Inmanta Documentation

Release 2021.2

Inmanta NV

May 05, 2021

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

This tutorial gets you started with the Inmanta orchestration tool.

Inmanta is intended to manage complex infrastructures, often in the cloud or other virtualized environments. In this guide, we go for a less complex setup: install the Drupal CMS on two VM-like containers. First, we use Docker to set up a basic environment with two empty VM-like containers, an Inmanta server and a postgres server used by inmanta as a database. Then, we use Inmanta to install Drupal on these VM-like containers.

Note: This is meant to get an example Inmanta environment set up and running quickly 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 Setting up the tutorial

To quickly get started with Inmanta, use Docker Compose to set up an environment to host the Inmanta server and some machines to be managed. Before starting this tutorial, first install Docker on your machine. Next install Docker Compose on your machine.

Then, grab the Docker quickstart from our Git repository.

```
git clone https://github.com/inmanta/quickstart-docker.git
cd quickstart-docker
```

Now that we have the needed docker files, we will need to get the Inmanta quickstart project itself:

```
git clone https://github.com/inmanta/quickstart.git quickstart-project
```

The quickstart project can now be found under the newly created *quickstart-project* directory. It will be the basis for this quickstart. The `quickstart-project` directory will also be shared with the Inmanta server container (mounted to `/home/inmanta/quickstart-project`). We will come back to the files in this repository later.

Note: If you are on *Windows*, be sure you make the drive with the quickstart project shareable with docker containers:

1. In Powershell: `$env:COMPOSE_CONVERT_WINDOWS_PATHS = 1`
2. Restart Docker for Windows
3. Go to Docker for Windows settings > Shared Drives > Reset credentials > select drive with quickstart project > set your credentials > Apply

Finally, have Docker Compose deploy the quickstart environment:

```
docker-compose up
```

Docker Compose will set up the Inmanta server, a postgres server and two VM-like containers to experiment on. When Docker Compose is done deploying and the Inmanta server is running, you will be able to open the dashboard at http://127.0.0.1:8888. When you see the following output, the Inmanta server is ready to be used:

```
inmanta_quickstart_server | inmanta.protocol.rest    DEBUG   Start REST transport
inmanta_quickstart_server | inmanta                 INFO    Server startup complete
```

Note: docker-compose will lock the current terminal and use it for output from all 4 containers. You will need to open a new terminal to continue with this quickstart

To get an interactive shell on the Inmanta server (this will be needed later):

```
docker exec -it "inmanta_quickstart_server" bash
```

Note: The rest of the quickstart guide assumes commands are executed from the root path of the quickstart-docker Git repository, unless noted otherwise.

1.1.1 Breaking down/Resetting the quickstart-docker environment

To fully clean up or reset the environment, run the following commands:

```
docker-compose down
docker volume prune -f
docker image rmi inmanta-agent inmanta-server
```

This will give you a clean environment next time you run `docker-compose up`.

1.2 Automatically deploying Drupal

At this point, you can go through the quickstart guide in one of two ways: via the dashboard or via the command line interface. For the CLI, go to the next section. For the Dashboard, go to *Using the dashboard*.

1.2.1 Single machine deployment using the CLI

To start a new project, all you need is a directory with a project.yml file, defining the parameters like location to search for modules and where to find the server. In this case we will be using the premade quickstart project we cloned in to `./quickstart-project` earlier.

That directory contains a project.yml, which looks like this:

```
name: quickstart
modulepath: libs
downloadpath: libs
repo: https://github.com/inmanta/
description: A quickstart project that installs a drupal website.
requires:
```

(continues on next page)

(continued from previous page)

```
    - apache ~= 0.3.1
    - drupal ~= 0.7.1
    - exec ~= 1.1.0
    - ip ~= 1.0.0
    - logging ~= 0.4.1
    - mysql ~= 0.6.0
    - net ~= 0.5.0
    - php ~= 0.3
    - redhat ~= 0.8.0
    - std ~= 0.26.2
    - web ~= 0.2.2
    - yum ~= 0.5.1
```

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

In the next section we will use existing modules to deploy a LAMP stack.

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 automatically.

The configuration model

In this section we will use the configuration concepts defined in the existing modules to set up Drupal on the host named `vm1`.

First delete the contents of `./quickstart-project/main.cf`, then put in the following:

```
import ip
import redhat
import redhat::epel
import apache
import mysql
import web
import drupal

# define the machine we want to deploy Drupal on
vm1=ip::Host(name="vm1", os=redhat::centos7, ip="172.28.0.4", remote_agent=true,␣
↪remote_user="root")

# add a mysql and apache http server
web_server=apache::Server(host=vm1)
mysql_server=mysql::Server(host=vm1, remove_anon_users=true)

# deploy drupal in that virtual host
name=web::Alias(hostname="localhost")
db=mysql::Database(server=mysql_server, name="drupal_test", user="drupal_test",␣
↪password="Str0ng-P433w0rd")
drupal::Application(name=name, container=web_server, database=db, admin_user="admin",
```

(continues on next page)

(continued from previous page)

```
20                      admin_password="test", admin_email="admin@example.com",
21                      site_name="localhost")
```

- Lines 1-7 import all the required packages.
- Line 10 defines on which machine we want to deploy Drupal.
  - The *name* attribute is the hostname of the machine, which is later used to determine what configuration needs to be deployed on which machine.
  - The *os* attribute defines which operating system this server runs. This is used to select the right tools (yum or dnf or apt).
  - The *ip* attribute is the IP address of this host. At this moment we define this attribute manually, later in this tutorial we let Inmanta discover this automatically.
- Line 13 deploys an Apache server on our host.
- Line 14 deploys a Mysql server on our host and removes its anonymous users.
- Line 17 defines the name (hostname) of the web application.
- Line 18 defines a database for our Drupal website.
- Lines 19-21 define the actual Drupal application.

Deploy the configuration model

To deploy the project, we must first register it with the management server by creating a project and an environment. A project is a collection of related environments. (e.g. development, testing, production, qa,...) An environment is associated with a branch in a git repository. This allows the server to recompile the model when the environment changes.

Connect to the terminal of the server-container:

```
docker exec -it "inmanta_quickstart_server" bash
```

Then, create the inmanta project and environment:

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

Note: The `--save` option tells `inmanta-cli` to store the environment config in the `.inmanta` file. The compiler uses this file to find the server and to export to the right environment.

Finally compile the project and deploy it:

```
inmanta -vvv  export -d
```

The first time you run this command, it may take a while, as all dependencies are downloaded.

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 *quickstart-env* environment. When the deployment fails for some reason, consult the *troubleshooting page* to investigate the root cause of the issue.

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

Accessing your new Drupal server

When the installation is done, you can access your new Drupal server at http://localhost:8080/.

1.2.2 Multi-machine deployment using the CLI

The real power of Inmanta becomes apparent when managing more than one machine. In this section we will move the MySQL server from `vm1` to a second machine called `vm2`.

Update the configuration model

A second machine is easily added to the system by adding the definition of the machine to the configuration model and assigning the MySQL server to the new machine.

Update `main.cf` to the following:

```
1  import ip
2  import redhat
3  import redhat::epel
4  import apache
5  import mysql
6  import web
7  import drupal
8
9  # define the machine we want to deploy Drupal on
10 vm1=ip::Host(name="vm1", os=redhat::centos7, ip="172.28.0.4", remote_agent=true,␣
   ↪remote_user="root")
11 vm2=ip::Host(name="vm2", os=redhat::centos7, ip="172.28.0.5", remote_agent=true,␣
   ↪remote_user="root")
12
13 # add a mysql and apache http server
14 web_server=apache::Server(host=vm1)
15 mysql_server=mysql::Server(host=vm2)
16
17 # deploy drupal in that virtual host
18 name=web::Alias(hostname="localhost")
19 db=mysql::Database(server=mysql_server, name="drupal_test", user="drupal_test",␣
   ↪password="Str0ng-P433w0rd")
20 drupal::Application(name=name, container=web_server, database=db, admin_user="admin",
21                     admin_password="test", admin_email="admin@example.com", site_name=
   ↪"localhost")
```

On line 11 the definition of the new machine is added. On line 15 the MySQL server is assigned to vm2.

Deploy the configuration model

To deploy the configuration model, compile the project and deploy it. In the Inmanta server container terminal:

```
inmanta -vvv export -d
```

If you browse to the Drupal site again, the database should be empty once more. When the deployment fails for some reason, consult the *troubleshooting page* to investigate the root cause of the issue.

Note: When moving the database, a new database is created and the content of the old database is not migrated automatically.

1.2.3 Using the dashboard

Inmanta can deploy from the server using only the dashboard. All changes have to go through the repository in this case.

1. Clone the quickstart project on github (or to another repository location).
2. Go to the dashboard.
3. Create a new project with the name `test` by clicking *Add new project*.
4. Go into the new project and create a new environment by clicking *Add new environment*:
   - Select the `test` project.
   - Give the environment a name, e.g. `env-quickstart`.
   - Specify the repo: for example `https://github.com/user/quickstart`.
   - Specify the branch: `master`.
5. Checkout your clone of the quickstart repository and make changes to the main.cf file, for example add the contents of single_machine.cf to the main.cf file. Commit the changes and push them to your repository.
6. Go into your new environment.
7. Press *Update & Recompile* (this may take a while, as all dependencies are downloaded).
   - Now the Inmanta server downloads the configuration model from your clone of the repository. It also downloads all required modules (i.e. dependencies). These modules contain the instructions to install specific parts of the setup such as for example *mysql* or *drupal* itself. To see the source go here, for a more in-depth explanation *see above*.
   - When this is done, it compiles all modules and integrates them into a new deployment plan.
8. When the compilation is done, a new version appears. This contains the new deployment plan. Click on this version to open it. This shows a list of all configuration items in this configuration.
9. Press *Deploy* to start rolling out this version.
   - An agent is now started that remotely logs in into the virtual machines (via SSH) and starts deploying the Drupal server.
   - It will automatically install the required software and configure it properly.
10. When the deployment is done, you can find your freshly deployed Drupal instance at http://localhost:8080/.

1.3 Create your own modules

Inmanta enables developers of a configuration model to make it modular and reusable. In this section we will create a configuration module that defines how to deploy a LAMP stack with a Drupal site in a two- or three-tiered deployment.

1.3.1 Module layout

A configuration module requires a specific layout:

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

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

We will create our custom module in the `libs` directory of the quickstart project. Our new module will be called *lamp*, and we require the `_init.cf` file (in the `model` subdirectory) and the `module.yml` file to have a valid Inmanta module. The following commands create all directories and files to develop a full-featured module:

```
mkdir ./quickstart-project/libs/{lamp,lamp/model}
touch ./quickstart-project/libs/lamp/model/_init.cf
touch ./quickstart-project/libs/lamp/module.yml
```

Note: Running into permission errors at this point is normal if you followed the cli version of the quickstart. The best way to resolve these is to `sudo mkdir ./quickstart-project/libs/lamp` and then `sudo chmod -R 777 ./quickstart-project/libs/lamp`. Now run the above commands again.

Next, edit the `./quickstart-project/libs/lamp/module.yml` file and add meta-data to it:

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

1.3.2 Configuration model

In `./quickstart-project/libs/lamp/model/_init.cf` we define the configuration model that defines the *lamp* configuration module.

```
1  import ip
2  import apache
3  import mysql
4  import web
5  import drupal
6
7  entity DrupalStack:
8      string hostname
9      string admin_user
10     string admin_password
11     string admin_email
12     string site_name
13 end
14
15 index DrupalStack(hostname)
16
17 DrupalStack.webhost [1] -- ip::Host
18 DrupalStack.mysqlhost [1] -- ip::Host
19
20 implementation drupalStackImplementation for DrupalStack:
21     # add a mysql and apache http server
22     web_server=apache::Server(host=webhost)
23     mysql_server=mysql::Server(host=mysqlhost)
24
25     # deploy drupal in that virtual host
26     name=web::Alias(hostname=hostname)
27     db=mysql::Database(server=mysql_server, name="drupal_test", user="drupal_test",
28                        password="Str0ng-P433w0rd")
29     drupal::Application(name=name, container=web_server, database=db, admin_
   ↪user=admin_user,
30                         admin_password=admin_password, admin_email=admin_email, site_
   ↪name=site_name)
31 end
32
33 implement DrupalStack using drupalStackImplementation
```

- Lines 7 to 13 define an entity which is the definition of a *concept* in the configuration model. On lines 8 to 12, typed attributes are defined which we can later on use in the implementation of an entity instance.
- Line 15 defines that *hostname* is an identifying attribute for instances of the DrupalStack entity. This also means that all instances of DrupalStack need to have a unique *hostname* attribute.
- Lines 17 and 18 define a relation between a Host and our DrupalStack entity. The first relation reads as follows:
  - Each DrupalStack instance has exactly one ip::Host instance that is available in the webhost attribute.
  - Each ip::Host has zero or one DrupalStack instances that use the host as a webserver. The DrupalStack instance is available in the drupal_stack_webhost attribute.

- On lines 20 to 31 an implementation is defined that provides a refinement of the DrupalStack entity. It encapsulates the configuration of a LAMP stack behind the interface of the entity by defining DrupalStack in function of other entities, which on their turn do the same. Inside the implementation the attributes and relations of the entity are available as variables.
- On line 33, the *implement* statement links the implementation to the entity.

1.3.3 The composition

With our new LAMP module we can reduce the amount of required configuration code in the `./quickstart-project/main.cf` file by using more *reusable* configuration code. Only three lines of site-specific configuration code are required.

```
1  import ip
2  import redhat
3  import redhat::epel
4  import lamp
5
6  # define the machine we want to deploy Drupal on
7  vm1=ip::Host(name="vm1", os=redhat::centos7, ip="172.28.0.4", remote_agent=true,␣
   ↪remote_user="root")
8  vm2=ip::Host(name="vm2", os=redhat::centos7, ip="172.28.0.5", remote_agent=true,␣
   ↪remote_user="root")
9
10 lamp::DrupalStack(webhost=vm1, mysqlhost=vm2, hostname="localhost", admin_user="admin
   ↪",
11                   admin_password="test", admin_email="admin@example.com", site_name=
   ↪"localhost")
```

1.3.4 Deploy the changes

Deploy the changes as before, by connection to the servers terminal. Nothing will change because the generated configuration should be exactly the same.

```
inmanta -vvv export -d
```

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

1.4 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

CentOS 7

For CentOS 7 use yum:

```
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 yum install -y epel-release
sudo yum 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 dashboard is installed with the server package.

CentOS 8

For CentOS 8 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
```

(continues on next page)

(continued from previous page)

```
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 epel-release
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 dashboard is installed with the server package.

Debian, Ubuntu and derivatives.

First make sure Python >= 3.6 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 >= 3.6 and pip
sudo apt-get update
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 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 file named `inmanta.cfg` (this name is arbitrary) in your virtual env:

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

If you want to use the dashboard you need to install it as well:

Get the pre-built package from our github page click on the latest release and 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 `[dashboard]` section, change the `path` value to the `dist` directory of where you extracted the pre-built package. For instance:

```
path=/opt/inmanta/package/dist
```

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

```
inmanta -vv -c /opt/inmanta/inmanta.cfg server
```

Other

First make sure Python >= 3.6 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 >= 3.6 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 inmanta
sudo /opt/inmanta/bin/inmanta --help
```

Install PostgreSQL using this guide

Download the configuration file named `inmanta.cfg` (this name is arbitrary) in your virtual env:

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

If you want to use the dashboard you need to install it as well:

Get the pre-built package from our github page click on the latest release and 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 `[dashboard]` section, change the `path` value to the `dist` directory of where you extracted the pre-built package. For instance:

```
path=/opt/inmanta/package/dist
```

Then the dashboard can be started using below command:

```
inmanta -vv -c /opt/inmanta/inmanta.cfg 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 >= 3.6 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 >= 3.6 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 (7 or 8) or CentOS (7 or 8) 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 10

PostgreSQL 10 can be installed by following the installation guide for your platform.

Step 3: Setup a PostgreSQL database for the Inmanta server

Initialize the PostgreSQL server:

```
sudo /usr/pgsql-10/bin/postgresql-10-setup initdb
```

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

```
sudo systemctl enable --now postgresql-10
```

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/10/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-10
```

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 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.

2.3.1 Auto-started agents

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

Requirements

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

- 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`.

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 Python2 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.

2.3.2 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` and the `inmanta-oss-agent` packages are 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://pkg.inmanta.com/inmanta-oss-stable/el7/
gpgcheck=1
gpgkey=https://pkg.inmanta.com/inmanta-oss-stable/inmanta-oss-stable-public-key
repo_gpgcheck=1
enabled=1
enabled_metadata=1
EOF

sudo yum install -y epel-release
sudo yum install -y inmanta-oss 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 autostarted agents.

Step 3: Starting the manually-started agent

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

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

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

CHAPTER

THREE

DASHBOARD DOCUMENTATION

3.1 The project overview

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

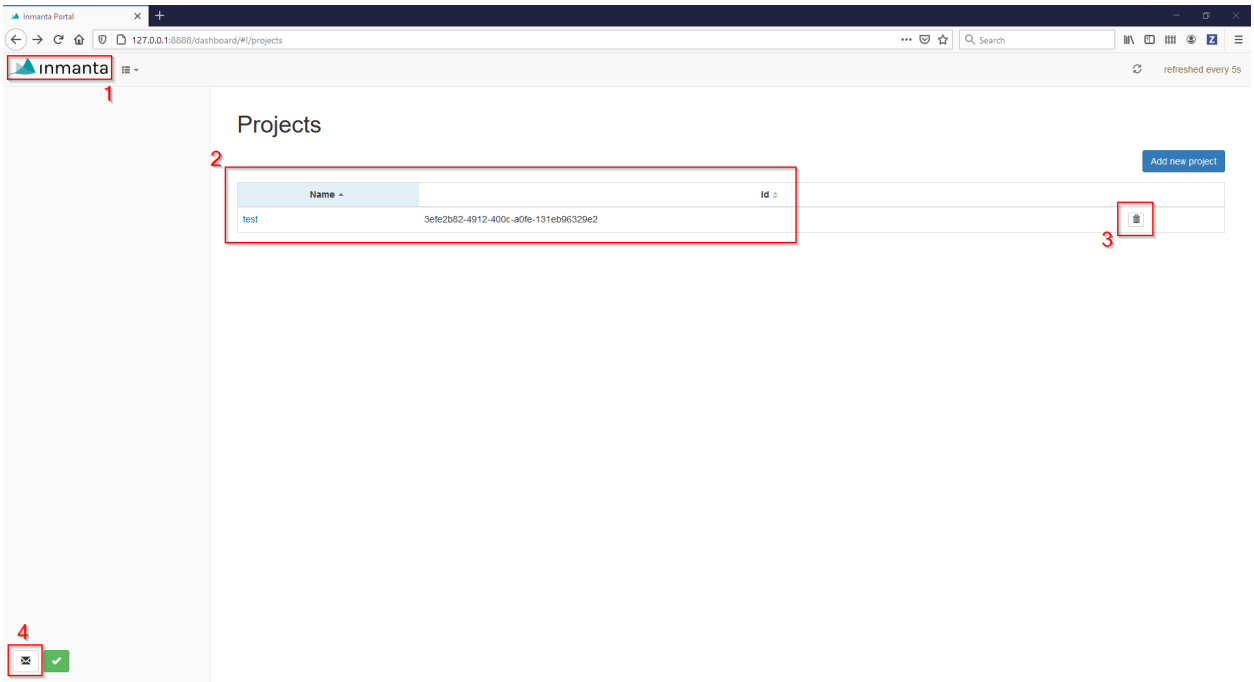


Fig. 1: The project overview page

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.
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.

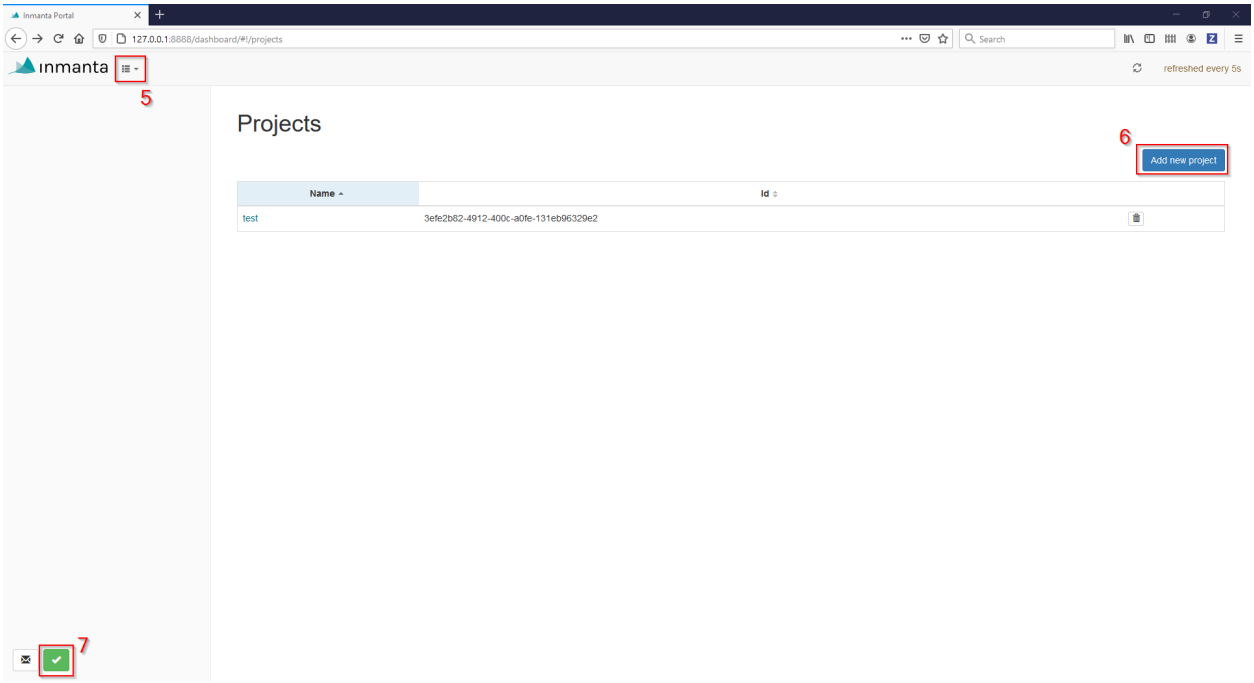


Fig. 2: The project overview page

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.

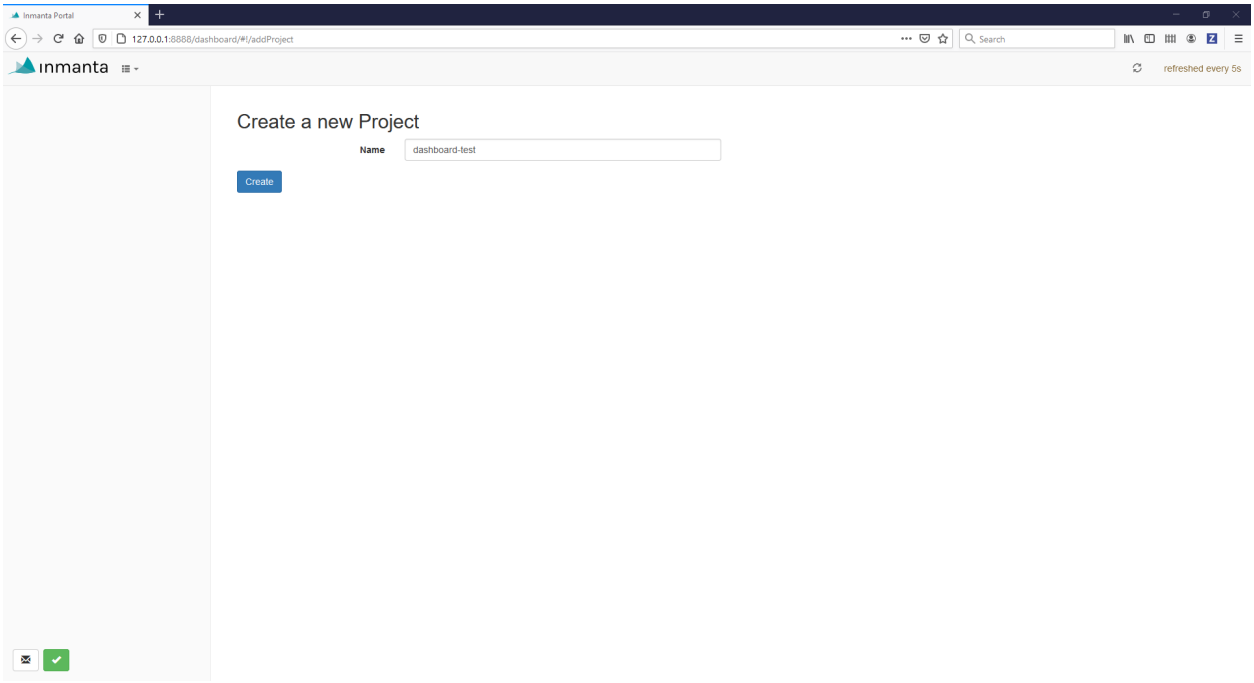


Fig. 3: Adding a new project

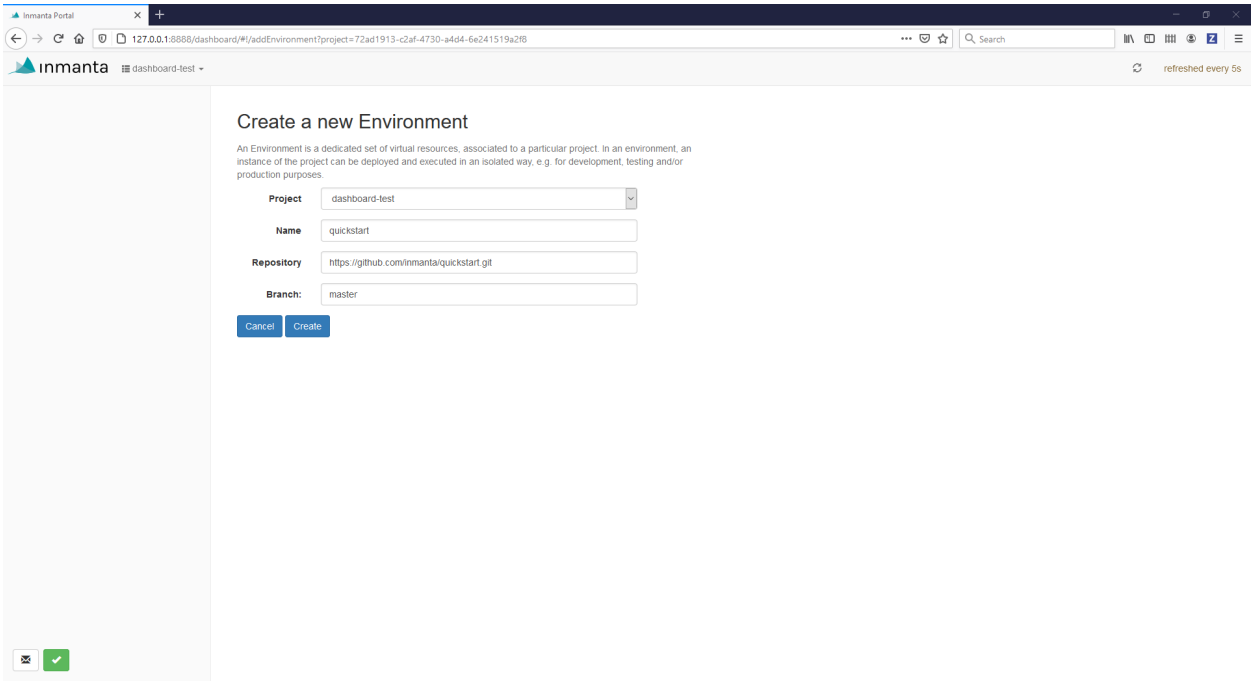


Fig. 4: Creating a new Environment

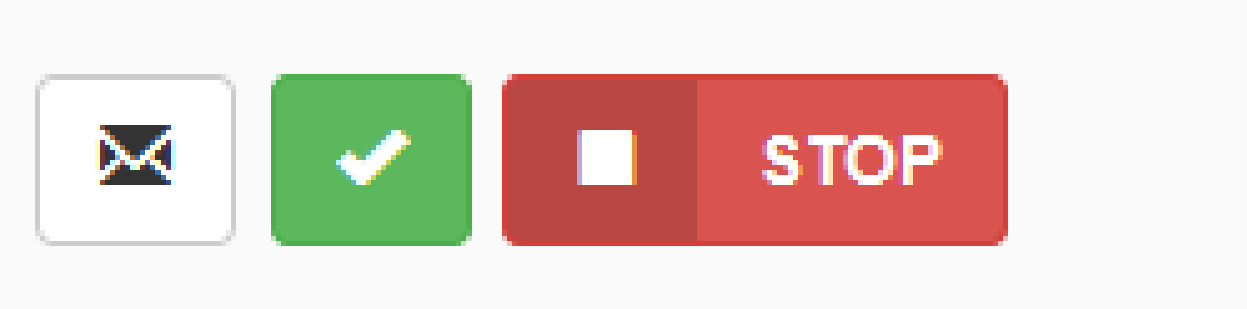


Fig. 5: The emergency stop button

3.3 The Environment Portal

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

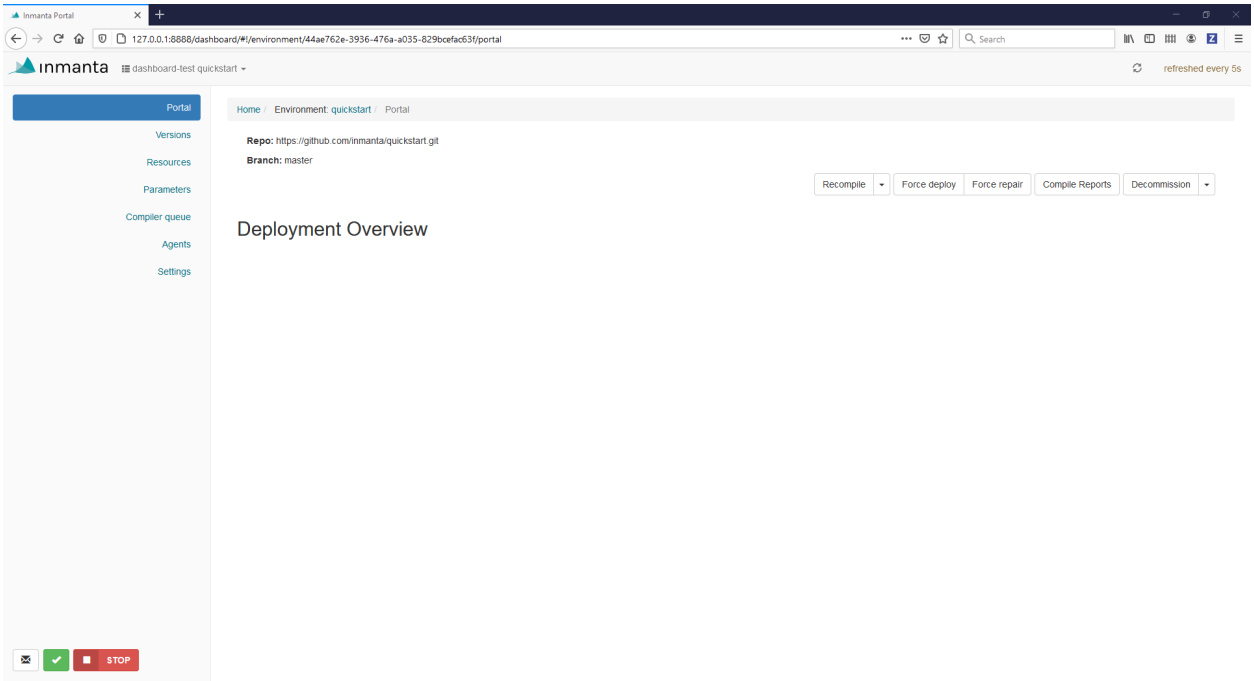


Fig. 6: A newly created environment

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:

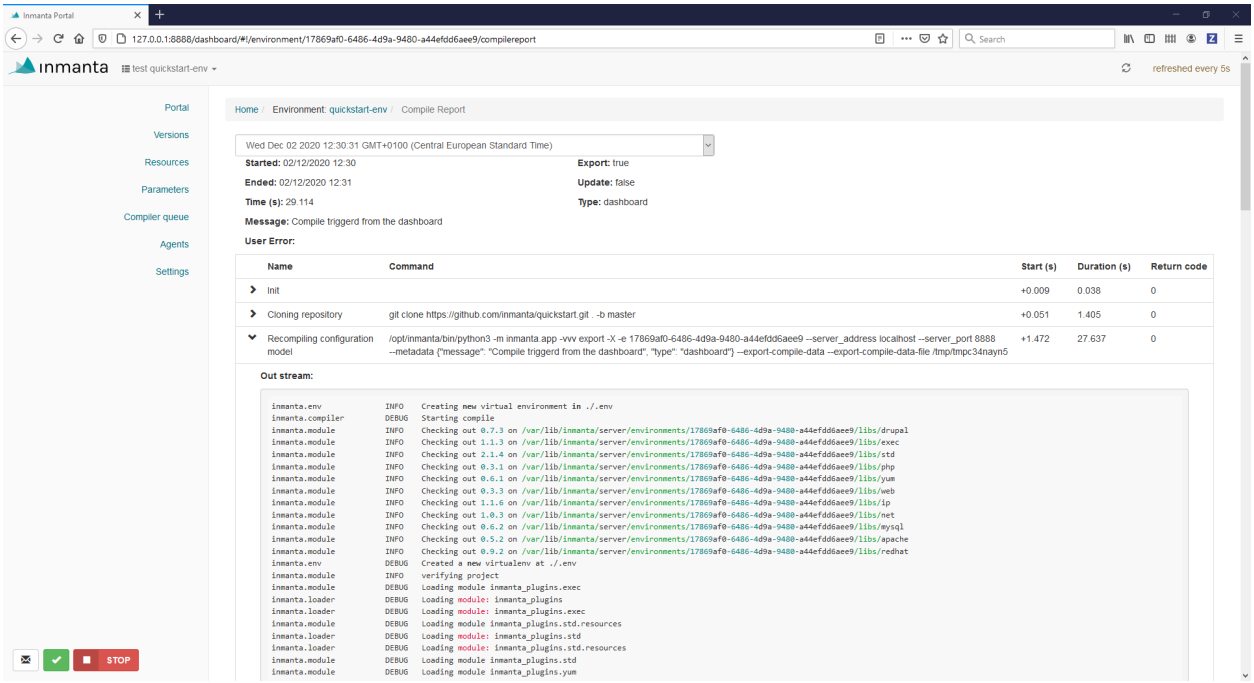


Fig. 7: A compile report

- 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` drop-down menu:

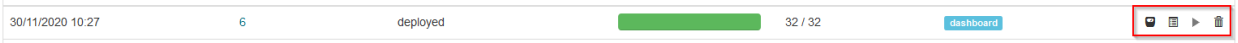
- Decommission: pushes a model that purges all resources deployed by the model.
- Edit: change the configuration of the environment, such as the git repo url or what branch to use.
- Clone: create a new environment using the same git repo and branch
- Clear: Clears the environment. This will remove all 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.

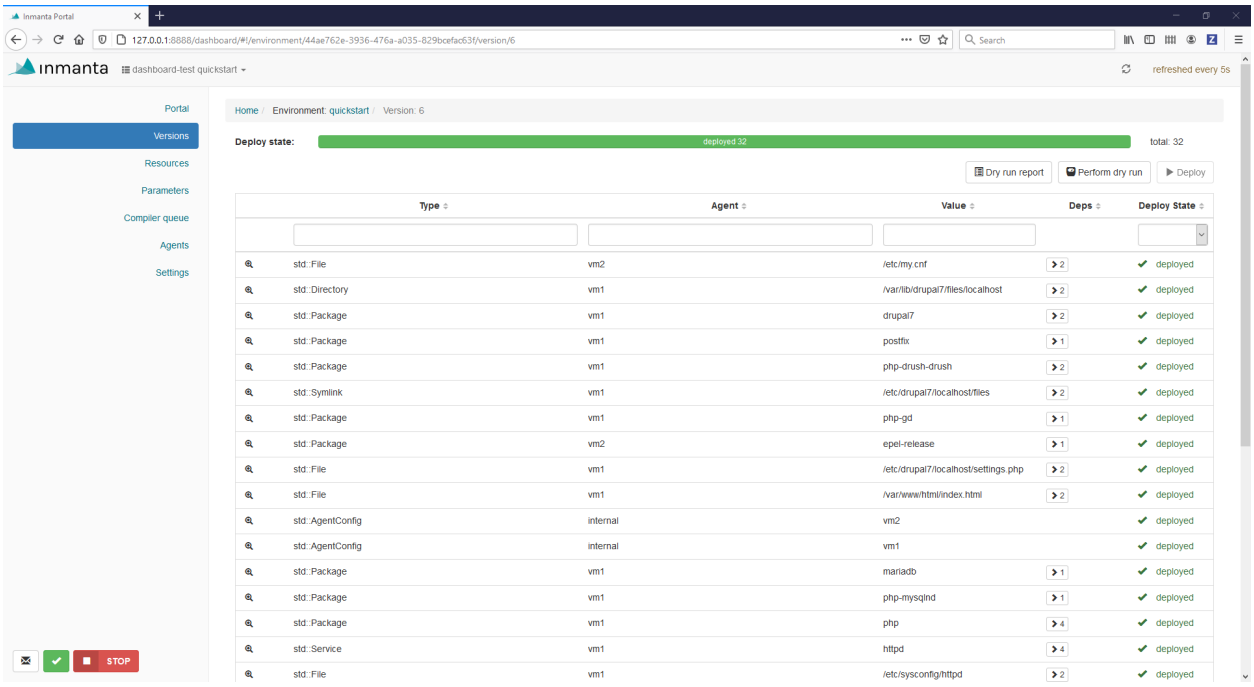
Each version has 4 buttons on the right to interact with it:



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.

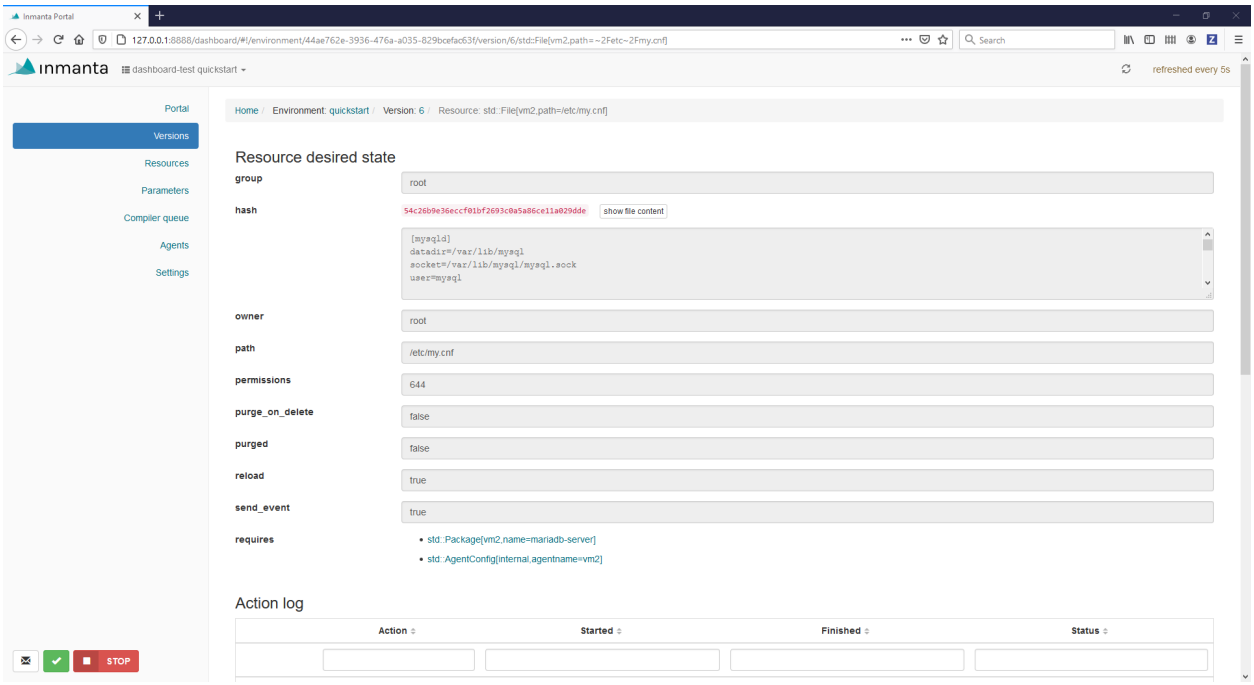


Fig. 9: Resource desired state view

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.

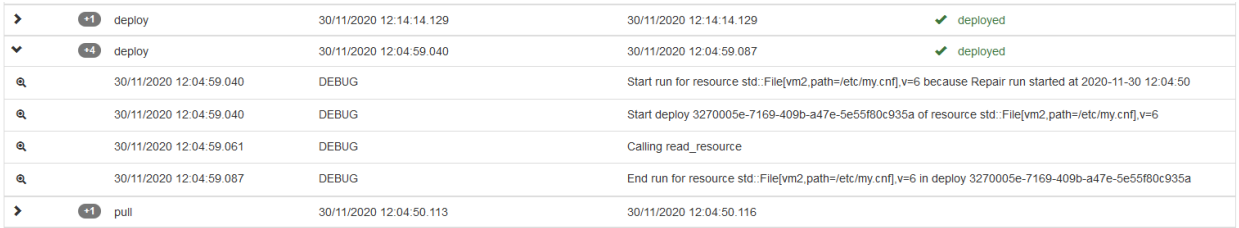


Fig. 10: Resource action view

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.

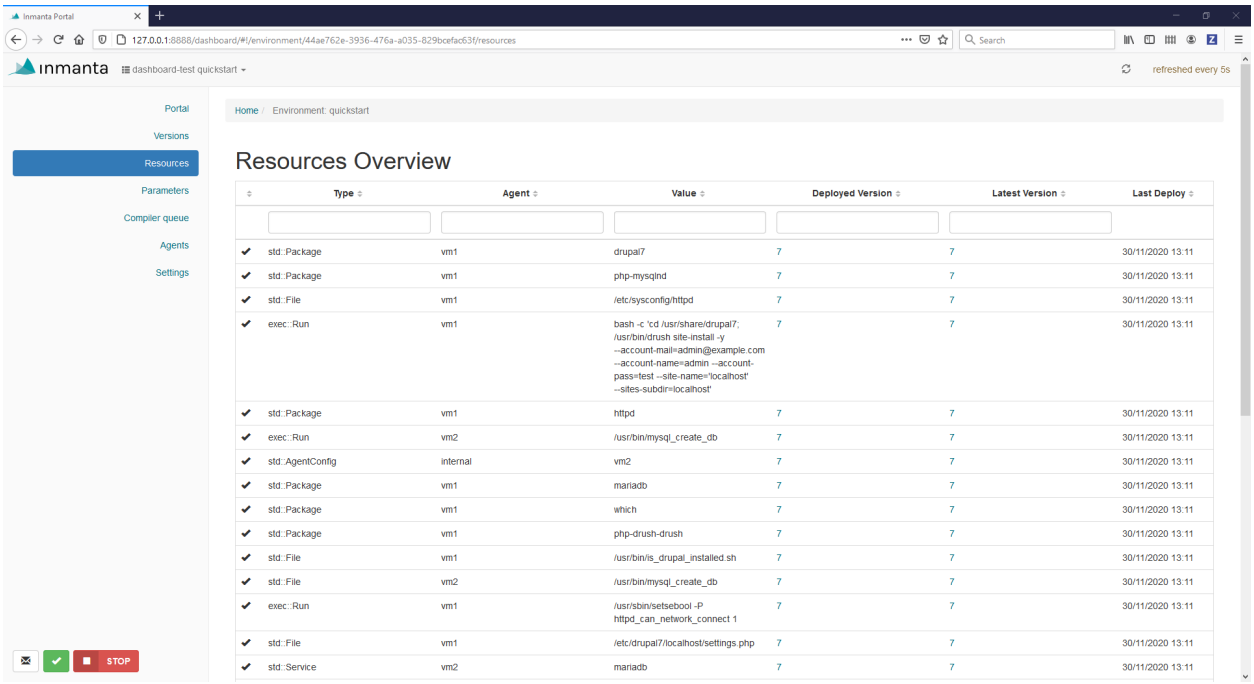


Fig. 11: The Resources Overview

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.

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:

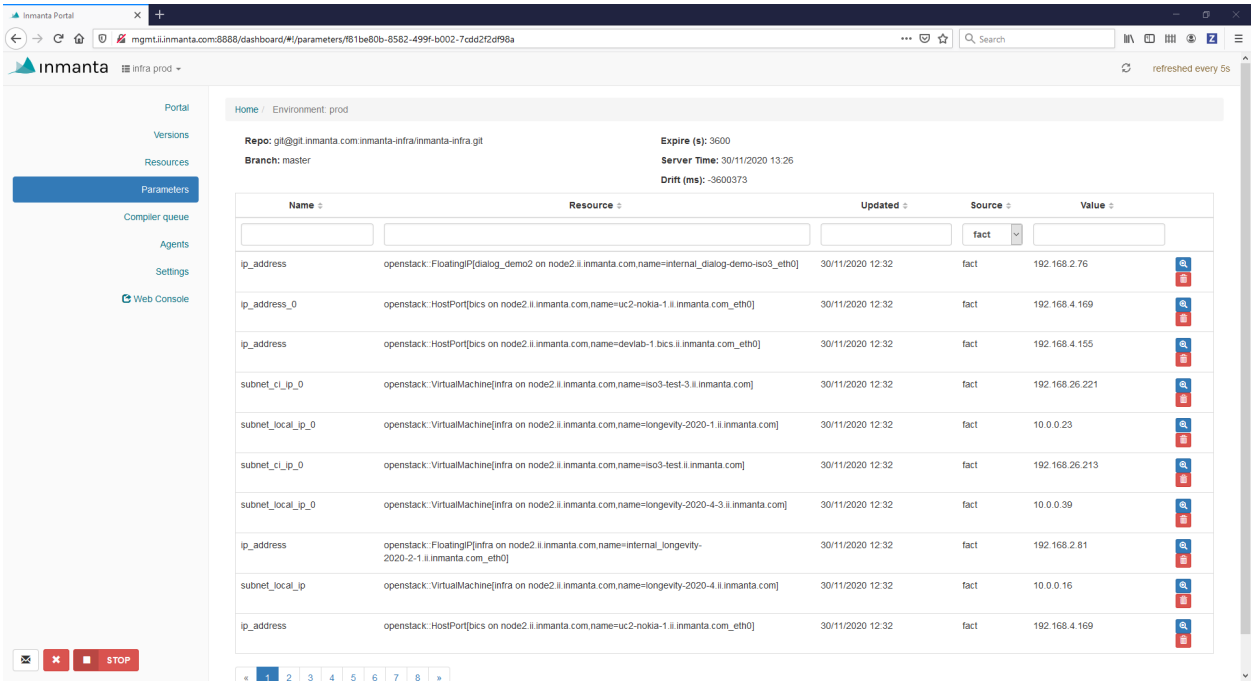


Fig. 12: The Parameter overview

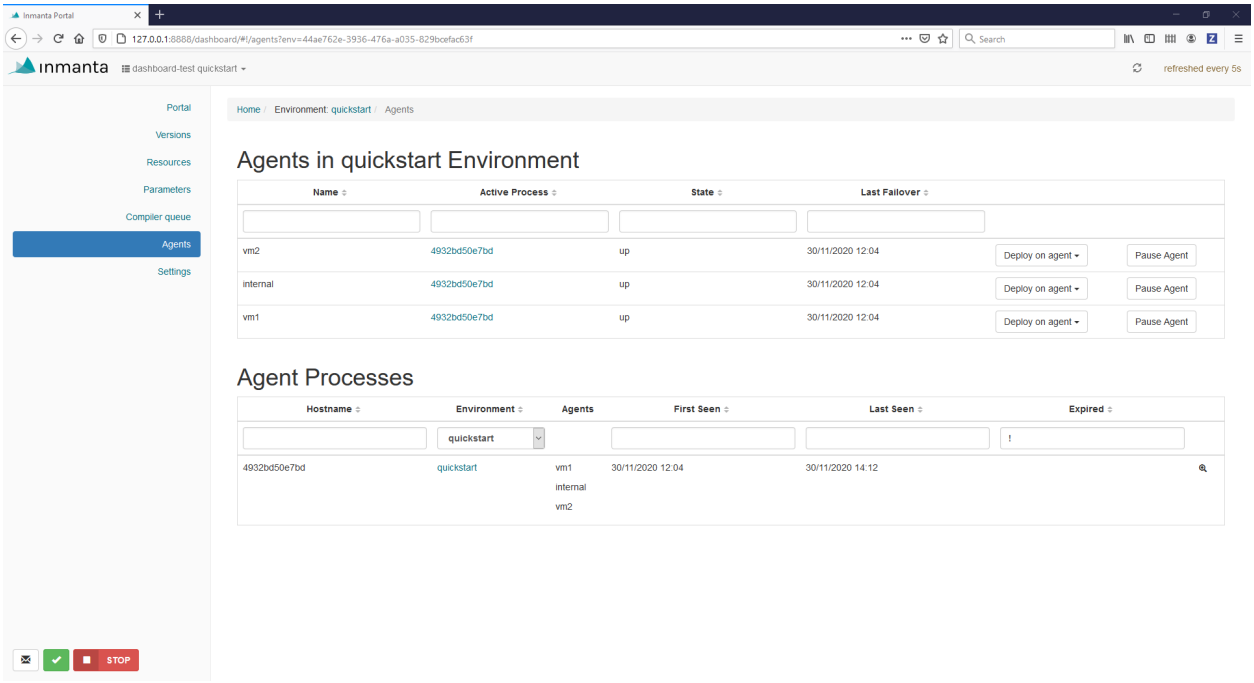


Fig. 13: The Agent overview

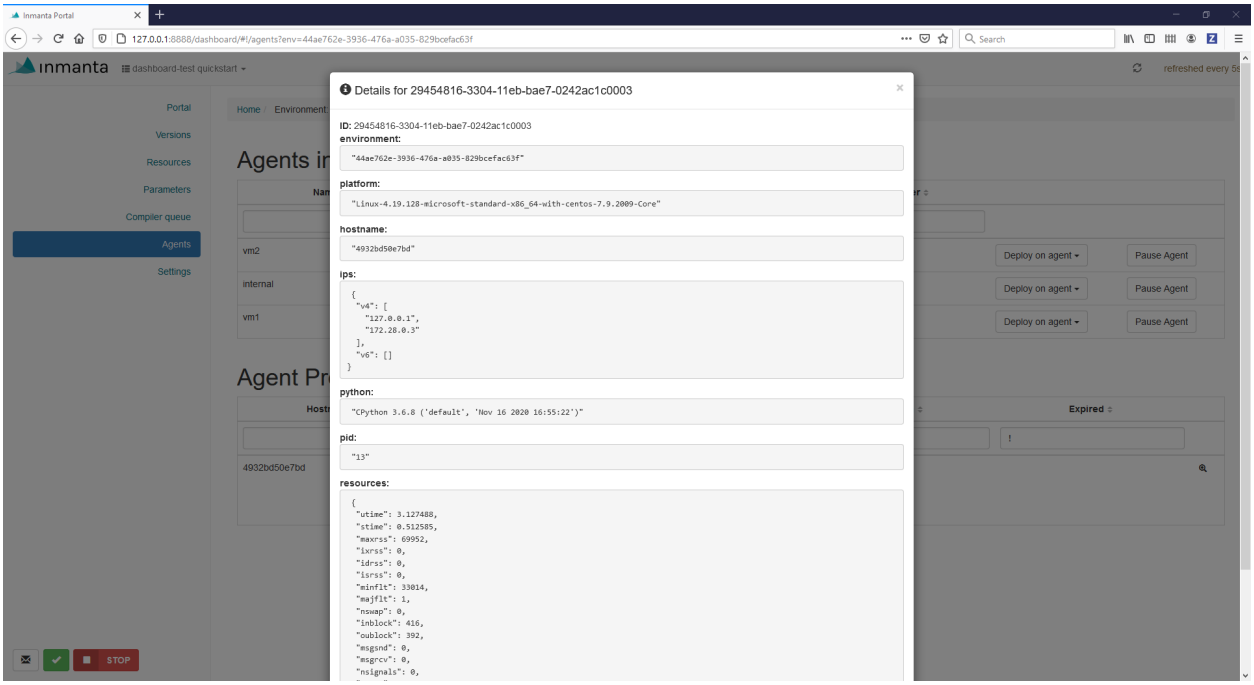


Fig. 14: Agent process inspection

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.

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.

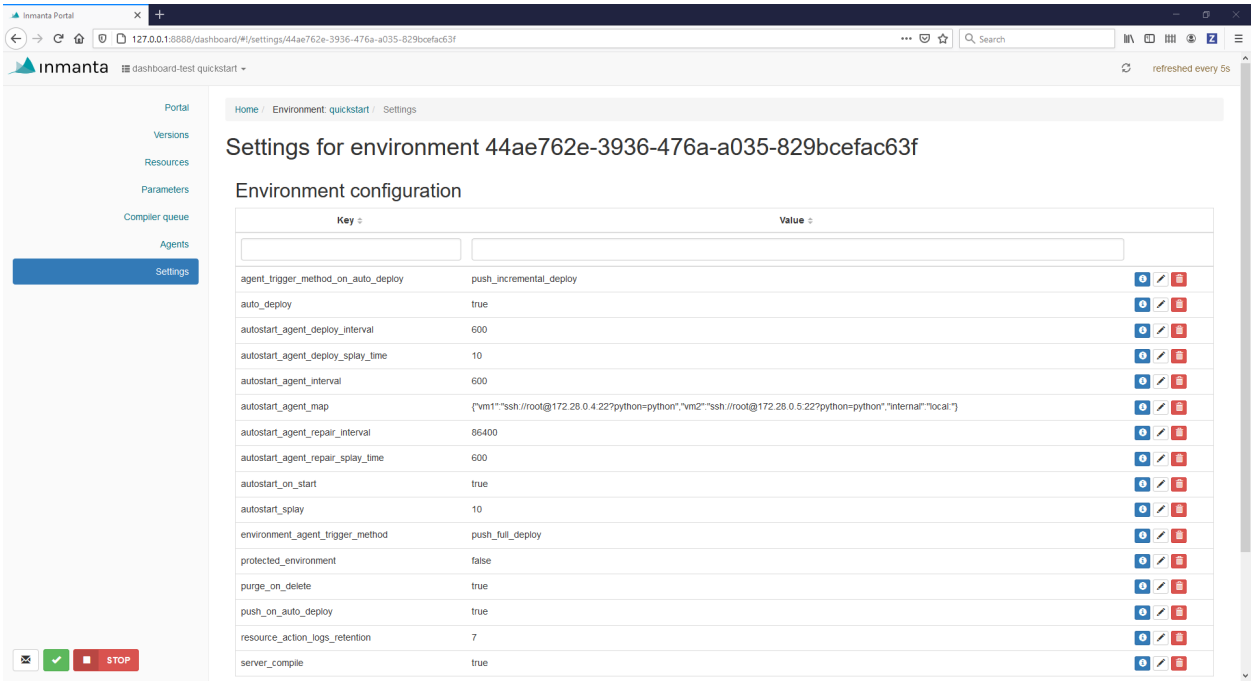


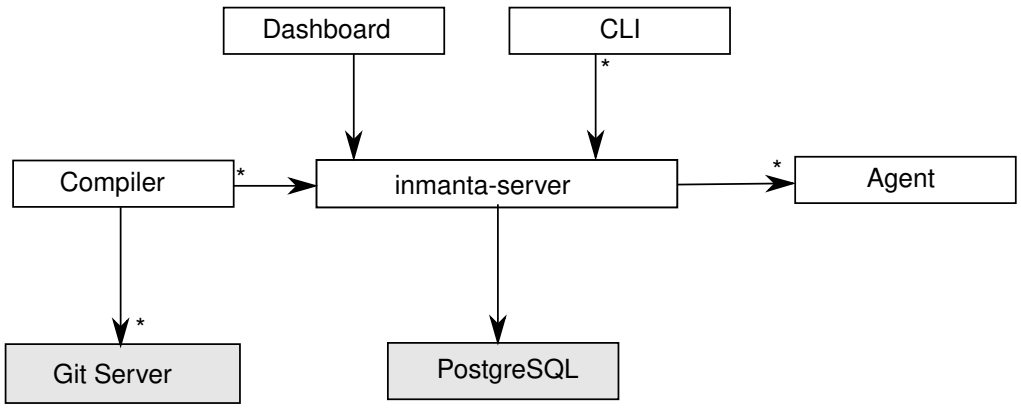
Fig. 15: The Environment settings

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 Dashboard: To control the server, you can use either the web dashboard 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.

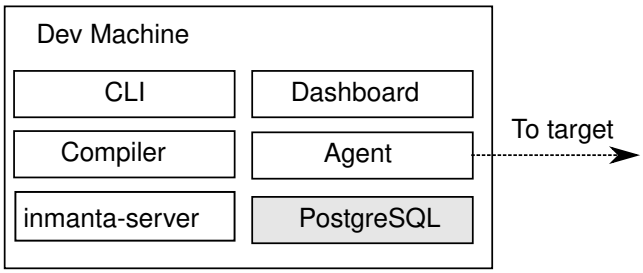
4.1 Usage modes

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



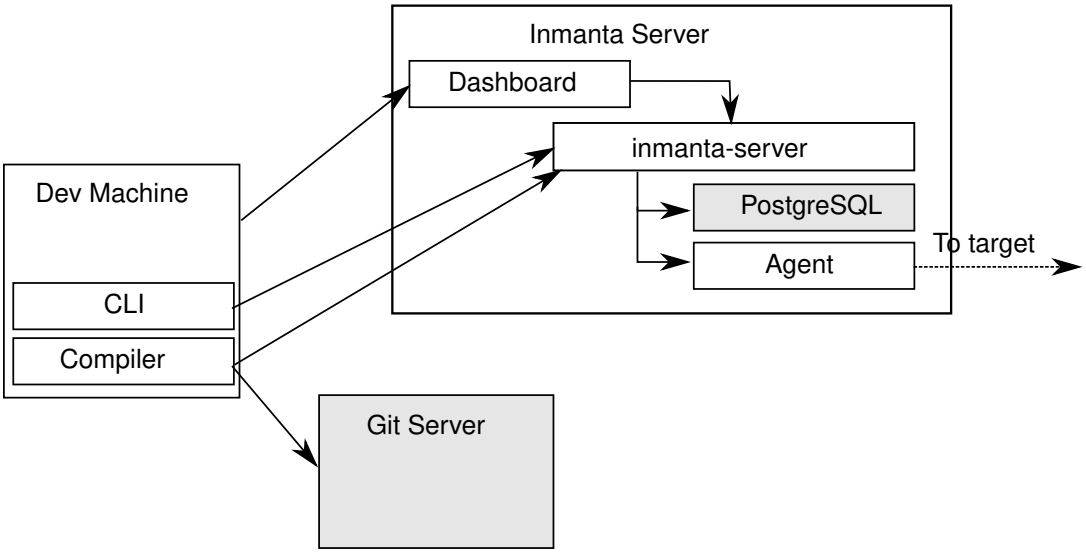
In a all in one deployment, all components (server, agent and postgres) are started embedded in the compiler and terminated after the deploy is complete. No specific setup is required. To deploy the current model, use:

```
inmanta deploy
```

The *–dashboard* option disable CLI reporting and sets up the dashboard. The embedded server is setup in such a way that the current project is also availble for server compilation. After the first deploy finshes, the command keeps running for additional deploys until ctrl+c is used to terminate the command.

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:

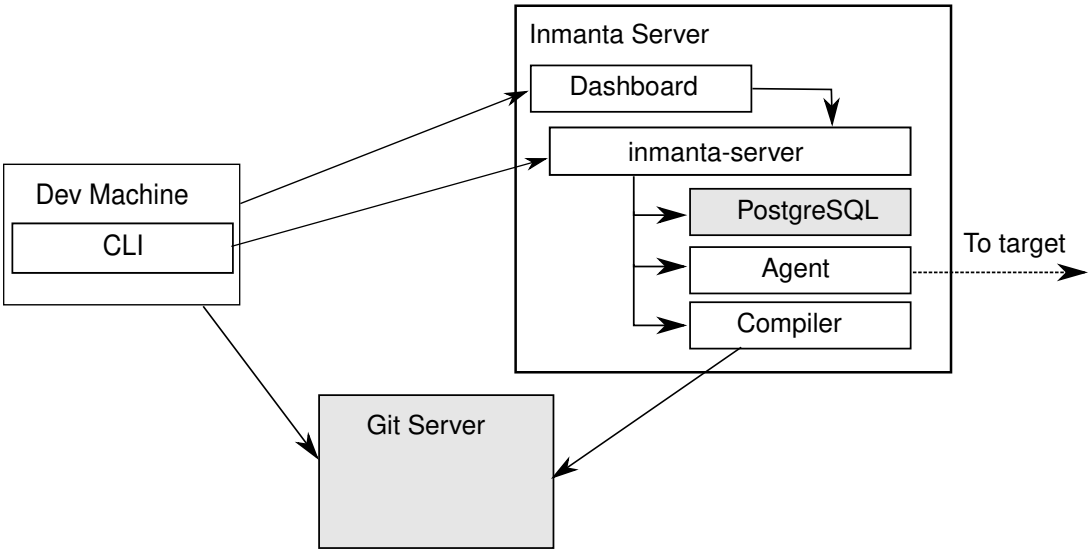
```
[config]
environment=$ENV_ID

[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
```

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

# var 4 and 5 are both booleans
var4 = true
var5 = false

# var6 is a list of values
var6 = ["fedora", "ubuntu", "rhel"]

# a dictionary with string keys and any type of values is also a primitive
var7 = { "foo":"bar", "baz": 1}

# var8 contains the same value as var2
var8 = var2

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

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

#ref to a variable from another namespace
```

(continues on next page)

(continued from previous page)

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

5.4 Primitive types

The basic primitive types are `string`, `number`, `int` or `bool`.

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;

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 = true
else:
    self.cluster_mode = false
end
```

The syntax is:

```
if : 'if' 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"
```

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 module-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 = ""
ssh_public_key=""
```

(continues on next page)

(continued from previous page)

```
14
15 # change OS parameter to match the actual image. If an OS is not modelled in an␣
   ↪existing module,
16 # std::linux can be used for example. However, other modules might not have support␣
   ↪for a
17 # generic os definition such as std::linux
18 os = redhat::fedora23
19 ## End edit
20
21 # register ssh key
22 ssh_key = ssh::Key(name="mykey", public_key=ssh_public_key)
23
24 # Define the OpenStack provider to use
25 provider = openstack::Provider(name="iaas_openstack", connection_url=std::get_env("OS_
   ↪AUTH_URL"),
26                                username=std::get_env("OS_USERNAME"),
27                                password=std::get_env("OS_PASSWORD"),
28                                tenant=std::get_env("OS_PROJECT_NAME"))
29
30 # Define the project/tenant to boot the VM in, but do not let inmanta manage it
31 project = openstack::Project(provider=provider, name=provider.tenant, description="",␣
   ↪enabled=true,
32                              managed=false)
33
34 # Define the network objects to connect the virtual machine to but again, do not␣
   ↪manage them
35 net = openstack::Network(provider=provider, project=project, name=network_name,␣
   ↪managed=false)
36 subnet = openstack::Subnet(provider=provider, project=project, network=net, dhcp=true,
   ↪ managed=false,
37                            name=subnet_name, network_address=network_address)
38
39 # Define the virtual machine
40 vm = openstack::Host(provider=provider, project=project, key_pair=ssh_key, name=
   ↪"testhost",
41                      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]
name=Copr repo for inmanta owned by bartvanbrabant
baseurl=https://copr-be.cloud.fedoraproject.org/results/bartvanbrabant/inmanta/fedora-
↪\$releasever-\$basearch/
type=rpm-md
```

(continues on next page)

(continued from previous page)

```
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:

```
1 import mymodule
2 import ssh
3 import redhat
4 import ubuntu
5
6 ## Edit this parameters
7 image_id = ""
```

(continues on next page)

(continued from previous page)

```
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

#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:
```

(continues on next page)

(continued from previous page)

```
    # 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")
    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 Project creation guide

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

7.1.1 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:
    - https://github.com/inmanta/
install_mode: release
requires:
```

7.1.2 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.

This example can be executed with `inmanta compile`

This prints out "hello world" on stdout. The first execution takes longer because Inmanta needs to fetch (clone) the std module from github. Subsequently compiles will use the std module downloaded to the libs directory.

7.2 Module Developers Guide

In inmanta all orchestration model code and related files, templates, plugins and resource handlers are packaged in a module.

7.2.1 Module layout

Inmanta expects that each module is a git repository with 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 `plugins` 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.

A complete module might contain the following files:

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

(continues on next page)

(continued from previous page)

```
|
|__ templates
     |__ conf_file.conf.tmpl
```

To quickly initialize a module use cookiecutter:

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

7.2.2 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
```

Module dependencies are indicated by importing a module in a model file. However, these imports do not have a specific version identifier. The version of a module import can be constrained in the module.yml file. The *requires* key expects a list of version specs. These version specs use PEP440 syntax.

To specify specific version are required, constraints can be added to the requires list:

```
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
```

A module can also indicate a minimal compiler version with the `compiler_version` key.

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

To automatically freeze all versions to the currently checked out versions

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

Or for the the current project

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

For more information about the `module.yml` file see *module.yml*.

7.2.3 Versioning

Inmanta modules are versioned based on git tags. The current version is reflected in the `module.yml` file and in the commit is should be tagged 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.

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 -m "First commit"
```

For the dev releases, no tags are created by default. If a tag is required for a dev release, use the –tag option.:

```
inmanta module commit -m "First commit" --tag
```

To make an actual release. It will automatically set the right tags on the module:

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

If a release shouldn't be tagged, the –no-tag option should be specified:

```
inmanta module commit -r -m "First Release" --no-tag
```

To set a specific version:

```
inmanta module commit -r -m "First Release" -v 1.0.1
```

The module tool also support semantic versioning instead of setting versions directly. Use one of `--major`, `--minor` or `--patch` to update version numbers: <major>.<minor>.<patch>

7.2.4 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 a requirements.txt file with all 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 folllowing:

- 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.

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.

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

```

(continues on next page)

(continued from previous page)

```
@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, include a requirements.txt in the module. The libraries listed in this file are automatically installed by the compiler and agents.

South Bound Integration

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: *Module Developers Guide*).
2. Model the target platform as set of *entities*.
3. Create *resources* and *handler*, as described below.

A *resource* defines how to serialize an *entity* so that it can be sent over to the server and the agent. A *handler* is the python code required by the agent to enforce the *desired state* expressed by a resource.

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.

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 *ResourceHandler*. The *@provider* decorator registers the class with the orchestrator. When the agent needs a handler for a resource it will load all handler classes registered for that resource and call the *available()* method. This method should check if all conditions are fulfilled to use this handler. The agent will select a handler, only when a single handler is available, so the *available()* method of all handlers of a resource need to be mutually exclusive. If no handler is available, the resource will be marked unavailable.

ResourceHandler is the handler base class. *CRUDHandler* provides a more recent base class that is better suited for resources that are manipulated with Create, Delete or Update operations. These operations often match managed APIs very well. The CRUDHandler is recommended for new handlers unless the resource has special resource states that do not match CRUD operations.

Each handler basically needs to support two things: reading the current state and changing the state of the resource to the desired state in the orchestration model. Reading the state is used for dry runs and reporting. The CRUDHandler handler also uses the result to determine whether create, delete or update needs to be invoked.

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

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 dashboard 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
```

7.3 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.

7.3.1 Install the pytest-inmanta package

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

```
pip install pytest-inmanta
```

7.3.2 Writing a test case

Take the following plugin as an example:

```
1  # <module-name>/plugins/__init__.py
2
3  from inmanta.plugins import plugin
4
5  @plugin
6  def hostname(fqdn: "string") -> "string":
7      """
8          Return the hostname part of the fqdn
9      """
10     return fqdn.split(".")[0]
```

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

```
1  # <module-name>/tests/test_hostname.py
2
3  def test_hostname(project):
4      host = "test"
5      fqdn = f"{host}.something.com"
6      assert project.get_plugin_function("hostname")(fqdn) == host
```

- **Line 3:** Creates a pytest test case, which requires the `project` fixture.
- **Line 6:** Calls the function `project.get_plugin_function(plugin_name: str): FunctionType`, which returns the plugin function named `plugin_name`. As such, this line tests whether `host` is returned when the plugin function `hostname` is called with the parameter `fqdn`.

For more information see: pytest-inmanta

7.4 Environment variables

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

7.4.1 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")
```

7.4.2 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.

7.5 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.
- Benefit from linting and code navigation by setting up a project.
- Set project sources
- Module developers guide
- Required environment variables

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

7.5.1 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.

7.5.2 Setting up Python virtual environments

For every project that you work on, we recommend using a new virtual environment using ``venv``s. If you need a refresher, 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 (due to changes in the pip resolver in version 20 and 21).

You can do so by running:

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

7.5.3 Benefit from linting and code navigation by 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 Working on a New Project.
2. Working on an existing project Working on an Existing Project.

Working on a New Project

To create a new project:

```
pip install cookiecutter

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

For more details go here.

You need to install some essential packages as follows:

```
pip install inmanta-core

pip install pytest

pip install pytest-inmanta
```

Once you are done with creating a project and installing the required modules, you can `cd` into that directory and open vs code by running:

```
cd <project_name>

code .
```

Upon opening your vs code, and the `main.cf` file, you should see modules downloading in `libs` directory.

Working on an Existing Project

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

```
git clone project_name
```

They also come with `requirements.txt` or `requirements.dev.txt` to install the required modules:

```
pip install -r requirements.txt

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

7.5.4 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. Otherwise, you can skip this step and just `import` your desired modules.

If you only use opensource 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:
    - git@code.inmanta.com:example/my_module.git
```

Becomes:

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

7.5.5 Module developers guide

Like projects, there are also two scenarios:

1. Working on a new module Working on a New Module.
2. Working on an existing module 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 https://github.com/inmanta/inmanta-module-template.git
```

There are also guides here and here that help you get up and running.

Working on an Existing Module

Modules that you want to work on, have to be `import``ed in the ``main.cf` file that is located in your main project directory. For instance:

:: import vyos

To download the `import``ed modules in your ``main.cf` file run:

```
inmanta compile
```

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. There is also a guide here that is useful in case you skipped the previous part.

7.5.6 Required Environment Variables

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. It can be set to something like:

1. Create the required TEST directories:

```
mkdir -p /tmp/env
```

2. Export below entries based on your setup:

```
export INMANTA_TEST_ENV="/tmp/env"
```

7.6 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.

7.6.1 Enabling the data trace

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

Listing 1: main.cf

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

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

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

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

7.6.2 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

```
1 x = 0
2 x = y
3 y = 1
```

Listing 3: data trace for variable-assignment.cf

```
1  x
2  ├── y
3  │   SET BY `x = y`
4  │   AT ./variable-assignment.cf:2
5  │   └── 1
6  │       SET BY `y = 1`
7  │       AT ./variable-assignment.cf:3
8  └── 0
9      SET BY `x = 0`
10     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 casused the equivalence. The rest of the trace is similar to before, except that the tree now shows all assignments to any of the three variables part of the equivalence. The tree now no longer shows just the data flow to `x` but to the equivalence as a whole, since any data that flows to the equivalence will also flow to `x`.

Listing 6: entities.cf

```
entity A:
    number n
end

implement A using std::none

x = A(n = 0)

template = x

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

Listing 7: data trace for entities.cf

```
attribute n on __config__::A instance
SUBTREE for __config__::A instance:
    CONSTRUCTED BY `A(n=template.n)`
    AT ./entities.cf:11
```

(continues on next page)

(continued from previous page)

```
├── template.n
    SET BY `A(n=template.n)`
    AT ./entities.cf:11
    SUBTREE for template:
        └── x
            SET BY `template = x`
            AT ./entities.cf:9
            └── __config__::A instance
                SET BY `x = A(n=0)`
                AT ./entities.cf:7
                CONSTRUCTED BY `A(n=0)`
                AT ./entities.cf:7
    └── 0
        SET BY `A(n=0)`
        AT ./entities.cf:7
└── 1
    SET BY `y.n = 1`
    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
```

(continues on next page)

(continued from previous page)

```
11      IN IMPLEMENTATION WITH self = __config__::A instance
12          CONSTRUCTED BY `A(n=0)`
13          AT ./implementation.cf:11
```

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

And finally, an index:

Listing 10: index.cf

```
1  entity A:
2      number n
3      number m
4  end
5
6  index A(n)
7
8  implement A using std::none
9
10 A(n = 42, m = 0)
11 A(n = 42, m = 1)
```

Listing 11: data trace for index.cf

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

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

7.6.3 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.

```
1  entity A:
2      number n
3  end
4
5  implement A using std::none
6
7  x = A()
8  y = A()
```

(continues on next page)

(continued from previous page)

```
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.

7.6.4 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
```

(continues on next page)

(continued from previous page)

```

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


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


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

Compiling this with data trace disabled outputs the following error:

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

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

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

Listing 16: data trace for service.cf

```
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
```

(continues on next page)

(continued from previous page)

```
13              CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',
↪portn=22)`
14              AT ./service.cf:33
15  ├── __config__::Service instance
16  │   SET BY `self.port = Port(host=self.host,portn=self.portn)`
17  │   AT ./service.cf:22
18  │   IN IMPLEMENTATION WITH self = __config__::Service instance
19  │       CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
20  │       AT ./service.cf:33
21  │   CONSTRUCTED BY `Service(name='some_custom_service',host='my_host',portn=22)`
22  │   AT ./service.cf:33
23  └── __config__::Service instance
24      SET BY `self.port = Port(host=self.host,portn=self.portn)`
25      AT ./service.cf:22
26      IN IMPLEMENTATION WITH self = __config__::Service instance
27          CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
28          AT ./service.cf:26
29      CONSTRUCTED BY `Service(name='opensshd',host='my_host',portn=22)`
30      AT ./service.cf:26
```

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

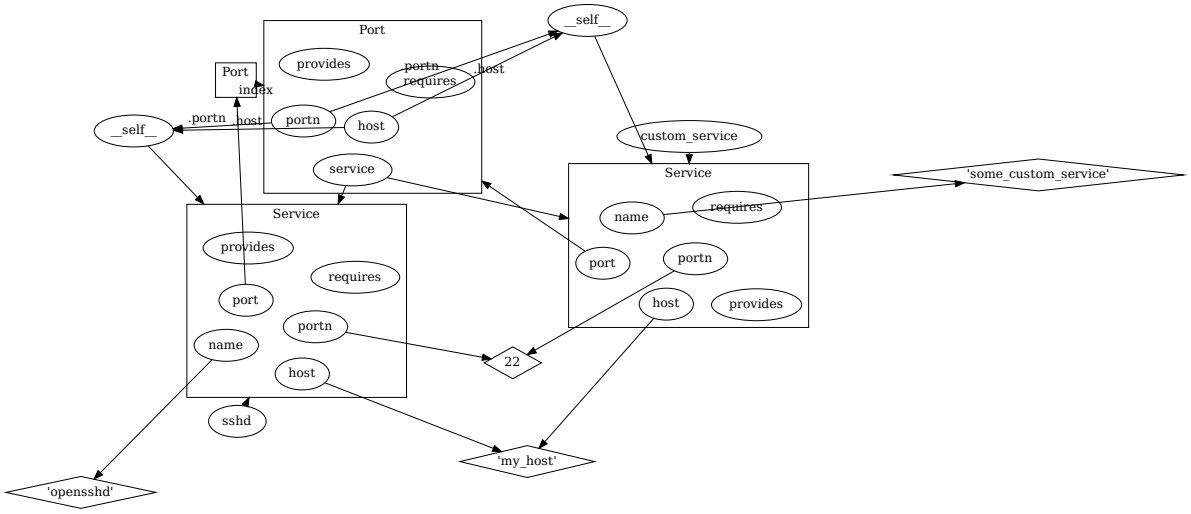
Lines 2–14 in the trace give some additional information about the `Port` instance. It indicates there is an index match between the `Port` instances constructed in the implementations for both `Service` instances. This illustrates the existence of the two branches at lines 15 and 23, and why the assignment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

7.6.5 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.

7.7 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.

7.7.1 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.

7.7.2 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.

7.7.3 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.

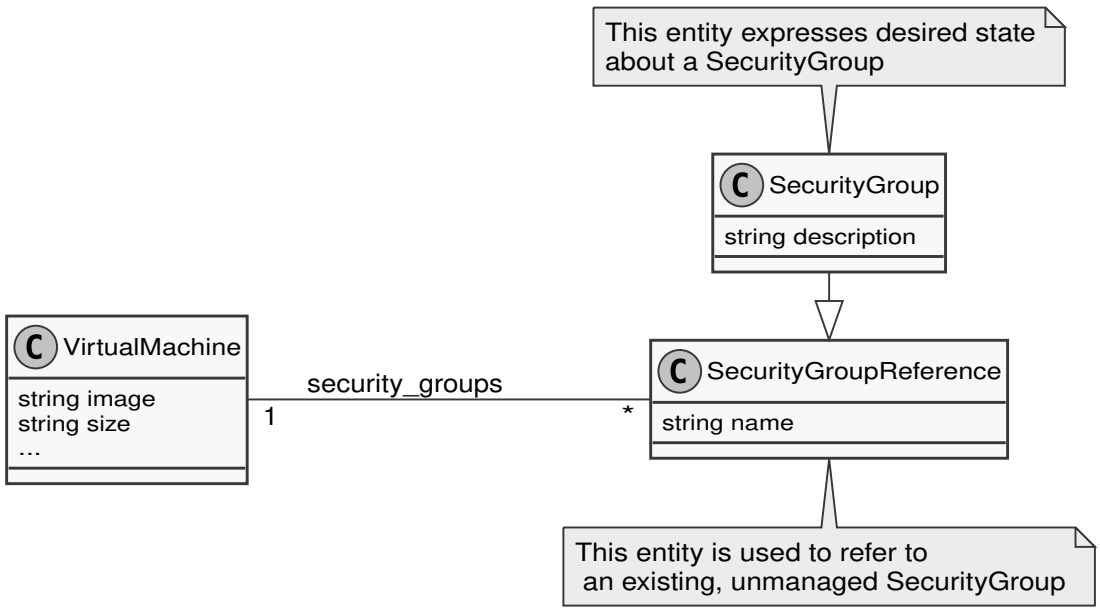
Textual references have the advantage that it is easy to refer to things that are not in the model.

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.

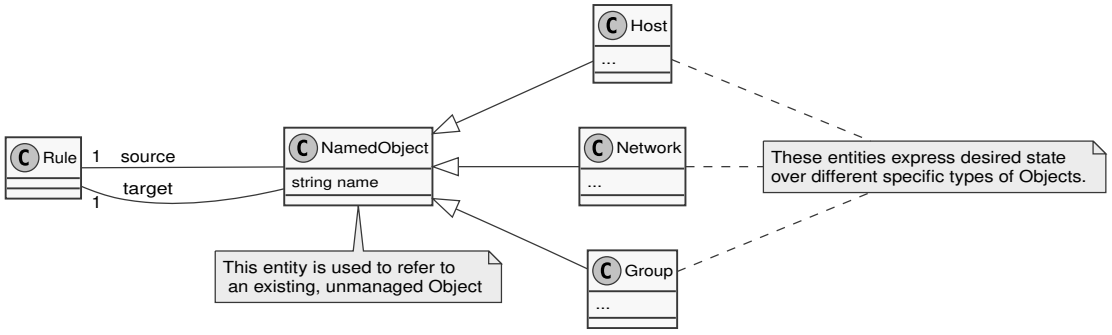
One solution to allow reference to unmanaged entities is to extend *std::ManagedResource*. This allows an entity to exist in the model, but when `managed` is set to `false`, it will never become a resource. However, the entity must still be valid. All attributes and relations still have to be filled 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.



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

This page describes how database schema updates are managed by the Inmanta core.

8.2.1 Definition new schema version

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<version>.py`, where <version> is an integer indicating the version of the new database schema. Version numbers start at 1.

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 (<version> - 1) to the new version of the database schema (<version>). **All changes done by the update function should be executed in the transaction.** An example is given in the code snippet below.

Each each of these Python modules must also contain the field `DISABLED` set to false to make the changes effective.

```
# File: src/inmanta/db/versions/v1.py
from asyncpg import Connection

DISABLED = False

async def update(connection: Connection) -> None:
    schema = """
    ALTER TABLE public.test
    ADD COLUMN new_column;
    """
    async with connection.transaction():
        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 latest schema version for which the `update` function did succeed. In that case the Inmanta server will fail to start.

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, dashboard and 3rd party service). The id parameter in the path will be passed to the associate API handle. A docstring can be associated with the API method. This information will be included in the OpenAPI documentation, available via the `/docs` endpoint of the Inmanta server.

A complete list of all the arguments accepted by the `method` decorator is given below.

`decorators.`**`method`**(*operation: str = 'POST', reply: bool = True, arg_options: Dict[str, inmanta.protocol.common.ArgOption] = {}, timeout: Optional[int] = None, server_agent: bool = False, api: Optional[bool] = None, agent_server: bool = False, validate_sid: Optional[bool] = None, client_types: 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:
```

(continues on next page)

(continued from previous page)

```
    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, enum.Enum, uuid.UUID, inmanta.types.StrictNonIntBool, float, datetime.datetime, str, Sequence[Union[BaseModel, enum.Enum, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]], Mapping[str, Union[BaseModel, enum.Enum, 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

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 tho**
   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()

Convert to serialized form:

```
{
    "type": self.type,
    "multi": self.multi,
    "nullable": self.nullable,
    "comment": self.comment,
    "location": self.location.to_dict()
}
```

class inmanta.model.**DirectValue**(*value*)

A primitive value, directly represented in the serialized form.

Parameters **value** – the value itself, as string or number

to_dict()

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()

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()

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()

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
- **int] multi** (*Tuple[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()

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 Setting up authentication

This guide explains how to enable ssl and setup authentication.

9.1.1 SSL

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.

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 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.

9.1.2 Authentication

Inmanta authentication uses JSON Web Tokens for authentication (bearer token). Inmanta issues tokens for service to service interaction (agent to server, compiler to server, cli to server and 3rd party API interactions). For user interaction through the dashboard Inmanta uses 3rd party auth brokers. Currently the dashboard 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, dashboard, 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 under Settings of the environment in the dashboard.

Configure an external issuer (See *External authentication providers*) for dashboard access to bootstrap access to the create token api call. When no external issuer is available and dashboard 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 valided 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.

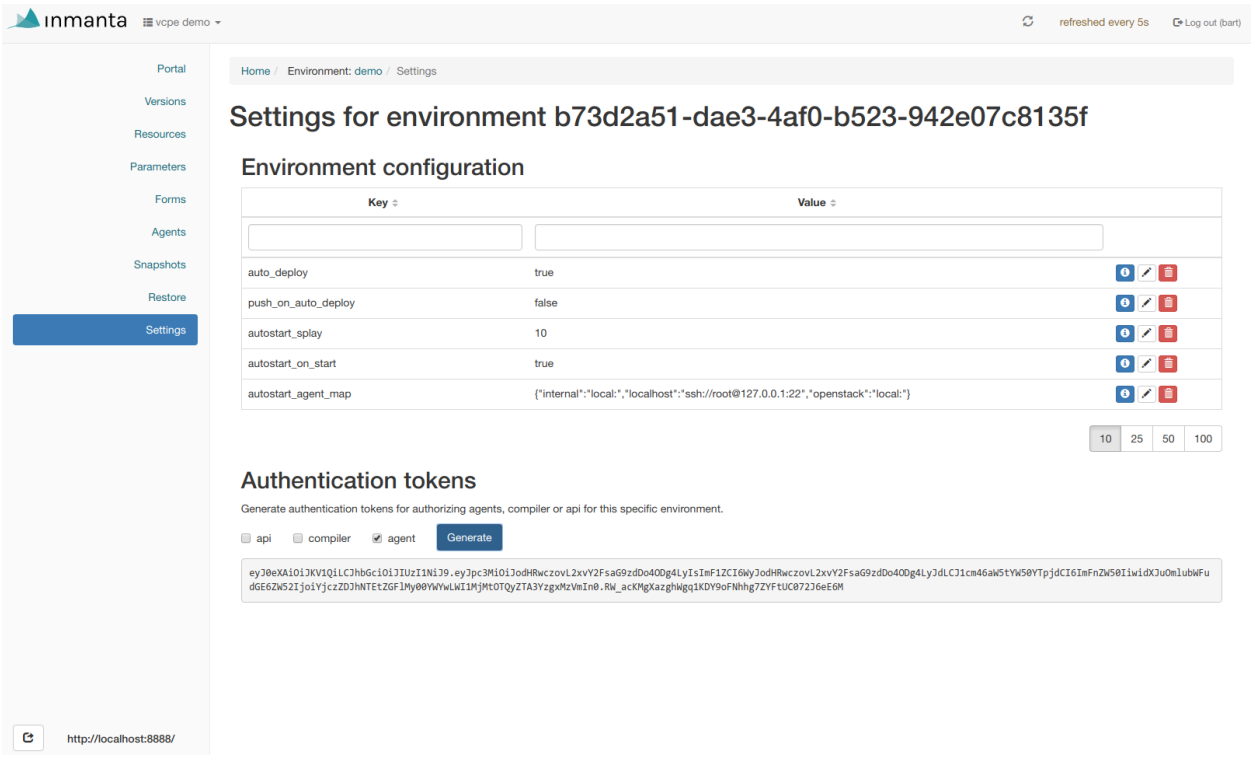


Fig. 1: Generating a new token in the dashboard.

- 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.

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('='));"
```

9.1.3 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 dashboard only has support for keycloak. However, each provider that can insert custom (private) claims should work. The dashboard 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. Alternativelyr contact Inmanta NV for custom integration services.

Keycloak configuration

The dashboard has out of the box support for authentication with Keycloak. Install keycloak and create an initial login as decribed 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 dashboard 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

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.

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

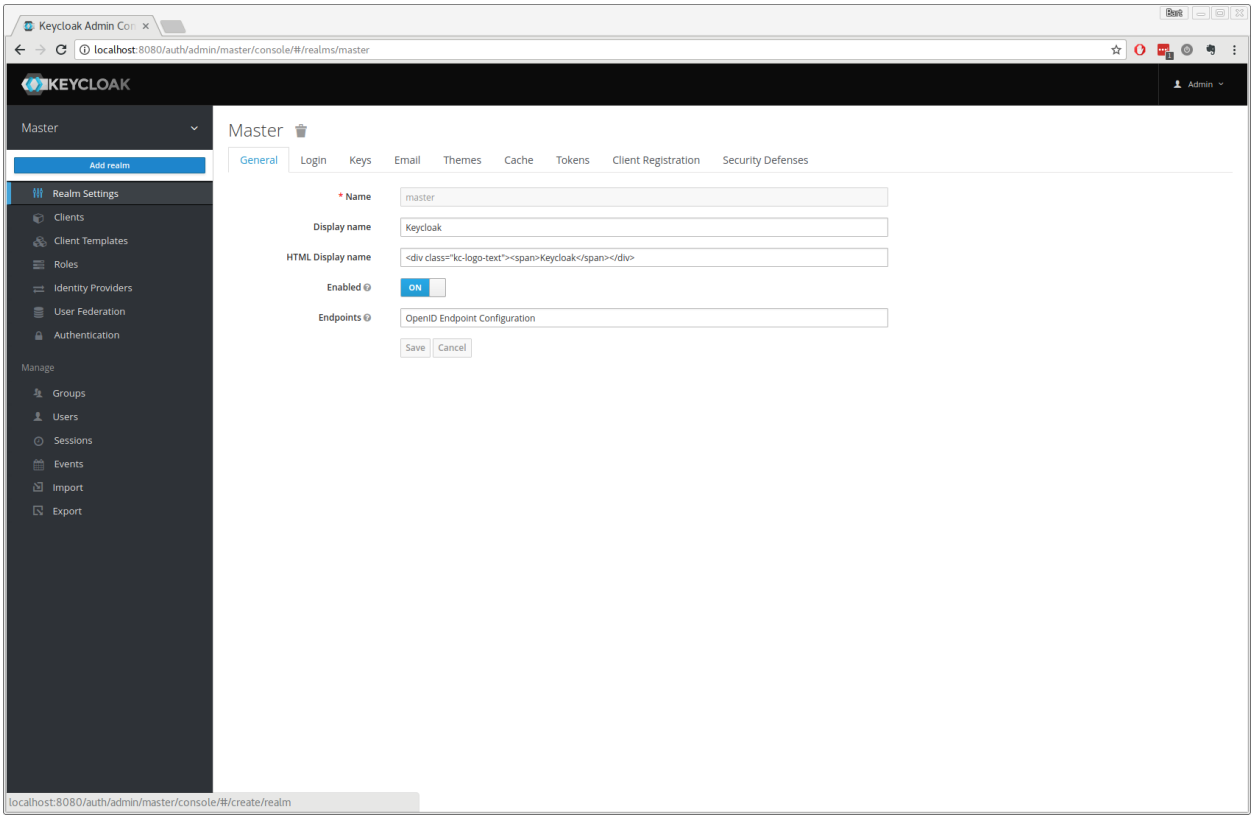


Fig. 2: Create a new realm

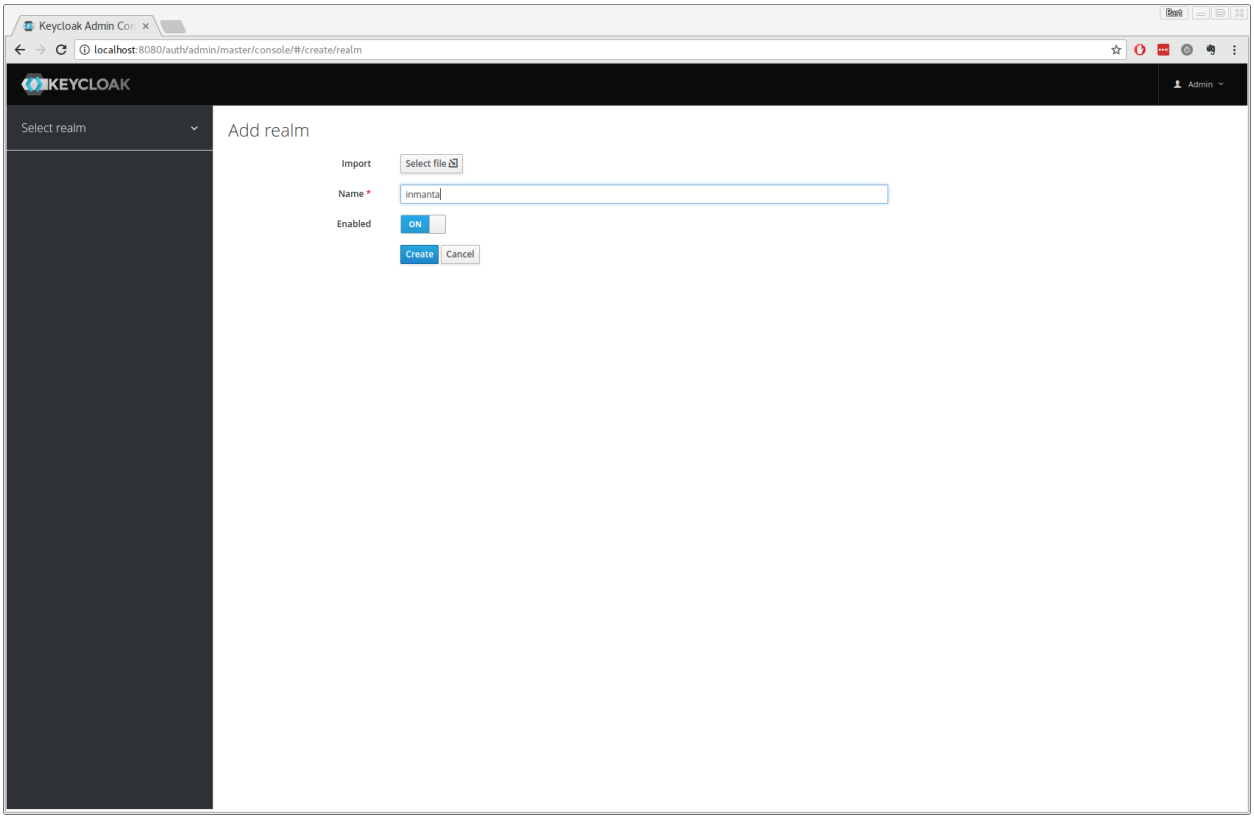


Fig. 3: Specify a name for the realm

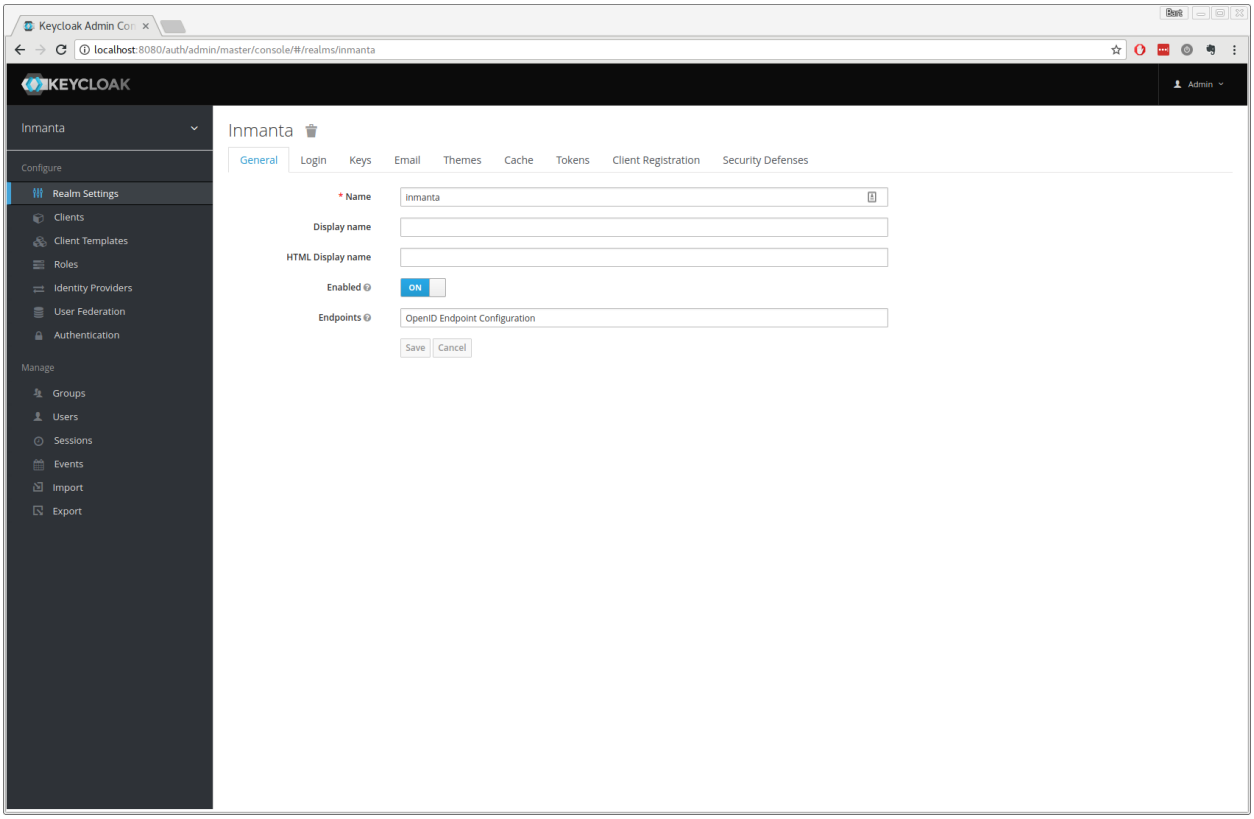


Fig. 4: The start page of a realm. Here you can edit names, policies, ... of the realm. The defaults are sufficient for inmanta authentication. This shows the inmanta realm.

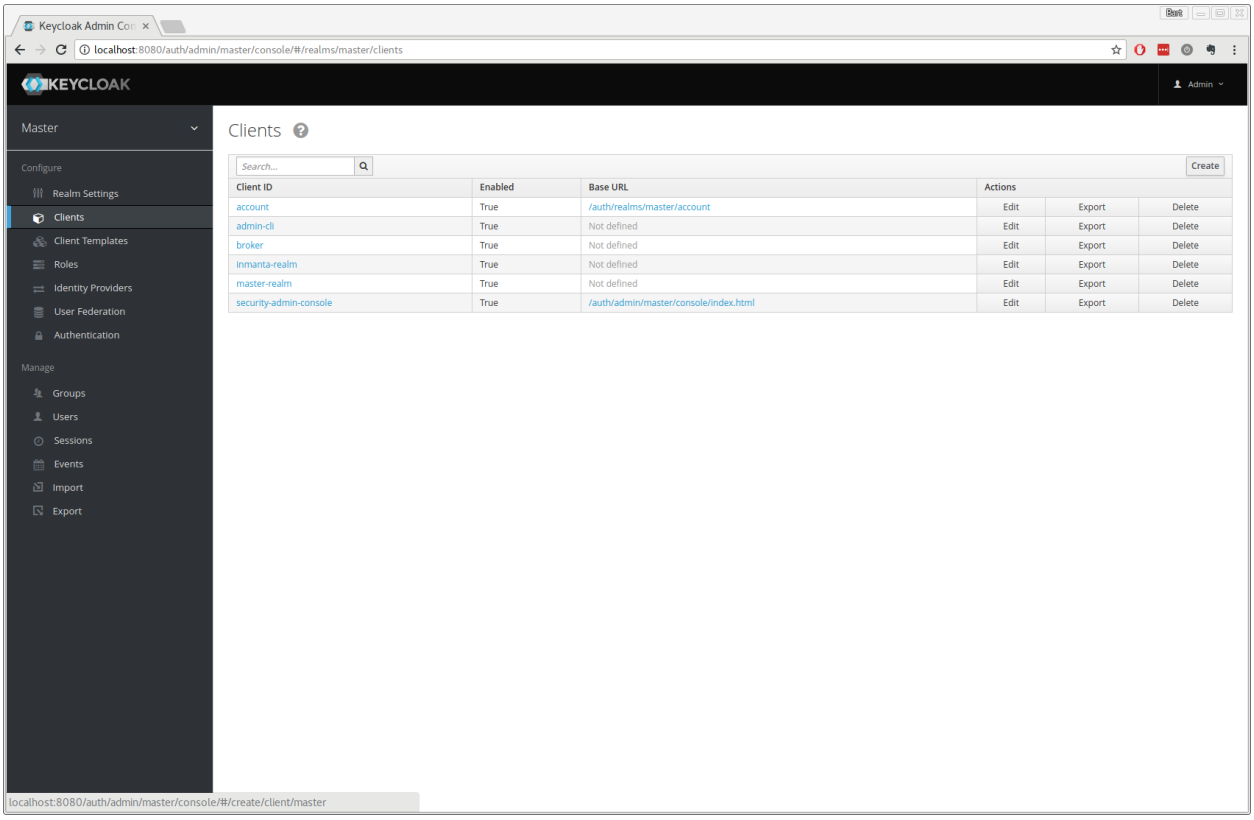


Fig. 5: Clients in the master realm. Click the create button to create an inmanta client.

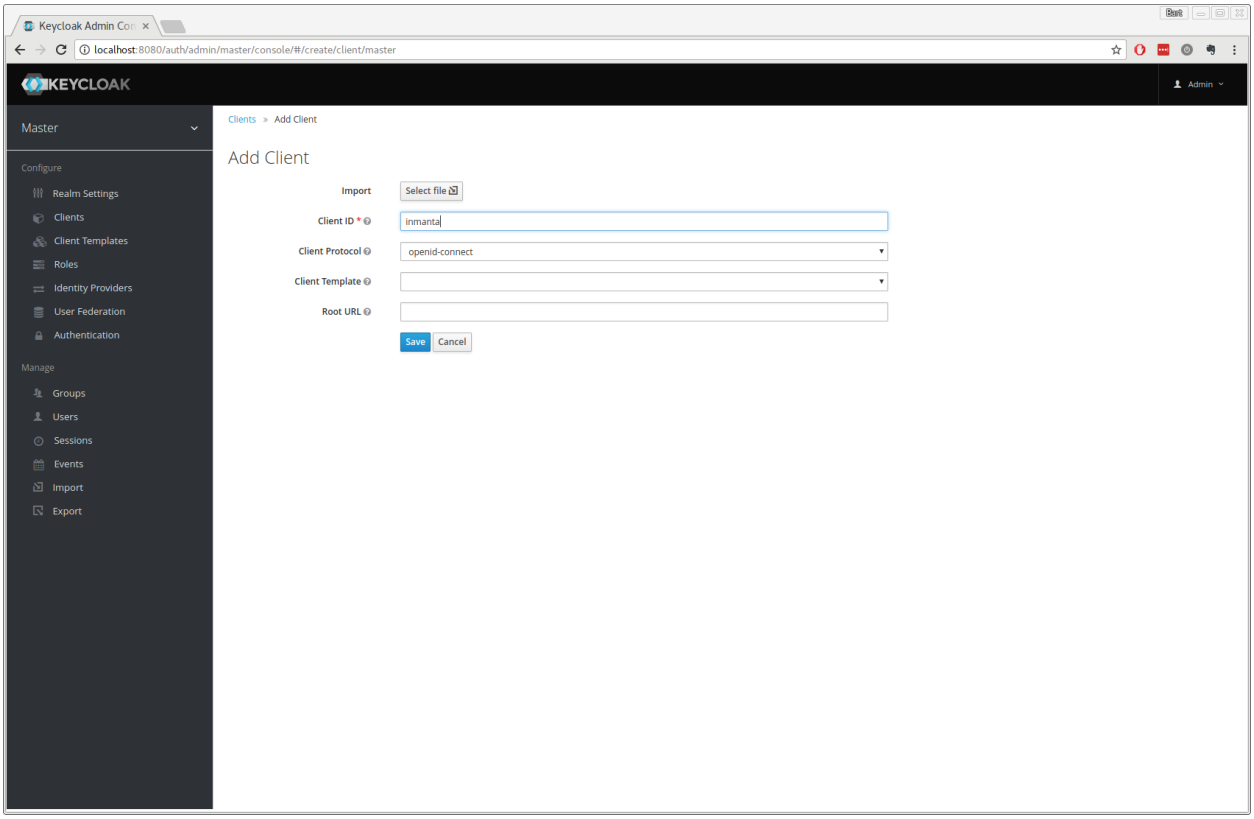


Fig. 6: Create client screen

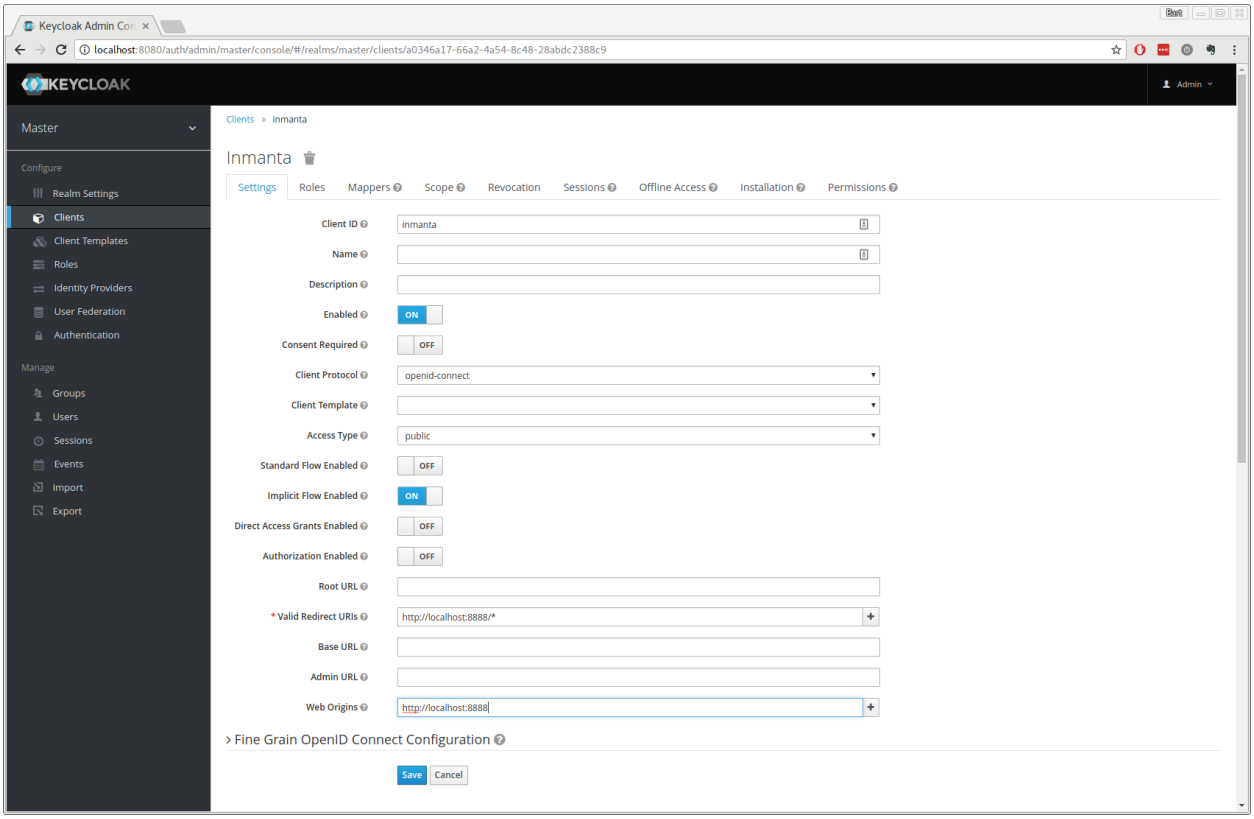


Fig. 7: Allow implicits flows (others may be disabled) and configure allowed callback urls of the dashboard.

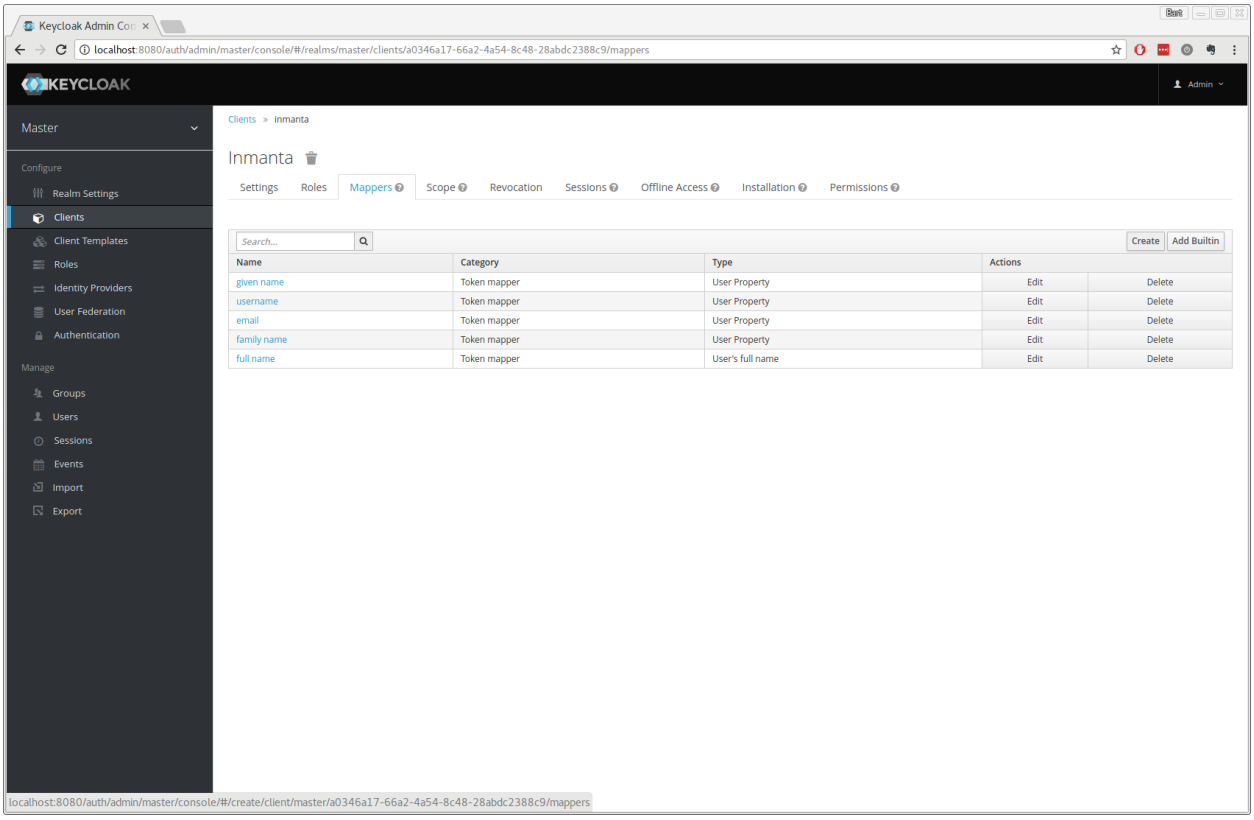


Fig. 8: Add a custom mapper to the client to include `:urn:inmanta:ct`

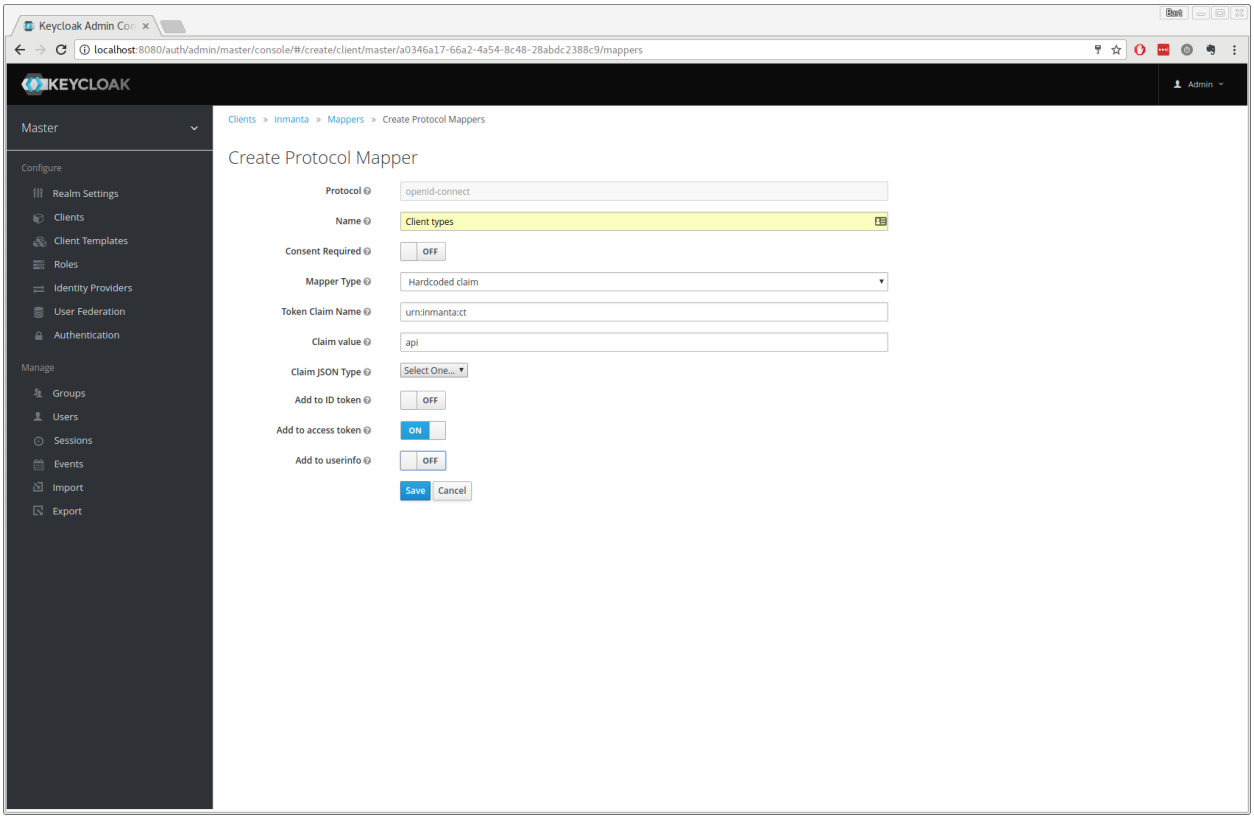


Fig. 9: Add the ct claim to all access tokens for this client.

Click save.

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 dashboard users to this keycloak instance and to valide the tokens issued by keycloak.

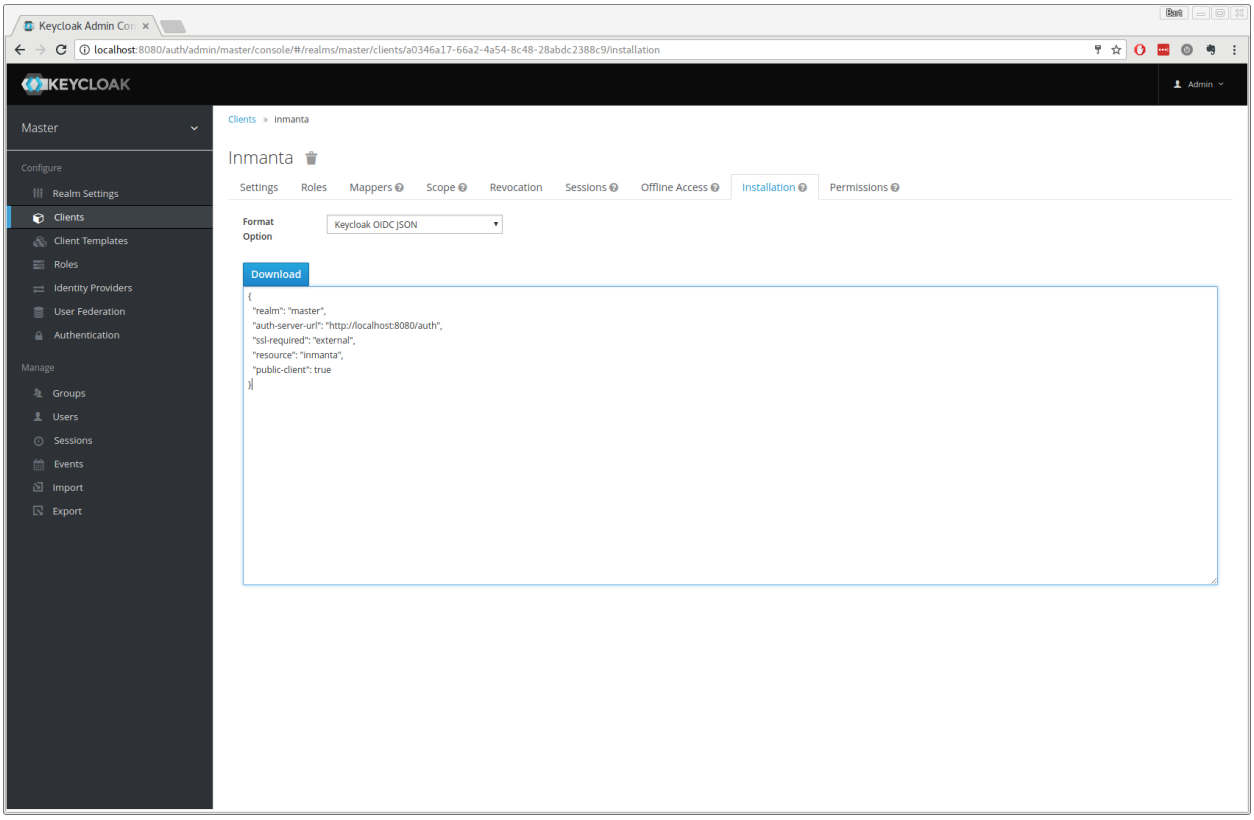


Fig. 10: Show the correct configuration parameters in JSON format.

Add a keycloak configuration parameters to the dashboard section of the server configuration file. (/etc/inmanta/inmanta.d/dashboard.cfg in most installs.) This section should already contain enabled=true and the path to the dashboard source.

Add the realm, auth_url and client_id to the dashboard section. Use the parameters from the installation json file created by keycloak.

```
[dashboard]
enabled=true
path=/opt/inmanta/dashboard

# keycloack 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 valide 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-2)

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.

9.2 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 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.

9.3.1 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 Imanta 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.

9.3.2 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
```

(continues on next page)

(continued from previous page)

```
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 dashboard. 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.

9.4 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.

9.4.1 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
```

9.4.2 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

9.4.3 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

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

ELEVEN

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

TWELVE

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.

12.1 Command Reference

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

12.1.1 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 only errors,-v warning, -vv info and -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]
                [--experimental-data-trace] [--experimental-dataflow-graphic]
                [--experimental-cache] [-f MAIN_FILE]
```

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.

--experimental-data-trace Experimental data trace tool useful for debugging

Default: False

--experimental-dataflow-graphic Experimental graphic data flow visualization

Default: False

--experimental-cache Enables the experimental caching of compiled files.

Default: False

-f Main file

Default: "main.cf"

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]]
                {list,do,update,install,status,push,verify,commit,create,freeze}
                ...
```

Named Arguments

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

subcommand

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

Sub-commands:

list

List all modules used in this project in a table

```
inmanta modules list [-h] [-r]
```

Named Arguments

-r 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 used in this project

```
inmanta modules update [-h]
```

install

Install all modules required for this this project

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

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] name
```

Positional Arguments

name	The name of the module

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 ~=

project

Subcommand to manage the project

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

subcommand

cmd	Possible choices: freeze, init

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

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]
               [--experimental-cache]
```

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.
--experimental-cache	Enables the experimental caching of compiled files. Default: False

12.1.2 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

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

12.2 Configuration Reference

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

The options are listed per config section.

12.2.1 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

12.2.2 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

12.2.3 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

12.2.4 compiler

cache

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

Default `False`

Enables the experimental 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.

12.2.5 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

12.2.6 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.

12.2.7 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.

12.2.8 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

12.2.9 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

12.2.10 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

12.2.11 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`

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`

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

12.2.12 server_rest_transport

port

Type int

Default `8888`

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

12.2.13 unknown_handler

default

Type str

Default `prune-agent`

default method to handle unknown values

12.3 Environment Settings Reference

This document lists all settings that can be set per environment. These changes are made through the API, the dashboard 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.

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.

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

protected_environment

Type bool

Default False

When set to true, this environment cannot be cleared, deleted or decommissioned.

purge_on_delete

Type bool

Default True

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.

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.

12.4 Compiler Configuration Reference

12.4.1 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**(*, *name: str*, *description: str = None*, *requires: List[str] = []*, *freeze_recursive: bool = False*, *freeze_operator: inmanta.module.ConstrainedStrValue = '~='*, *author: str = None*, *author_email: pydantic.networks.NameEmail = None*, *license: str = None*, *copyright: str = None*, *modulepath: List[str] = []*, *repo: List[str] = []*, *downloadpath: str = None*, *install_mode: inmanta.module.InstallMode = <InstallMode.release: 'release'>*)

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. Paths are separated with `:`
- **downloadpath** – (Optional) This value determines the path where Inmanta should download modules from repositories. This path is not automatically included in in 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) This key requires a list (a yaml list) of repositories where Inmanta can find modules. Inmanta creates the git repo url by formatting {} or {0} with the name of the repo. If no formatter is present it appends the name of the module. Inmanta tries to clone a module in the order in which it is defined in this value.
- **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 '~='*

12.4.2 module.yml

Inside any module the compiler expects a `module.yml` file that defines metadata about the module.

The `module.yml` file defines the following settings:

class inmanta.module.**ModuleMetadata**(*, *name: str*, *description: str = None*, *requires: List[str] = []*, *freeze_recursive: bool = False*, *freeze_operator: inmanta.module.ConstrainedStrValue = '~='*, *version: str*, *license: str*, *compiler_version: str = None*)

Parameters

- **name** – The name of the module.
- **description** – (Optional) The description of the module
- **version** – The version of the inmanta module.
- **license** – The license for this module
- **compiler_version** – (Optional) The minimal compiler version required to compile this module.

- **requires** – (Optional) Model files import other modules. These imports do not determine a version, this is based on the install_model setting of the project. Modules and projects can constrain a version in the requires setting. 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 any submodule of this module will be set in module.yml. If it is set to true, all modules imported in any of those modules will also be set.
- **freeze_operator** – (Optional) This key determines the comparison operator used by the freeze command. Valid values are [==, ~=, >=]. *Default is '~='*

12.5 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.

12.5.1 Constants

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

Bases: enum.Enum

Log levels used for various parts of the inmanta orchestrator.

CRITICAL = 50

DEBUG = 10

ERROR = 40

INFO = 20

NOTSET = 0

TRACE = 3

WARNING = 30

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'

12.5.2 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*, *msg=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: 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: inmanta.ast.Locatable*, *msg: str*, *cause: PluginException*)
Bases: *inmanta.ast.ExternalException*

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

12.5.3 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*)
Get a type from the configuration model.

run_sync(*function: Callable[..., T]*, *timeout: int = 5*) → 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*) → None

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.

12.5.4 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*)
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

12.5.5 Handlers

inmanta.agent.handler.**cache**(*func: Optional[T_FUNC] = None*, *ignore: List[str] = []*, *timeout: int = 5000*, *for_version: bool = True*, *cache_none: bool = True*, *cacheNone: bool = True*, *call_on_delete: Optional[Callable[Any, None]] = None*) → Union[T_FUNC, Callable[T_FUNC, T_FUNC]]
decorator for methods in resource handlers to provide caching

this decorator works similar to memoization: when the decorate method is called, its return value is cached, for subsequent calls, the cached value is used instead of the actual value

The name of the method + the arguments of the method form the cache key

If an argument named version is present and for_version is True, the cache entry is flushed after this version has been deployed If an argument named resource is present, it is assumed to be a resource and its ID is used, without the version information

Parameters

- **timeout** – the number of second this cache entry should live
- **for_version** – if true, this value is evicted from the cache when this deploy is ready
- **ignore** – a list of argument names that should not be part of the cache key
- **cache_none** – cache returned none values
- **call_on_delete** – A callback function that is called when the value is removed from the cache, with the value as argument.

inmanta.agent.handler.**provider**(*resource_type: str*, *name: str*) → None
A decorator that registers a new handler.

Parameters

- **resource_type** – The type of the resource this handler 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: 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, enum.Enum, 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*, ***kwargs*) → 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*, ***kwargs*) → 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*, ***kwargs*) → 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*, *exc_info=True*, ***kwargs*) → 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*, ***kwargs*) → 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[Union[BaseModel, enum.Enum, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]]]]*) → None
update_changes(*changes: Dict[str, Tuple[Union[BaseModel, enum.Enum, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str], Union[BaseModel, enum.Enum, uuid.UUID, inmanta.types.StrictNonIntBool, int, float, datetime.datetime, str]]]*) → None
Update the changes list with changes

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

warning(*msg: str*, **args*, ***kwargs*) → 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: 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_process_events() → bool
Can this handler process events? This is a more generic version of the reload mechanism.

See the *ResourceHandler.process_events()* for more details about this mechanism.

Returns Return true if this handler processes events.

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
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

do_changes(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *changes: dict*) → 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
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.

process_events(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *events: dict*) → None
Process events generated by changes to required resources. Override this method to process events in a handler.

The default implementation provides the reload mechanism. It will call do_reload when the handler can_reload() and if at least one of the dependents have successfully deployed and there were changes. Make sure to call this method from a subclass if the reload behaviour is required.

This method is called for all dependents of the given resource (inverse of the requires relationship) that have send_event set to true and for which a deploy was started. These are the only conditions, even if all dependents have failed or no changes were deployed. It is up to the handler to filter out irrelevant events.

In case of partial deployments (e.g. incremental deploy), only those resources that are being deployed will produce an event. I.e. it is possible to receive less events then expected.

In case of failure of agent, server or the system being managed, delivery of events can not be guaranteed. Update events can be lost unrecoverably in case the agent or server fails after the update was performed, but before the event was emitted. In the current implementation, start of a new deploy while another is in progress can also causes updates to be lost.

However, while event delivery can not be guaranteed, convergence to the desired state can be reliably detected. If the record of the convergence is lost, it will be retried until it is recorded. For strong behavioral guarantees, it is better to rely on desired state than on events.

Events are best used to accelerate convergence. For example, cross agent dependencies primarily make use of the deployment log on the server to determine if their dependencies are in their desired state. To speed up convergence, events are sent to notify other agents of relevant changes to resources they depend on.

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to process the events for.
- **dict** – A dict with events of the resource the given resource requires. The keys of the dict are the resources. Each value is a dict with the items status (const.ResourceState), changes (dict) and change (const.Change). The value is also defined by inmanta.data.model.Event

run_sync(*func: Callable[T]*) → 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: 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_process_events() → bool
Can this handler process events? This is a more generic version of the reload mechanism.

See the *ResourceHandler.process_events()* for more details about this mechanism.

Returns Return true if this handler processes events.

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
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.

do_changes(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *changes: dict*) → 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
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.

process_events(*ctx:* inmanta.agent.handler.HandlerContext, *resource:* inmanta.resources.Resource, *events: dict*) → None

Process events generated by changes to required resources. Override this method to process events in a handler.

The default implementation provides the reload mechanism. It will call do_reload when the handler can_reload() and if at least one of the dependents have successfully deployed and there were changes. Make sure to call this method from a subclass if the reload behaviour is required.

This method is called for all dependents of the given resource (inverse of the requires relationship) that have send_event set to true and for which a deploy was started. These are the only conditions, even if all dependents have failed or no changes were deployed. It is up to the handler to filter out irrelevant events.

In case of partial deployments (e.g. incremental deploy), only those resources that are being deployed will produce an event. I.e. it is possible to receive less events then expected.

In case of failure of agent, server or the system being managed, delivery of events can not be guaranteed. Update events can be lost unrecoverably in case the agent or server fails after the update was performed, but before the event was emitted. In the current implementation, start of a new deploy while another is in progress can also causes updates to be lost.

However, while event delivery can not be guaranteed, convergence to the desired state can be reliably detected. If the record of the convergence is lost, it will be retried until it is recorded. For strong behavioral guarantees, it is better to rely on desired state than on events.

Events are best used to accelerate convergence. For example, cross agent dependencies primarily make use of the deployment log on the server to determine if their dependencies are in their desired state. To speed up convergence, events are sent to notify other agents of relevant changes to resources they depend on.

Parameters

- **ctx** – Context object to report changes and logs to the agent and server.
- **resource** – The resource to process the events for.
- **dict** – A dict with events of the resource the given resource requires. The keys of the dict are the resources. Each value is a dict with the items status (const.ResourceState), changes (dict) and change (const.Change). The value is also defined by inmanta.data.model.Event

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[T]*) → 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*, *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: Dict[str, str] = None*, *cwd: str = None*, *timeout: 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

12.5.6 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.

12.5.7 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
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

12.5.8 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'
Use the module's master branch.

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* and *ModuleMetadata*).

A version is released when there is a tag on a commit. This tag should be a valid version identifier (PEP440) and should not be a prerelease version. Inmanta selects the latest available version (version sort based on PEP440) that is compatible with all constraints.

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*)
Commons superclass for projects and modules, which are both versioned by git

property metadata

property name

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

This class models an inmanta configuration module

12.5.9 Project

class inmanta.module.**Project**(*path: str*, *autostd: bool = True*, *main_file: str = 'main.cf'*, *venv_path: Optional[str] = None*)
Bases: *inmanta.module.ModuleLike*[*inmanta.module.ProjectMetadata*]

An inmanta project

classmethod get(*main_file: str = 'main.cf'*) → *inmanta.module.Project*
Get the instance of the project

load() → None

classmethod set(*project:* inmanta.module.Project) → None
Set the instance of the project

12.5.10 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: 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: 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*.

12.5.11 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
The result object dict.

12.5.12 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: Any*)
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.

async classmethod get_by_id(*doc_id: uuid.UUID*, *connection: Optional[asyncpg.connection.Connection] = None*) → Optional[*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: str = 'ASC', limit: Optional[int] = None, offset: Optional[int] = None, no_obj: bool = False, connection: Optional[asyncpg.connection.Connection] = None, **query: Any*) → List[*TBaseDocument*]

Get a list of documents matching the filter args

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

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.

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*)

Bases: *inmanta.data.BaseDocument*

A specific version of the configuration model. Any transactions that update ResourceAction, Resource, Parameter and/or ConfigurationModel should acquire their locks in that order.

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
- **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: Any*)

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

class inmanta.data.**Report**(*from_postgres: bool = False*, ***kwargs: Any*)

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: Any*)

Bases: *inmanta.data.BaseDocument*

A specific version of a resource. This entity contains the desired state of a resource. Any transactions that update Resource should adhere to the locking order described in *inmanta.data.ConfigurationModel*.

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

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

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

Bases: *inmanta.data.BaseDocument*

Log related to actions performed on a specific resource version by Inmanta. Any transactions that update ResourceAction should adhere to the locking order described in *inmanta.data.ConfigurationModel*

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`

12.5.13 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, int, float, *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.

12.5.14 Rest API

The rest API is also available as a swagger spec

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: '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*)
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
- **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

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[inmanta.const.ResourceState] = 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** – Send events to the dependents of this resource

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** – Does the resource have send_events enabled?
- **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**(*tid: uuid.UUID*, *id: int*, *resource: str*, *sources: dict*)
Upload the supporting code to the server

Parameters

- **tid** – The environment the code belongs to
- **id** – The id (version) of the configuration model
- **sources** – The source files that contain handlers and inmanta plug-ins {code_hash:(file_name, provider.__module__, source_code, [req])}

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: A paused agent cannot execute any deploy operations. * unpause: A unpaused agent will be able to execute deploy operations.

Raises ***Forbidden*** – The given environment has been halted.

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: A paused agent cannot execute any deploy operations. * unpause: A unpaused agent will be able to execute deploy operations.

Raises ***Forbidden*** – The given environment has been halted.

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*) → 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

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*) → inmanta.data.model.Environment

Get an environment and all versions associated

Parameters **id** – The id of the environment to return

inmanta.protocol.methods_v2.**environment_list**() → List[inmanta.data.model.Environment]

Create a list of environments

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

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_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, str, Dict[str, Union[str, int, inmanta.types.StrictNonIntBool]]]*) → inmanta.protocol.common.ReturnValue[None]

Set a value

inmanta.protocol.methods_v2.**get_api_docs**(*format: Optional[inmanta.const.ApiDocsFormat] = <ApiDocsFormat.swagger: 'swagger'>*) → inmanta.protocol.common.ReturnValue[Union[inmanta.protocol.openapi.mo 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_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.mode

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.**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.**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*) → inmanta.data.model.Project
Get a project and a list of the ids of all environments

inmanta.protocol.methods_v2.**project_list**() → List[inmanta.data.model.Project]
Create a list of projects

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

inmanta.protocol.methods_v2.**reserve_version**(*tid: uuid.UUID*) → int
Reserve a version number in this environment.

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

12.6 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: 'runtime_error'>*, *type: str*, *message: str*, *location: 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**

12.7 Inmanta modules

12.7.1 Module apache

- License: Apache 2.0
- Version: 0.5.2
- 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

12.7.2 Module apt

- License: Apache 2.0
- Version: 0.4.7
- 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*

12.7.3 Module aws

- License: Apache 2.0
- Version: 3.1.2
- This module requires compiler version 2017.2 or higher
- Upstream project: https://github.com/inmanta/aws.git

Typedefs

typedef aws::direction

- Base type `string`
- Type constraint `((self == 'ingress') or (self == 'egress'))`

typedef aws::instance_tenancy

- Base type `string`
- Type constraint `/^(default|dedicated|host)$/`

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*

12.7.4 Module cron

- License: Apache 2.0
- Version: 1.0.1
- 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

12.7.5 Module docker

Module to manage docker based containers

- License: Apache 2.0
- Version: 0.4.7
- Upstream project: https://github.com/inmanta/docker.git

Typedefs

typedef docker::container_state

- Base type string
- Type constraint ((((self == 'running') or (self == 'stopped')) or (self == 'latest')) or (self == 'purged'))

Entities

entity docker::Container
Parents: *std::Entity*

A docker container deployed on a container service

attribute string name
The name of the docker container

attribute string image
The image to base this container on

attribute bool detach=true
Detach this container when started?

attribute docker::container_state state='running'
The state of the container

attribute string memory_limit='0'
RAM allocated to the container in human readable format ("128MB")

attribute string command=''
The command to execute

attribute string entrypoint=''
The entrypoint of the container

relation docker::Service service [1]
other end: *docker::Service.containers [0:*]*

relation docker::Port ports [0:*]
other end: *docker::Port.container [1]*

relation docker::Volume volumes [0:*]
other end: *docker::Volume.container [1]*

The following implements statements select implementations for this entity:

- *std::none* constraint `true`

entity `docker::Port`
Parents: *std::Entity*

A portmapping between the container and the host

attribute ip::ip `host_ip`='0.0.0.0'

attribute ip::port `host_port`

attribute ip::port `container_port`

relation docker::Container `container` [1]
other end: *docker::Container.ports [0:*]*

entity `docker::Registry`
Parents: *ip::services::Server*

Deploy a docker registry

The following implementations are defined for this entity:

- *docker::dockerRegistry*

The following implements statements select implementations for this entity:

- *docker::dockerRegistry* constraint `true`

entity `docker::Service`
Parents: *ip::services::Server*

A docker service

attribute ip::cidr `bridge_ip`='172.17.0.1/16'

relation docker::Container `containers` [0:*]
other end: *docker::Container.service [1]*

The following implementations are defined for this entity:

- *docker::docker*

The following implements statements select implementations for this entity:

- *docker::docker* constraint `true`

entity `docker::Volume`
Parents: *std::Entity*

A volume mounted from the host into the container

attribute string `host_path`

attribute string `container_path`

attribute string `options`='rw'

relation docker::Container `container` [1]
other end: *docker::Container.volumes [0:*]*

Implementations

implementation docker::docker

implementation docker::dockerRegistry

Resources

class docker.**Container**

This class represents a docker container

- Resource for entity *docker::Container*
- Id attribute name
- Agent name service.host.name
- Handlers *docker.ContainerHandler*

Handlers

class docker.**ContainerHandler**

- Handler name docker
- Handler for entity *docker::Container*

12.7.6 Module drupal

- License: Apache 2.0
- Version: 0.7.3
- 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

12.7.7 Module exec

- License: Apache 2.0
- Version: 1.1.4
- 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*

12.7.8 Module graph

- License: Apache V2
- Version: 0.8.3
- 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

12.7.9 Module ip

- License: Apache 2.0
- Version: 1.2.0
- 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

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

12.7.10 Module mysql

- License: Apache 2.0
- Version: 0.6.2
- 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`

12.7.11 Module net

- License: Apache 2.0
- Version: 1.0.3
- 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

12.7.12 Module openstack

- License: Apache 2.0
- Version: 3.6.12
- 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

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*

12.7.13 Module param

- License: Apache 2.0
- Version: 0.6.4
- 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.

12.7.14 Module php

- License: Apache 2.0
- Version: 0.3.1
- 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.

12.7.15 Module platform

- License: ASL 2.0
- Version: 1.0.5
- 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`

12.7.16 Module postgresql

- License: Apache 2.0
- Version: 0.1.3
- 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]

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

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

relation postgresql::User `users` [0:*]
other end: *postgresql::User.server [1]*

The following implementations are defined for this entity:

- *postgresql::postgresqlServer*

The following implements statements select implementations for this entity:

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

entity `postgresql::User`
Parents: *std::PurgeableResource*

attribute postgresql::username_t `username`

attribute string `password`

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

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`

Implementations

implementation `postgresql::db_requires`

implementation `postgresql::postgresqlServer`

implementation `postgresql::user_requires`

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*

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*

12.7.17 Module redhat

- License: Apache 2.0
- Version: 0.9.2
- 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

12.7.18 Module rest

- License: Apache 2.0
- Version: 0.2.4
- 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*

12.7.19 Module ssh

- License: Apache 2.0
- Version: 0.6.3
- 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

12.7.20 Module std

- License: Apache 2.0
- Version: 2.1.9
- This module requires compiler version 2020.1 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*
- *ip::agentConfig*
- *redhat::epel::epel7*

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')`
- *ip::agentConfig* constraint `(host.ip is defined and host.remote_agent)`
- *redhat::epel::epel7* constraint `(std::familyof(host.os,'rhel') and (host.os.version >= 7))`

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"
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=true
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::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*

12.7.21 Module ubuntu

- License: Apache 2.0
- Version: 0.4.3
- 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*

12.7.22 Module user

- License: ASL 2
- Version: 0.1.5
- 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

12.7.23 Module vyos

- License: ASL2.0
- Version: 1.3.4
- 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::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.interfaces [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 `interfaces` [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 net::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*

12.7.24 Module web

- License: Apache 2.0
- Version: 0.3.3
- 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:*]*

12.7.25 Module yum

- License: Apache 2.0
- Version: 0.6.2
- 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

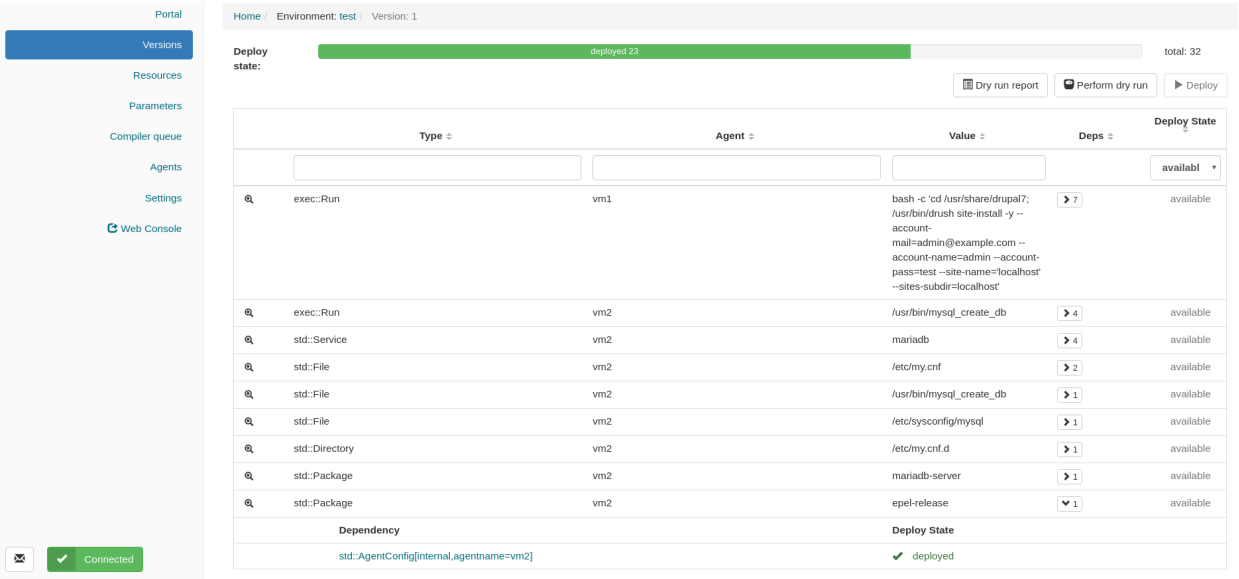
THIRTEEN

TROUBLESHOOTING

This page describes typical failure scenario's and provides a guideline on how to troubleshoot them.

13.1 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

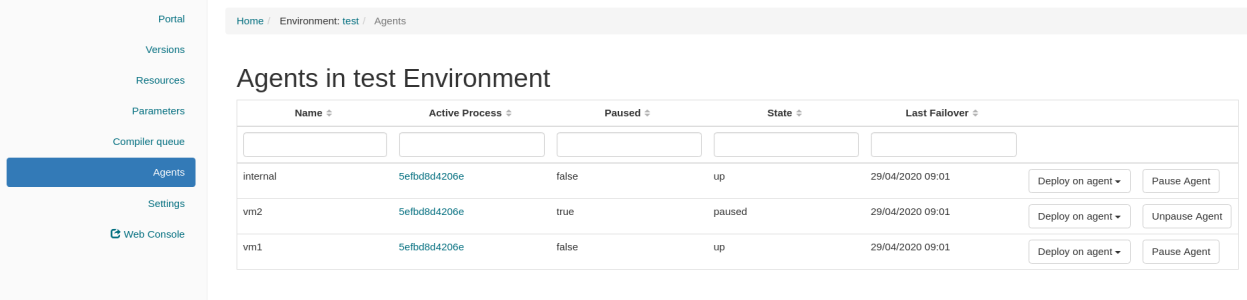


Next, go to the agents tab of the dashboard to verify the state of that agent.

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.

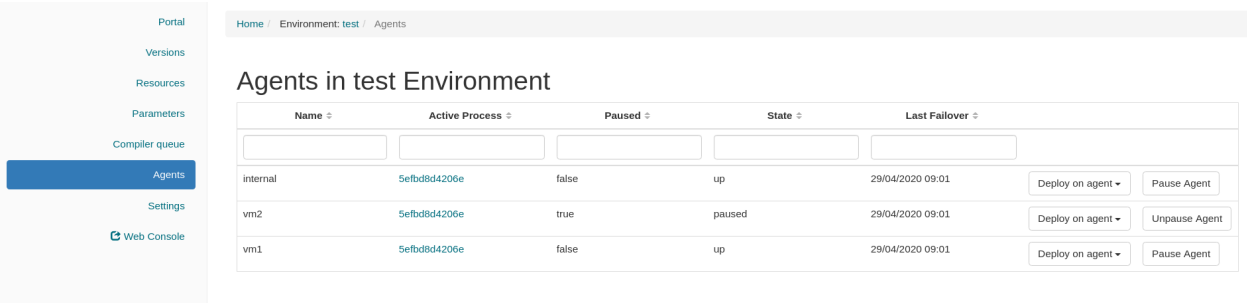


13.1.1 The agent is down

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.

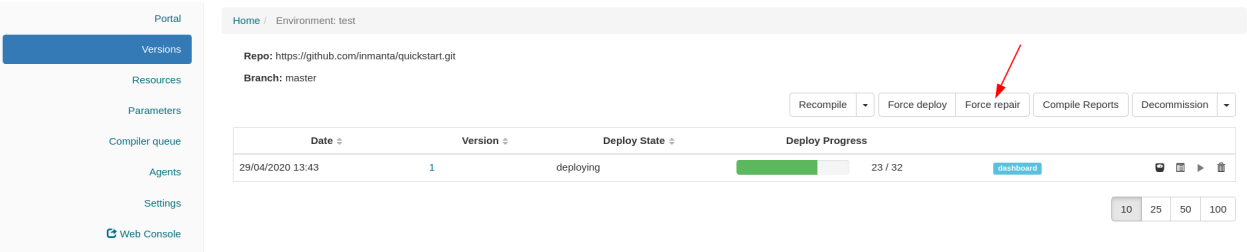
13.1.2 The agent is paused

Unpause the agent by clicking the `Unpause agent` button in the agents tab of the dashboard.



13.1.3 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`.

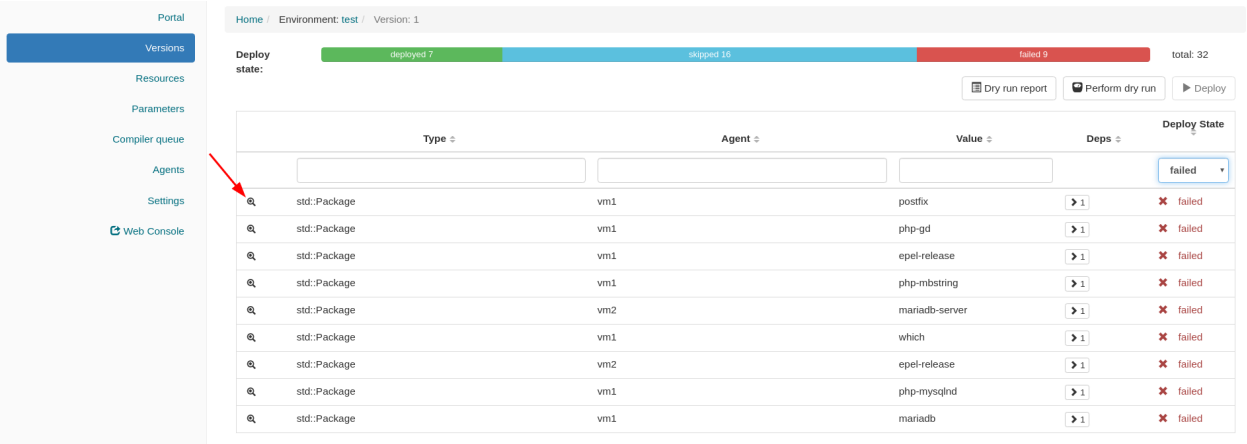
13.2 The deployment of a resource fails

When a resource cannot be deployed, it ends up in one of the following deployment states:

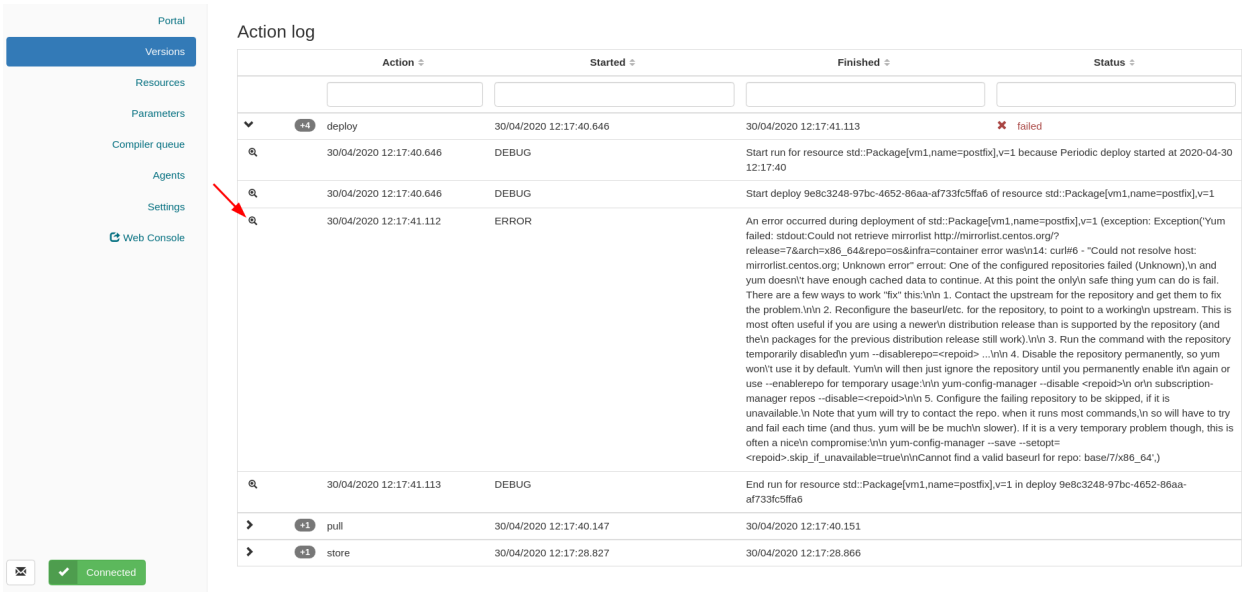
- **failed:** A resource ends up in the `failed` state when the handler of that resource raises an uncaught exception. *Check the log of the resource* to get more details about the issue.
- **unavailable:** A resource ends up in the `unavailable` state when no handler could be found to deploy that resource. *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.

13.2.1 Read the logs of a resource

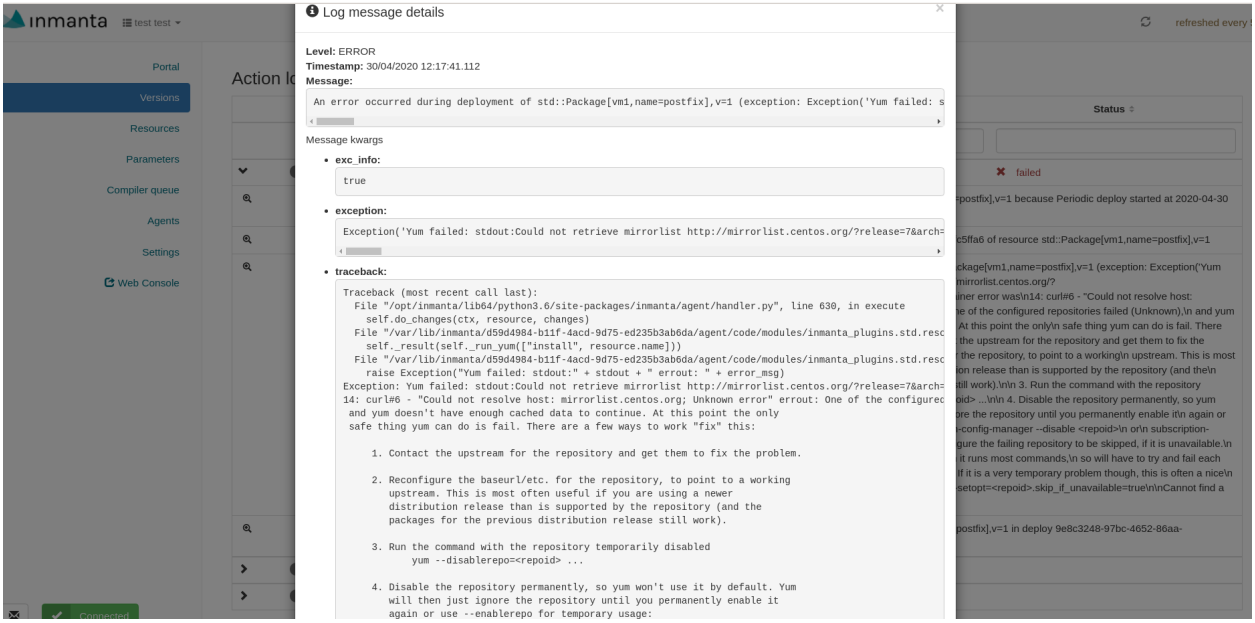
This section describes how to obtain the logs for a specific resource. In the versions tab of the dashboard, click on the version of the configuration model being deployed to get a list of all the resource in that configuration model. Next, click on the magnifier in front of a resource, as shown in the 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.



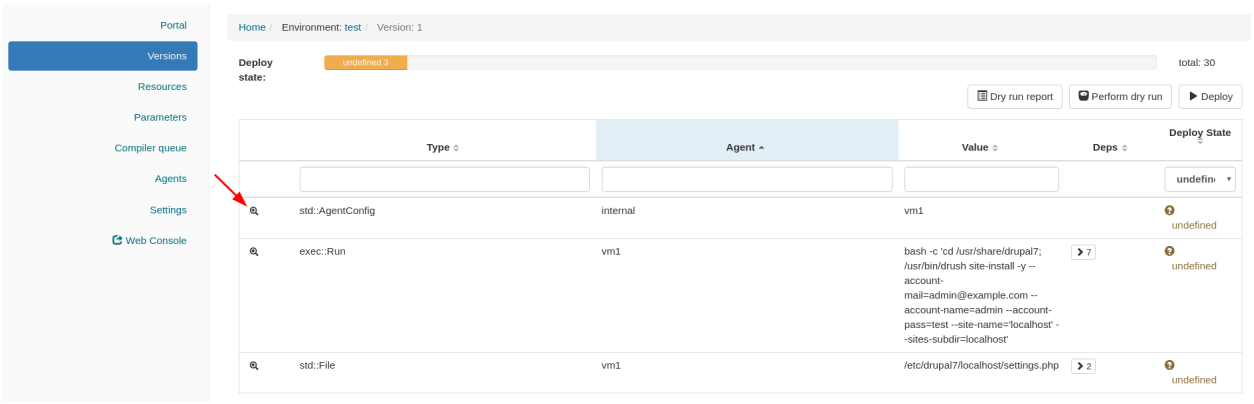
In the figure below, the traceback of the exception is shown.



13.2.2 Check which facts are not yet resolved

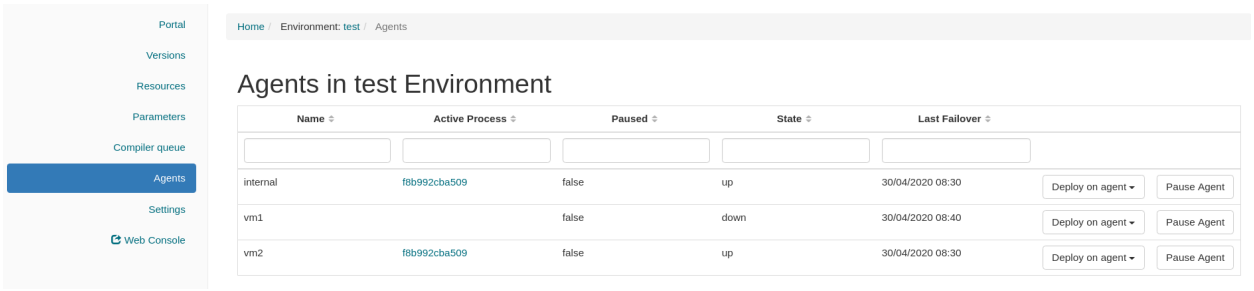
To find out which fact of a certain resource is not yet resolved, click on the magnifier in front of the resource in the `undefined` state, as shown in the figure below.

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.



13.3 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.

13.3.1 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.

13.3.2 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
```

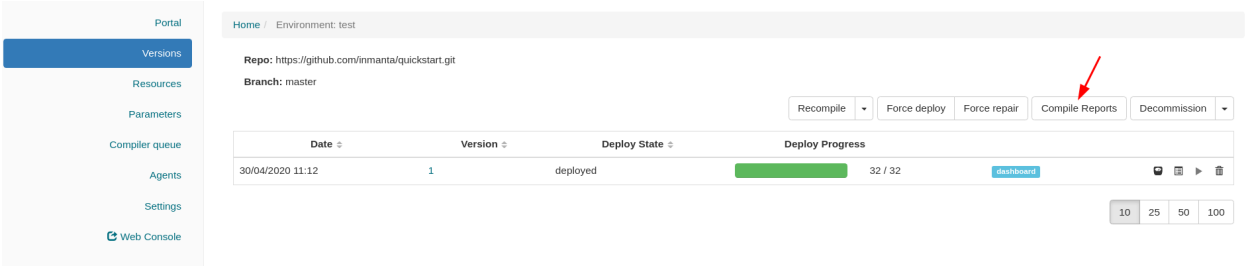
13.3.3 Potential reasons why an agent doesn't start

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.

13.4 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.

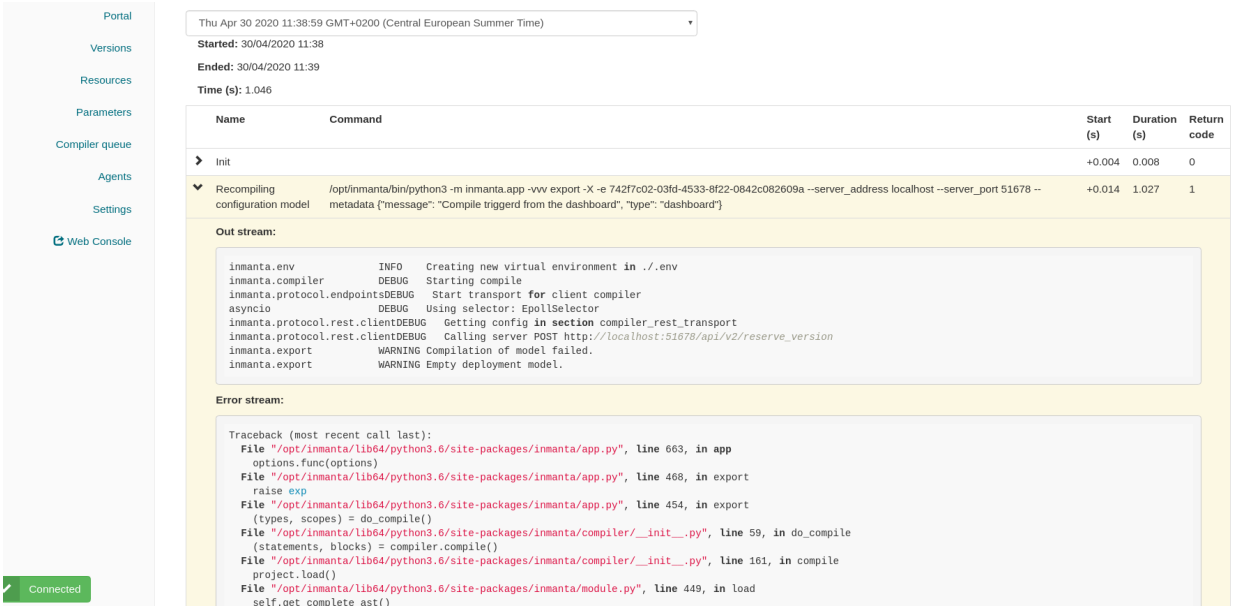


Each step of the compile process is shown, together with the output produced by that step and the return code. Verify that the timestamp of the compile report corresponds to the time the compilation was triggered in the dashboard. If no compile report was generated or the compile report doesn't show any errors, check the server logs as well. By default the server log is present in `<config.log-dir>/server.log`.

13.5 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
```



13.6 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

FOURTEEN

CHANGELOG

14.1 Release 2021.2 (2021-05-05)

14.1.1 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`

14.1.2 Inmanta-dashboard: release 3.7.0 (2021-05-05)

No changelog entries.

CHAPTER

FIFTEEN

ADDITIONAL RESOURCES

- Inmanta User Mailinglist
- Inmanta Developer Mailinglist
- Inmanta Twitter

CHAPTER

SIXTEEN

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