mr-provisioner Documentation

Release

Linaro

Nov 23, 2018

User Documentation

The main documentation for the site is organized into a couple sections:

- *User Documentation*

Information about development is also available:

- *Developer Documentation*
- *Design Documentation*

CHAPTER 1

Getting started

1.1 Install external dependencies

mr-provisioner requires the following external dependencies to be installed:

- virtualenv
- python-pip
- ipmitool

Additionally, mr-provisioner also relies on the following external services:

- postgresql
- tftp-http-proxy
- ws-subprocess

1.2 Create a virtual env

Set up a virtual environment to run the application:

```
virtualenv --python=python3 env
```

NOTE: Make sure to specify python 3 if your system doesn't use it by detault (-p PYTHON_EXE)

After that, activate the virtual env:

```
source env/bin/activate
```

1.3 Install requirements

First, make sure the virtual env is activated (see above). Then, install the required python dependencies by running:

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

1.4 Configuration file

Copy the example configuration file from *examples/config.ini* to a location of your chosing, and adjust it according to your needs. At the least, you will have to configure the database uri and the TFTPRoot setting.

See *Configuration options* for more information.

1.5 Set up database

Create a new database and user for *mr-provisioner* if you haven't already set one up:

```
sudo -u postgres -s
psql

CREATE DATABASE <dbname>;
CREATE ROLE <username> WITH PASSWORD '<password>' LOGIN;
GRANT ALL PRIVILEGES ON DATABASE <dbname> TO <username>;
```

Create the required tables by running the database migrations:

```
./run.py -c /path/to/your/config.ini db upgrade
```

After this, a first user called *admin* with password *linaro* will be available.

1.6 Run the app

First, make sure the virtual env is activated in the current shell.

Start up ws-subprocess:

```
/path/to/ws-subprocess -controller-url "http://localhost:5000/admin/ws-subprocess" -
↪listen "0.0.0.0:8866"
```

Start up tftp-http-proxy:

```
/path/to/tftp-http-proxy -http-base-url "http://localhost:5000/tftp/"
```

And finally, start up *mr-provisioner*:

```
./run.py -c /path/to/your/config.ini tornado -h 0.0.0.0 -p 5000
```

1.7 Next steps

mr-provisioner can be used with any DHCP server, but works best with Kea and the mr-provisioner-kea plugin. See *Kea integration* for more information. Some of the features that are only enabled with Kea include:

- Showing DHCP IP lease in the UI
- Assigning static/reserved IPs to machines

For additional deployment instructions, see *Deploy*.

CHAPTER 2

Configuration options

This section will document all the supported config.ini options. Still needs to be written.

CHAPTER 3

Kea integration

3.1 Install

3.1.1 Dependencies

To build Kea and the mr-provisioner-kea plugin, you need some libraries in addition to standard build tools:

- log4cplus (e.g. on Ubuntu: liblog4cplus-dev)
- curl (e.g. on Ubuntu: libcurl4-openssl-dev or libcurl4-gnutls-dev)
- openssl (e.g. on Ubuntu: libssl-dev)
- boost c++ (e.g. on Ubuntu: libboost-all-dev)

3.1.2 Install Kea

Download a Kea 1.2.0 source tarball from the Kea website. In the Kea source directory, run:

```
./configure --prefix=/opt/kea
make -j5
sudo make install
```

3.1.3 Install Kea mr-provisioner hook/plugin

Download a mr-provisioner-kea release compatible with the Kea release (e.g. version 0.1) from the mr-provisioner-kea releases page. In the mr-provisioner-kea source directory, run:

```
make KEA_SRC=/path/to/kea-1.2.0 KEA_PREFIX=/opt/kea
sudo make KEA_SRC=/path/to/kea-1.2.0 KEA_PREFIX=/opt/kea install
```

This will install the plugin as *libkea-hook-mr-provisioner.so* under *$(KEA_PREFIX)/lib*.

3.2 Configure

Add the following section to your Kea Dhcp4/Dhcp6 configuration section(s), adjusting the URL to your deployment of mr-provisioner:

```
"hooks-libraries": [
    {
        "library": "/opt/kea/lib/libkea-hook-mr-provisioner.so",
        "parameters": {
            "provisioner_url": "http://127.0.0.1:5000/dhcp",
            "timeout_ms": 5000
        }
    }
]
```

For additional setup information including systemd files, see *Deploy*.

CHAPTER 4

Deploy

This section will document how to properly deploy *mr-provisioner*, with sample systemd files, etc. Still needs to be written.

4.1 Systemd

4.1.1 Service files

Copy the example systemd service files in *examples/systemd* into */etc/systemd/system* and adjust the paths in them.

Reload systemd to ensure the new service files are picked up:

```
systemctl daemon-reload
```

4.1.2 Start the services

Enable the services so they start automatically at boot time:

```
systemctl enable mr-provisioner.service
systemctl enable mr-provisioner-ws.service
systemctl enable mr-provisioner-tftp.service
```

Start the services:

```
systemctl start mr-provisioner.service
systemctl start mr-provisioner-ws.service
systemctl start mr-provisioner-tftp.service
```

Optionally, if you followed *Kea integration*, also enable and start the Kea services:

```
systemctl enable kea-dhcp4.service
systemctl start kea-dhcp4.service
```

CHAPTER 5

Upgrade

This section will document how to upgrade from one release to another.

5.1 Releases

New releases of `mr-provisioner` are available in mr-provisioner's github.

New releases are published when there are enough fixes or new features to grant them. An occasional release may be done when a critical issue is found and fixed.

5.2 Download and install

Note these instructions will need to be contextualized to whichever way your deployment is running the different services. The recommended way to do upgrades is to set up the new version from scratch and then point the service file to the new version, or use a symlink to point to the current version's directory.

1. Download the new release `tar.gz` from mr-provisioner's github and extract it to a new directory.
2. Go into the new version's directory and follow the usual installation instructions:

```
virtualenv --python=python3 env
source env/bin/activate
pip install -r requirements.txt
```

3. Stop the service that runs the old version.
4. Make a backup of the database before the database upgrade.
5. Upgrade the database:

```
./run.py -c /path/to/your/config.ini db upgrade
```

6. Start the service using the newly installed version.

CHAPTER 6

Design Overview

`mr-provisioner` helps you automate and provision servers, manage and assign your hardware. It can handle multiple architectures.

`mr-provisioner` handles the entire network boot process from controlling DHCP and handling TFTP requests to providing installation configuration files. See *Netboot explained* for detailed explanation.

Hardware reservation is available and it enables assigning machines to users. Users can then manage the OS that gets installed and access the console. `mr-provisioner` provides users restart and PXE reboot functionality by talking to BMC (see *BMC Access*) and *Console Access* via ipmi.

CHAPTER 7

Netboot explained

In the first version of `mr-provisioner` the installation process is simple, see the following diagram: In the diagram the *Client* is the server that is going to be reprovisioned. The *DHCP server* in this case is `dnsmasq` configured to serve `bootfile-grub-aa64.efi` as follows:

```
dhcp-boot=bootfile-grub-aa64.efi,,<tftp-proxy-ip-address>
```

The bootfile should be placed inside the `tftp` folder for `mr-provisioner` to serve.

`mr-provisioner` relies on the bootloader requesting for a configuration file with the MAC in its name. E.g. `(tftp)/grub/01-${MAC}`.

`mr-provisioner` will serve any file that it is requested by the tftp proxy if the file exists.

CHAPTER 8

Console Access

Engineers can access the console of the server they are installing from the web app, see diagram: The user makes a request to `mr-provisioner`, and a token for the session is generated, then the browser of the user makes a websocket request to **'ws-subprocess'_**, who then has all that is needed to get the command to run the console from `mr-provisioner`.

CHAPTER 9

BMC Access

There are two BMC types supported by `mr-provisioner`. Plain BMC is the most common one. Moonshot BMC exists separately because it needs to support double bridging. `mr-provisioner` uses ipmitool to access both BMC types.

CHAPTER 10

Getting started

10.1 Install development requirements

First, make sure the virtual env is activated. Then, install additional development requirements:

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

10.2 Run development server

First, make sure the virtual env is activated.

Start up the development server by running:

```
./run.py -c /path/to/your/config.ini runserver -h 0.0.0.0 -p 5000 -d -r
```

10.3 Develop with Docker!

Alternatively, use the standalone docker development environment.

Usage:

    ./scripts/docker-dev.sh

The script creates a standalone local development environment in an ephemeral docker container.

The container will operate as your UID and mount in your local mr-provisioner source path to /work. It will install mr_provisioner, set up and start postgresql, run all database migrations, run tests, run linters (javascript and python), run mr_provisioner on localhost:5000 in the background, and also return a bash shell.

From the docker shell, interactively run "make test", "make lint", etc while editing files in real-time.

Log into the web interface with username admin, password linaro @ http://localhost:5000/

CHAPTER 11

Working with documentation

In the *docs/* directory, generate the documentation by running:

```
make html
```

Or, you can start up a live-updating documentation server by running:

```
sphinx-autobuild . _build
```

The documentation will then be available at http://127.0.0.1:8000

CHAPTER 12

Database migrations

12.1 Creating a new migration

A new migration can either be autogenerated, or created manually.

12.1.1 Autogenerated migration

Automatically generate a migration based on detected model changes by running:

```
./run.py -c /path/to/your/config.ini db migrate -m "some relevant description"
```

12.1.2 Manual migration

Generate a new empty migration file to write a manual migration by running:

```
./run.py -c /path/to/your/config.ini db revision -m "some relevant description"
```