Minisat Documentation

Release

Minisat

Apr 19, 2018

Contents:

CHAPTER 1

Overview

Minisat is an open source provisioning, managing and monitoring tool for virtual machines and Docker containers, built on Django Web Framework.

It offers a web interface for the user to interact with, which helps in easy manipulation of virtual machines and containers.

1.1 Features

- Current state (Running, Initializing, Shutdown) of virtual machines and containers.
- A dashboard to toggle the current state of virtual machines and containers.
- Virtual machine provisioning with a kickstart which makes the installation of the guest operating system uninteractive.
- Mapping the container port to the host port making the service available outside the container.

1.2 Virtualization API

- Minisat uses Libvirt API, which is a toolkit to manage virtualization hosts. The bindings for this API are available in C, Python, Perl, Java.
- Supports provisioning on many hypervisors like KVM, QEMU, Xen, Virtuozzo, VMWare ESX, LXC, BHyve and more.
- Minisat provisions virtual machines on a remote QEMU hypervisor.
- The facts of virtual machines are gathered using command line tools provided by Libvirt API.

CHAPTER 2

Components

Everything which is required to provision a virtual machine and Docker container are wrapped under components. Read further for complete details.

2.1 Infrastructure

Infrastructure is a remote system which uses libvirt API and QEMU hypervisor installed.

2.1.1 Compute Resource

Compute Resource is the very first step in provisioning virtual machines and running Docker containers.

- Create New

  Initially, add compute resource which includes following parameters

  - Compute resource Name.
  - IP Address of remote machine.
  - The **root** password of the remote machine.

Note: All the above details are very much essential to set up a compute resource.

- View Existing

  Once a compute resource is added, it is enlisted under **View Existing** section.

Compute Resource has various validations such as

- A unique name should be given to each compute resource.
- IP address should be valid, reachable and sshd service on compute resource should be running.
- Root password should be entered correctly.

2.1.2 Profiles

Profile allows user to set various essential parameters to create a virtual machine. A profile holds values for RAM, disk space and number of virtual CPUs.

- Create New

  The following parameters are asked to add a profile

  - Profile Name
  - RAM (in MB)
  - Virtual CPUs
  - Disk Space (in GB)

- View Existing

  Previously created profiles are visible under this section. Same profile can be used multiple times.

  Note: Use appropiate profile name which will give the correct idea about all the other details included with it.

2.2 Host

Minisat is host-based virtualization in which one can have access and control over virtual machine from single server.

2.2.1 Operating System

Operating System is the most important program which runs on computer. Any distribution of Linux can be used as guest operating system for the virtual machine. Operating system url from mirrors of Fedora, CentOS can be added. Else, use a tool called rsync to fetch the operating system tree, host these files on a local HTTP server and provide the local url in location field. The latter method will be more reliable and quicker to provision virtual machines.

- Create new

  Fill the two fields

  - Operating System Name

  Name of the operating system which will give the exact idea of the guest operating system.

  - Location

  Provide the location from where the server can fetch the operating system tree.

- Already Existing

  All existing operating systems are enlisted under this section.

2.2.2 Create Host

To provision a virtual machine the following parameters need to filled, some are optional though

- Name
- Compute Resource

- Profile
- Operating System
- Activation Name
- Host Group
- Root Password

Except Name and Root Password user have to select other details from drop down as they are created earlier.

If you Have already created **Host Group** then you have to only enter

- Name
- Host Group
- Root Password

All the remaining fields are filled according to the selected host group.

2.3 Content

2.3.1 Product

While provisioning a virtual machine, packages can be added to a virtual machine. A single repository is identified under the term **Product**.

- Create New

  Consists of two fields

  - **Product Name**
    - The repository will be recognized with the product name instead of the repository URL.
    - Mapping a repository URL to a name, makes identifying a repository URL with the help of product name easy.
  - **Product URL**
    - The location from where the repository for a package can be added.

  Note: A single product name will hold only one URL of a repository not more than that.

- View Existing

  All existing products are enlisted here along with their repository URL.

2.3.2 View

A single view consists of multiple products along with their corresponding repository URLs.

- Create New

  Consists of

  - **View Name**
    - Multiple products will be recognized with a single name, **View Name**.

        * If a view is selected, all the underlying products consisted in that view are added.
    - **Select Products**
        * To create a view, one or more products can be selected.
        * The view will now consist of the selected products.

- View Existing

  All existing view are enlisted here along with the included products and their corresponding repository URLs.

2.3.3 Activation Key

A single activation key consists of multiple views, and each of these views will consist of multiple products. **Activation Key**, **View**, **Product** exhibit a hierarchy, **Product** being at the top, followed by **View**, and **Activation Key** being at the bottom. The hierarchical structure allows the server to inherit views from activation key and products from view.

Create New

- Consists of
    - **Activation Name**
        * Multiple views are bundled inside a single activation key along with the products that they consist of.
        * If an activation key is selected, all the underlying views along with products consisted in that view are added.
    - **Select View**
        * To create an activation key, one or more views can be selected.
        * The key will now consist of the selected views.
- View Existing

  All existing activation keys are enlisted here along with the multiple views and products that they consist of.

2.4 Containers

Containerization is a solution to reliable sofware delivery. They offer better consistency between testing environments and production environment. Deployment of application with containers is perfect for microservices approach.

For now, Minisat can run Docker containers only. Support for other kind of containers like LXC , CoreOS's rkt will soon be added.

2.4.1 New Container

- Docker image name and tag name is to be known before running it on any compute resource.
- Container is assigned a name so as to identify it on the dashboard.
- Host port and container port are mapped to each other which makes services running inside container accessible from outside.

- If image is not available on the selected compute resource, then it is pulled from Docker registry and then run accordingly.

2.4.2 Local Images

- Docker images available on remote compute resources are displayed with details such as **Image Name & Tag**, **Image ID**, **Created**, **Size**.
- Any new image found on any compute resource will be enlisted here.

CHAPTER 3

Installation

System Requirements

- 64-bit Architecture
- A minimum of 250GB storage and 4GB memory
- Developed and tested on Fedora 27

Prerequisites

- All system should have Libvirt API installed for virtual machine provisioning.

```
# dnf install qemu-kvm qemu-img virt-manager libvirt libvirt-python libvirt-client␣
↪virt-install -y
```

- All compute resources should have Docker installed for running Docker containers. To install Docker on Fedora follow
- Server should have Docker machine installed.

```
# curl -L https://github.com/docker/machine/releases/download/v0.13.0/docker-machine-
↪`uname -s`-`uname -m` >/tmp/docker-machine && sudo install /tmp/docker-machine /usr/
↪local/bin/docker-machine
```

- Server should have SSH public key or it can be generated using SSH keygen.

```
# ssh-keygen -t rsa -f /root/.ssh/id_rsa -q -P ""
```

Installation

From Docker

To containerize Minisat, first clone the Github Minisat repository using and go to directory Minisat

```
# git clone https://github.com/miniSat/minisat.git
# cd Minisat
```

Build docker image using

```
# docker build -t minisat:latest .
```

After building the image now run the image using

```
# docker container run -it -p 8000:8000 minisat:latest 0.0.0.0:8000
```

Head to http://localhost:8000 for Minisat

From Source code

Minisat uses Django web framework which can be installed in Python 3 virtual environment. To create Python 3 virtual environment

```
# python3 -m venv <environment_name>
```

After that we need to activate the virtual environment by executing

```
# source <environment_name>/bin/activate
```

Now clone the Github Minisat repository from

```
# git clone https://github.com/miniSat/minisat.git
```

Minisat requires some Python modules like Django (version 2.0). We can install them by executing

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

Django ORM is used to create database.

```
# python manage.py makemigrations
```

Above command will create a Python script which will contain all SQL queries that we need to create the schema of database. The migration files are stored at `.../satellite/migrations/`.

```
# python manage.py migrate
```

It will create a database and execute the SQL queries in Python script. Minisat, uses SQLite database to store values.

Now our environment is ready to run Minisat server. To start server

```
# python manage.py runserver
```

By default, Django server is running at http://localhost:8000.

If you encounter below error

```
# Error: That port is already in use.
```

Try changing the port number while running the server

```
# python manage.py runserver <port_number>
```

CHAPTER 4

Usage

4.1 Creating Virtual Machines

Follow the steps to provision a virtual machine.

4.1.1 Step 1

First create compute resource by clicking on Infrastructure -> Compute Resource.

- Fill all the details
- Make sure there is no repetition of compute name, compute resource is reachable by Minisat server and root password should be correct.

4.1.2 Step 2

Once done with compute resource move to profiles.

Click on Infrastructure -> Profile

- Name the Profile, fill the fields of RAM, virtual CPUs and Disk Space.
- Name should not get repeated.

Click *Submit* Profile gets added to database and is enlisted in **View Existing** Section.

4.1.3 Step 3

Now, add Packages for the virtual machine.

- Click on Content -> Product

  Enter the name of *Product* and URL from where it should fetch the package repository.

- Click on Content -> View

  Enter the name for *View* and select the product from the list avialable. Select one or more products to encapsulate them into single *View*.

- Click on Content -> Activation key

  Here enter the name for activation key and select one or more *View* as per requirement.

Note: Adding packages is optional. If required then only follow **Step 3** else skip it.

4.1.4 Step 4

The next thing that comes into picture is Host Group. To make use of the same compute resources, profiles, operating systems and activation keys frequently, they can be bundled together under a single unit called *Host Group*.

Click Host Group -> Host Group

- Name the *Host Group*.
- Select Compute Resource, Profile, Operating System and Activation key from their respective drop down options.

Once a *Host Group* is created, a virtual machine is provisioned with less efforts as selecting a host group populates the other parameters necessary to provision a virtual machine.

Note: *Host Group* are advantageous when multiple virtual machines are to provisioned with little or no changes in their specifications.

4.1.5 Step 5

This is the final step of provisioning a virtual machine on remote machine.

Click Host -> Create Host

- Name the virtual machine so as to identify it on the dashboard.
- Selecting *Host Group* will populate the fields such as compute, profile, operating system and activation key according to the values the *Host Group* consists of.

Note: Choosing *host group* and *activation key* is optional.

- At the end provide **root** password for the virtual machine.

Finally, just hit the `Create Instance` button and virtual machine deployment starts at background.

4.2 Running Docker containers

Docker containers are created either from existing local images or by pulling images from Docker registry and then running them.

4.2.1 Step 1

First create compute resource.

Click on Infrastructure -> Compute Resource

- Fill all the details
- Make sure there is no current existance of name and IP address in database.
- Also see to it that compute resource is reachable to Minisat server and root password is correct.
- As there is validation for the above.

Note: If a compute for virtual machine is added no need to add it again, same compute can be used for containers also.

4.2.2 Step 2

The previous step will now allow the server to perform SSH on remote Docker server and gather container related facts.

To deploy a Docker container on a compute resource

- Enter name for container
- Enter the Docker image and tag name
- Provide the host port and container port
- Select compute resource
- To run containers in background select the *checkbox* provided there.

4.2.3 Step 3

Finally hit `Run` to run image.

Check running containers on dashboard under Docker containers tab.

CHAPTER 5

Methods

5.1 Make SSH connection

This module is to make connection with newly added compute resources

satellite.modules.ssh_connect.**copy_ssh_id**(*ip_address*, *password*)
Copy SSH key with remote system

Parameters

- **ip_address** – IP address of remote system (Compute resource)
- **password** – Root password of remote system

Returns True

satellite.modules.ssh_connect.**make_connection**(*ip_address*, *password*)
Check the compute is reachable and password is correct or not

Parameters

- **ip_address** – IP address of remote system (Compute resource)
- **password** – Root password of remote system

Returns Calls the copy_ssh_id(ip_address, password) to copy SSH key else error

5.2 Manage virtual machine

This module is to manage virtual machines in Minisat

satellite.modules.vm_manage.**change_repo**(*compute_ip*, *vm_ip*, *repo_id*, *repo_flag*, *vm_name*)
Change the repo status

Parameters

- **compute_ip** – IP address of compute on which virtual machine is running

- **vm_ip** – IP address of virtual machine
- **repo_id** – repo ID
- **repo_flag** – flag of repo (enable or disable)
- **vm_name** – Name of virtual machine

Returns success or failed

satellite.modules.vm_manage.**filter_repo**(*repo_info*)
Filter the repos

Remove all unnecessary data from repos

Parameters **repo_info** – Raw repo data

Returns repo_info Cleaned repo data

satellite.modules.vm_manage.**get_memory**(*compute_ip*, *vm_name*, *vm_ip*)
Find memory consumption of virtual machine

Parameters

- **compute_ip** – Compute IP on which virtual machine is running
- **vm_name** – Name of virtual machine
- **vm_ip** – IP address of virtual machine

Returns total_mem Total memory of virtual machine

Returns free_mem Free memory of virtual machine

satellite.modules.vm_manage.**get_packages**(*compute_ip*, *vm_ip*, *root_passwd*)
Get packages installed in virtual machine

Parameters

- **compute_ip** – IP address of compute on which virtual machine is running
- **vm_ip** – IP address of virtual machine
- **root_passwd** – Root password of virtual machine

Returns package_info List of all packages in virtual machine

satellite.modules.vm_manage.**get_repo**(*activation_name*)
Get repo list included in Activation name

Parameters **activation_name** – Name of activation

Returns repo Dictionary of repo name and repo URL included in that activation

satellite.modules.vm_manage.**get_status**(*compute_name*, *compute_ip*, *vm_name*)
Get status of virtual machine

Parameters

- **compute_name** – Name of compute on which virtual machine is running
- **compute_ip** – Compute IP on which virtual machine is running
- **vm_name** – Name of virtual machine

Returns Running or Initializing

satellite.modules.vm_manage.**get_vm_repo**(*compute_ip*, *vm_ip*, *vm_name*)
Get virtual machine repo

Find repo added in virtual machine and its status whether its enable or disable

Parameters

- **compute_ip** – IP address of compute on which virual machine is running
- **vm_ip** – IP address of virtual machine
- **vm_name** – Name of virtual machine

Returns repo_info Contain of list of enabled and disabled repo

satellite.modules.vm_manage.**isOnline**(*host*)
Check whether host is online or offline

Parameters **host** – IP of remote system

Returns True if online else False

satellite.modules.vm_manage.**virsh_delete_vm**(*vm_name*, *com_ip*)
Delete the virtual machine on remote system

Parameters

- **vm_name** – Name of virtual machine
- **com_ip** – Compute IP on which virtual machine is running

Returns delete_vm_flag

satellite.modules.vm_manage.**virsh_pause_vm**(*vm_name*, *com_ip*)
Shutdown the virtual machine on remote system :param vm_name: Name of virtual machine :param com_ip: Compute IP on which virtual machine is running

Returns shut_vm_flag

satellite.modules.vm_manage.**virsh_start_vm**(*vm_name*, *com_ip*)
Starts the virtual machine on remote system :param vm_name: Name of virtual machine :param com_ip: Compute IP on which virtual machine is running

Returns start_vm_flag

satellite.modules.vm_manage.**vm_create**(*compute_ip*, *name*, *ram*, *cpus*, *disk_size*, *location_url*, *kickstart_loc*)
Create virtual machine on remote system

Parameters

- **compute_ip** – Remote system IP address
- **name** – Name of virtual machine
- **ram** – RAM size for virtual machine
- **cpus** – Number of virtual CPUS for virtual machine
- **disk_size** – Disk size for virtual machine
- **location_url** – URL location of OS
- **kickstart_loc** – location of kickstart

Returns Boolean, True if success or False

satellite.modules.vm_manage.**vm_details**(*compute_name*, *compute_ip*, *vm_id*)
Find details of virtual machine

In this function virtual machine details like ID, name, state(Running or shut), virtual CPUs, Total memory allocated, Free memory, virtual machine IP address, virtual machine MAC address, Compute name (on which it is provisioned).

Parameters

- **compute_name** – Name of compute on which virtual machine is running
- **compute_ip** – IP address of compute on which virtual machine is running
- **vm_id** – UUID of virtual machine

Returns details Dictionary of ID, name, state(Running or shut), virtual CPUS, Total memory allocated, Free memory, virtual machine IP address, virtual machine MAC address, Compute name

satellite.modules.vm_manage.**vm_ip**(*vm_name*, *compute_ip*)
Find the IP address of virtual machine

Parameters

- **vm_name** – Name of virtual machine
- **compute_ip** – Compute IP on which virtual machine is running

Returns vm_ipaddress contain IP address of virtual machine

satellite.modules.vm_manage.**vm_status**(*compute_ip*, *vm_name*, *vm_ip*)
Get status of virtual machine

Parameters

- **compute_ip** – Compute IP on which virtual machine is running
- **vm_name** – Name of virtual machine
- **vm_ip** – IP address of virtual machine

Returns Running or Initializing or Shutdown

5.3 Create kickstart file

This module generate the kickstart file for virtual machine

satellite.modules.kickstart.**kick_gen**(*vm_name*, *passwd*, *location*, *repo*)
Generate kickstart file

Parameters

- **vm_name** – Name of virtual machine
- **passwd** – Password of virtual machine
- **location** – Location of operating system
- **repo** – List of repo needed to add in virtual machine

Returns location of kickstart file

5.4 Manage Docker container

This module is to manage Docker container in Minisat

satellite.modules.docker_manage.**destroy_cont**(*cont_name*, *compute_name*)
Destroy a container

Parameters

- **cont_name** – Name of container
- **compute_name** – Name of compute

Returns Destroyed if success else 0

satellite.modules.docker_manage.**get_docker_images**(*compute=[]*)
Get Docker images available in all the compute resources

Parameters **compute** – Name of compute resources

Returns docker_dict Dictionary of docker images available

satellite.modules.docker_manage.**make_connection**(*ip_address*, *name*)
Make Docker connection using docker-machine

Parameters

- **ip_address** – IP address of remote machine
- **name** – Name of docker-machine connection

Returns True if success or False

satellite.modules.docker_manage.**run_cont**(*new_cont*, *stat*)
Run the container

Parameters

- **new_cont** – Class of container details
- **stat** – Running status

satellite.modules.docker_manage.**start_cont**(*cont_name*, *compute_name*)
Start a container

Parameters

- **cont_name** – Name of container
- **compute_name** – Name of compute

Returns Paused if success else 0

satellite.modules.docker_manage.**stop_cont**(*cont_name*, *compute_name*)
Stop a container

Parameters

- **cont_name** – Name of container
- **compute_name** – Name of compute

Returns Running if success else 0

5.5 Fetch data for dashboard

dashboard_details fetch data from remote compute system and virtual machine to display on dashboard

satellite.modules.dashboard_details.**get_vms**(*ip_list=[]*)
Get list of virtual machine and their details from remote compute resources

Parameters **ip_list** – list of IP address of remote compute resources

Returns final_dict Contain details of all the virtual machine on all compute

satellite.modules.dashboard_details.**running_containers**(*compute=[]*)
Get list of virtual machine and their details from remote compute resources

Parameters **compute** – list of IP address of remote compute resources

Returns data Contain details of all the container on all compute

5.6 Testcase of Minisat

This Module is for testing Minisat. All testcase are tested on Travis-CI and triggered when new pull request is opened. Most of the tesing is done by **Selenium** and in **headless mode**.

satellite.test_minisat.**test_operating_system**()
Test Operating System page Some testcase are already designed with their expected output. If testcase fails it will split error.

satellite.test_minisat.**test_pep8**()
Testing PEP 8 standards. Flake8 is use to check PEP 8.

Error E501 (line too long error), E122 (Continuation line missing indentation or outdented), E722 (do not use bare except) are ignored.

satellite.test_minisat.**test_product**()
Test product page Some testcase are already designed with their expected output. If testcase fails it will split error.

satellite.test_minisat.**test_profile**()
Test Profile page Some testcase are already designed with their expected output. If testcase fails it will split error.

satellite.test_minisat.**test_web_ui**()
Check Minisat is working properly by visiting all availabe pages. It visits Dashboard, compute_resource, profile, create_host, operating_system, new_container, local_images.

CHAPTER 6

Contribute

Minisat is an open source project that's licensed under the GNU General Public License version 3. All contributions gladly accepted as long as they follow our programming guidelines.

Setting up the development

To create a development environment follow steps given for Minisat installation.

Setting up test environment

Pytest is used for testing and Travis-ci for Continuous Integration. Pull requests are tested against testcases by Travis-ci. If your pull request didn't pass our test cases, you can visit the test job that failed and view its console output.

It is possible for you to run these same tests locally. To setup a testing environment, you need to download Selenium webdriver for Mozilla Firefox at mozilla geckodriver.

Extract the driver.

Export path

```
# export PATH=$PATH/:/path/of/driver
```

It will set a path variable to the webdriver.

And run the test

```
# pytest
```

To check whether your programming style matches our, use *flake8*

```
# flake8 --ignore=E501,E122,E722 minisat satellite
```

Submit Patches

Patches to fix bugs are always appreciated. Before introducing a new feature, create an issue first. If you are going to work on a specific issue, make a note in the issue section so that everyone knows what you're working on. Please try to create an issue which is specific for your patch details. - Fork the project and Clone it

On GitHub, navigate to the Minisat repository. In the top-right corner of the page, click Fork.

To clone repo

```
# git clone https://github.com/<your-user-name>/minisat.git
```

- Create a feature/topic branch

```
# git checkout -b <branchName>
```

- Make the changes required and commit the code

```
# git add <modifiedFile(s)>
# git commit -m "Fixes #<bug> - <message>"
```

- Push topic branch to your fork

```
# git push origin <branchName>
```

- Create a pull request from *<branchName>* to *testing* branch.

To create pull request follow the link.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index