoint is now versioned and available under /api/v1. The old API methods still work, but are deprecated and will be removed in the next release.
- Added support for escapes in regex (#540)
- Added per env config for agent_interval (#542): This adds an per environment setting that controls the agent interval for the agents started by the server.
- Removed implicit string to number conversion (#539)
- Fix dockerfile (#538)
- Fixed execnet resource leak (#534)
- Solution for resource leak issue in agent (#518): Numerous stability fixes for the agent related to resource leaks and races
- Remove compile reports on env clean
- Refactor report API: The report list no longer contains the output of the processes. This reduces the size of the response.
- Fix recompile triggered from a form change
- Add missing mongo indexes to improve performance
- Remove catchlog from tox run
- Create a post method for notify: only the post method allows to pass metadata
- Fix trigger metadata (#520): Add compile metadata to each version. Fixes #519 and add delete with resource_id for parameters
- Add representation for null value

v 2017.4 (2017-11-27) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- added keyword parents, and implemented implementation inheritance (#504)
- set_param recompile parameter
- Raise an exception when duplicate resources are exported (#513)
- Added fix for index issue (#512)
- Allow to configure server compile per environment
- Add remove parameter API call
- Attributes and lists now accept trailing comma (#502)
- Added check for attribute redefinition within one entity (#503)
- Parse bool values in the rest api

- Fix bug in dryrun reporting with auth enabled

v 2017.3 (2017-10-27) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- Add relation annotations to the relation attribute and resolve it for exporters to use
- Documentation improvements
- Add an undefined resource state to the server (#489) Previously all unknown handling was done in the server. This resulted in strange reporting as the number of managed resource could go up and down. Now, an additional resource state "undefined" is introduced. This state is handled similar to skipped during deploys. Undefined resources are undeployable.
- Undeployable resources are now already marked as finished at the moment a version is released or a dryrun is requested. Resources that depend on resources in an undeployable state will be skipped on the server as well.
- Sort index attributes: This patch ensure that std::File(host, path) and std::File(path, host) are the same indexes.
- Improved modules list ouput: rename columns and added a column to indicate matching rows
- Improve attribute check. fixes (#487)
- Fix index issues related with inheritance (#488)
- When a resource is purged, its facts will be removed. (#3)
- Add location to type not found exception in relation (#475. #294)
- Add JWT authz and limitation to env and client type (#473)
- Added fix for function execution in constraints (#470)
- Each agent instance now has its own threadpool to execute handlers. (#461)
- Allow agent instances to operate independently (#483)
- Improved error reporting on parser errors (#468, #466)
- Fixed selection of lazy arguments (#465)

v 2017.2 (2017-08-28) Changes in this release:

- Various bugfixes and performance enhancements
- Dependency updates
- Preserve env variables when using sudo in the agent
- Prune all versions instead of only the ones that have not been released.
- Use python 2.6 compatible syntax for the remote io in the agent
- Gradual execution for for-loops and constructors
- Stop agents and expire session on clear environment
- Improve purge_on_delete semantics
- New autostart mechanism (#437)
- Add settings mechanism to environment. More settings will become environment specific in later releases.
- Do not create index in background to prevent race conditions
- Add support for exception to the json serializer

- Invert requires for purged resources (purge_on_delete)
- Add autodeploy_splay option
- Remove ruaml yaml dependency (#292)
- Handle modified_count is None for mongodb < 2.6
- Add python3.6 support
- Add nulable types
- Various documentation updates
- Added monitor command to inmanta-cli (#418)
- Generate inmanta entrypoint with setuptools
- Update quickstart to use centos
- Improve event mechanism (#416)
- Added auto newline at end of file (#413)
- Improved type annotations for plugins and improved object unwrapping (#412)
- Inline index lookup syntax (#411)
- Added cycle detection (#401)
- Fixed handling of newlines in MLS lexer mode (#392)
- Added docstring to relations, typedef, implementation and implement (#386)
- Fix agent-map propagation from deploy

v 2017.1 (2017-03-29) New release with many improvements and bug fixes. Most noteable features include:

- Port CLI tool to click and improve it. This removes cliff and other openstack deps from core
- Complete rewrite of the database layer removing the dependency on motorengine and improve scalability.
- Cleanup of many API calls and made them more consistent
- Improved handler protocol and logging to the server.

v 2016.6 (2017-01-08) Mainly a bugfix and stabilisation release. No new features.

v 2016.5 (2016-11-28) New release with upgraded server-agent protocol

- Upgraded server agent protocol
- New relation syntax

v 2016.4 (2016-09-05) New relase of the core platform

- Various compiler improvements
- Add list types
- Cleanup of is defined syntax in the DSL and templates
- Many additional test cases
- Various bugfixes

v 2016.3 (2016-08-18) New release. Way to late due to kids and vacation.

- Added SSL support
- Added auth to server

- Add JIT loading of modules
- Various bug fixes

v 2016.2.3 (2016-05-30)

- Fix memory leak in server

v 2016.2.2 (2016-05-25)

- Remove urllib3 dependency to ease packaging on el7

v 2016.2.1 (2016-05-04)

- Various bugfixes related to new ODM and async IO

v 2016.2 (2016-05-02)

- First bi-monthly release of Inmanta
- New compiler that speeds up compilation an order of magnitude
- All RPC is now async on the tornado IOLoop
- New async ODM for MongoDB
- Increased test coverage

CHAPTER

SIXTEEN

ADDITIONAL RESOURCES

- Inmanta User Mailinglist
- Inmanta Developer Mailinglist
- Inmanta Twitter

CHAPTER

SEVENTEEN

PDF VERSION

Download: `inmanta.pdf`

PYTHON MODULE INDEX

i

INDEX

Symbols

A

B

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W

Y