Price ₹ 300

DevOps

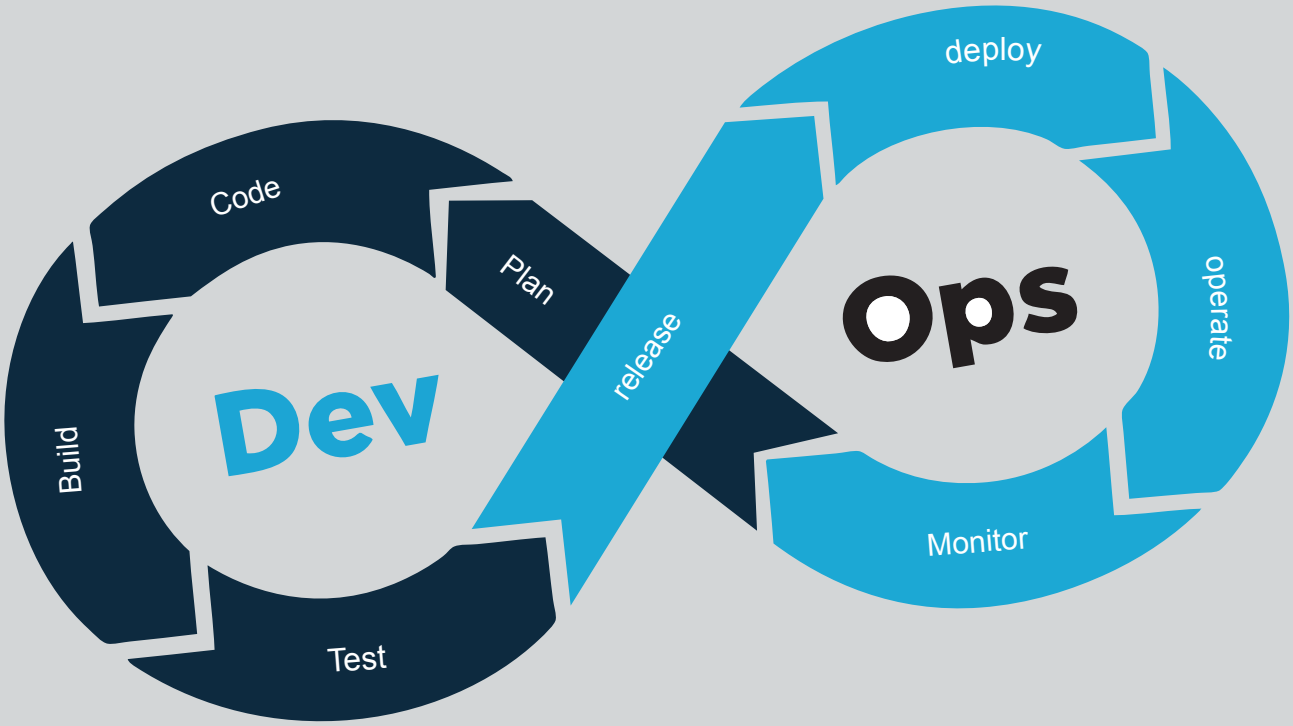


Table of Contents

Printed, published and owned by Ramesh Chopra. Published from D-87/1, Okhla Industrial Area, Phase I, New Delhi 110020.

Chapter 1

Starting the DevOps Journey Using Cucumber and Selenium

DevOps is a software building process which emphasises communication and collaboration between the teams involved in product management, software development, and operations. Cucumber is a behaviour driven development tool, which when combined with Selenium, a test recording and playback tool, improves the development team's efficiency and productivity.

It is often said that continuous change is the law of the universe and the same is true in the software industry. We have seen a variety of software development models, starting from Waterfall, V and spiral models to the incremental option. All these models have different requirements and guidelines and suit different scenarios. These days, most organisations have embraced the Agile methodology for software development.

The Agile method of developing software and applications focuses on delivering high quality products frequently and consistently, which leads to an increase in business value and profits.Table 1 lists the differences between the Waterfall and Agile software development approaches.

You can see that the Waterfall model can cause overshooting in time and resources, which can lead to huge losses to the company in terms of profits and user satisfaction. To avoid this, organisations have started adopting the Agile model. There are other reasons too, for choosing this model, some of which are listed below.

- *Client engagement:* In the Agile model, the client is engaged in the software development process at every step — before, during and after the sprint. This helps the development team to understand the client's vision clearly so that defect-free and high quality software can be developed and delivered in less time.
- *Transparency:* Since the client is actively involved in all the sprint activities, ranging from feature prioritisation and planning to reviewing and, finally, to deployment, this ensures transparency to the client.

- *Timely delivery:* Usually, the duration of a sprint is fixed and varies between one and four weeks, which forces the team to deliver features rapidly and frequently. This also helps product owners to predict the costs involved in the development and keep these under check.
- *Changing requirements:* The Agile methodology also allows teams to incorporate changes in the requirements at an early stage of the development cycle, which helps companies to develop high end products without overshooting their budgets.
- *User focused:* Instead of test cases, the Agile model employs user stories that have business and user focused acceptance criteria. This helps teams to understand the needs of the users and deliver products that can be beta tested in time, so that the necessary changes can be done at the earliest.

Table 1

Waterfall model	Agile model
Processes are divided into different phases such as design, development, tests, etc.	Here the software development process is divided into sprints, usually spanning a few weeks.
First, the final product to be developed is defined according to the customers' needs and then the different phases begin until a 'finished' product is released.	Small targets are finalised and work is done on them, sprint by sprint. This means the final product is developed in small iterations.
It's difficult to incorporate changes in the requirements since it involves cost and time.	Due to the iterative approach being used in this model, it becomes easy to incorporate changes in the requirements.
Client participation in the development becomes negligible in this process.	Clients or product owners are actively involved in the process and constant feedback is given.

Steps in the Agile approach

Let's look at the steps involved in implementing the Agile methodology.

1. *Discovery:* To develop a high quality product, one needs to have a clear vision and considerable experience in the technology used in that project. Discovery sessions are significant, since they are the basis for all the upcoming activities in the sprint. During these sessions, the clients' goals, the users' expectations and the business challenges are understood deeply so that no ambiguity remains in the minds of the team, regarding the product.

2. *Product backlog:* The result of successful discovery sessions is product backlog, which contains a list of all the features that need to be developed. These features are then classified on the basis of priority by the product owner (in discussion with the client), so that high priority features can be developed, tested and delivered first.
3. *Iterations:* After the high-level product backlog is finalised along with the priority, sprints are planned and work begins on the features mentioned in the backlog.

 **Note:** Every successive sprint in Agile is both iterative and incremental. It is iterative in the sense that it provides improvements based on the experience gained in the previous sprints, and incremental because it adds new features to the system.

4. *Cycle:* If all the features are completed and tested successfully, then the cycle stops; otherwise, additional sprints are planned to carry out the remaining work.

Agile and DevOps: The connection

Agile and DevOps – these two terms have become the buzzword these days. Though these two words are used interchangeably, there's a stark difference between them. Agile is mainly concerned with software development and the processes or steps involved in it, whereas DevOps comes into the picture after a high quality product has been developed, i.e., it is about the deployment and management of software. The term DevOps is derived from two words – development and operations. Before delving deeper into the details of DevOps, let's see how it emerged in the IT scene.

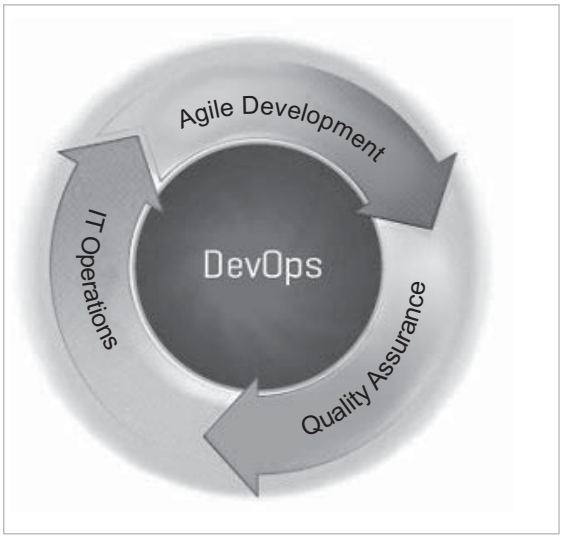


Figure 1: Agile and DevOps complement each other with the support of the QA and IT operations teams

Figure 2: Wall of confusion between the development and operations teams

We have seen how organisations reaped benefits by implementing the Agile methodology, but this model also had some hitches, which are listed below:

- There were chances of incompatibility between old features and new features during integration.
- Often, budget goals and deadlines were missed.
- There was a lack of cooperation between the development and IT operations teams.

Usually, whenever any product is released or any service is made live by an IT organisation, two departments come together to support this release – the development and operations teams. Yet, there is a lack of coordination between development activity and operations activity. The development team feels it is being paid to bring about 'change', whereas the operations team is looking at stability and considers 'change' its enemy. This conflict in mindsets often leads to inefficiency and overhead costs for the company. DevOps is a practice employed to smoothen the IT service delivery by promoting communication between development and operations teams, which is essential to increase a company's productivity. It helps the company to continually deliver software with highly stable features, faster and more frequently.

DevOps brings more flexibility to the Agile methodology and leverages its productivity. It widens the scope of Agile principles by including operations teams in its ambit instead of stopping the Agile cycle at code check-in only. So, you can deduce that Agile principles and processes can be employed as a part of DevOps. In layman's language, we can say that by using the Agile methodology, high-end products are developed and by implementing DevOps, the developed products are deployed in a timely manner. So the Agile model and DevOps complement each other, but are totally different from one another.

The need for DevOps in IT

We have seen how DevOps helps in reducing the friction between the development and operations teams. Now let's see what effects DevOps has if it's integrated into our software development process.

- Bugs can be identified and resolved at an early stage of development due to better communication and collaboration between teams.
- In the Agile model, with DevOps, there is better management of resources since there is less free time due to the presence of cross-functional teams.
- One striking feature of DevOps is the use of version control systems that can reduce the time and effort of the coder.
- Implementation of DevOps also provides an opportunity to

dedicate more time and effort to innovation and research.

Behaviour driven development (BDD)

We have just discussed the Agile model and DevOps, and the need to implement these in today's software development scenario. But since DevOps involves various teams such as developers, testers, stakeholders, etc, sometimes there can be a lack of communication between them too. Developers can misinterpret the needs of business stakeholders, and testers can misunderstand the ideas of developers. This can cause a huge negative impact on the overall productivity of the team, affecting actual deliverables. So there is a need for a common language to drive the team and bridge the communication gap. In addition, the following disadvantages were observed in Agile projects:

- Usually, user stories in the Agile model are more focused on the users and their needs rather than on the business logic of the feature. This aspect gets overlooked during sprint planning meetings and can lead to the development of unnecessary features.
- Acceptance criteria, which indicate the completion of a user story, can be understood differently by different individuals.
- Most often, Agile teams adopt the test driven development (TDD) approach, but this approach is very costly.

These weaknesses in the Agile model led to the birth of behaviour driven development (BDD). BDD is also an Agile software development process, which encourages effective communication between the business team, developers, project managers and QA by increasing the focus on business goals and business values. It was conceived by Dan North in 2003 who defines it as follows: "BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, Agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters."

BDD is an extension of TDD with a few minor differences such as:

1. Tests in BDD are written in plain English.
2. Tests are more behaviour focused and deal with the functionality of the application.
3. BDD uses extensive examples.

BDD has a lot of advantages over the traditional TDD approach. A few of these have been listed below.

- Since BDD involves plain English, it encourages collaboration among different parties involved in the software development cycle. Everyone has a clear understanding of the project and can contribute to planning sessions constructively.
- BDD puts more emphasis on business values and needs, which

can help developers in delivering better results because they can understand what the stakeholder wants and work accordingly.
- Due to a single language being used, there is less scope for misunderstanding and misconceptions.
- BDD also leads to better acceptance testing, since the user can also understand the user stories.

BDD testing tools

There are various tools available in the market, which support the BDD framework. Some of these are Cucumber, Specflow, Behave, JBehave, JBehave Web, Lettuce, Behat, Kahlan, etc.

Cucumber: Cucumber is the most widely used open source tool that supports behaviour driven development. It allows you to write application behaviour in a simple, English-like language known as Gherkin. It is written in Ruby but can support various languages like Java, JS, Python, .NET, C++, etc. Due to its wide language support, it can integrate with almost all testing tools and frameworks. You can read more about this tool at *https://cucumber.io/docs*.

Gherkin: Gherkin is a language that Cucumber understands. It is defined on GitHub as, "... a business readable, domain-specific language that lets you describe the software's behaviour without detailing how that behaviour is implemented." Gherkin is easy to learn and understand by non-programmers, but allows illustration of business rules in real-world domains. It has a particular syntax that contains keywords such as *scenario, given, then, and, examples, but*, etc. A sample Gherkin document looks like what's shown below:

```
Feature: Refund item
Scenario: Jeff returns a faulty microwave
Given Jeff has bought a microwave for $100
     And he has a receipt
     When he returns the microwave
     Then Jeff should be refunded $100
```

So you can see that we have specified the behaviour of the 'Refund item' system using various keywords underlined above in plain English text. You can study Gherkin and its various rules at *https://cucumber.io/docs/reference.*

Configuring Cucumber with Selenium

Now, let's look at how we can integrate Cucumber with Selenium for automated testing in DevOps.

The prerequisites are any IDE (I will be taking Eclipse Neon for tutorial

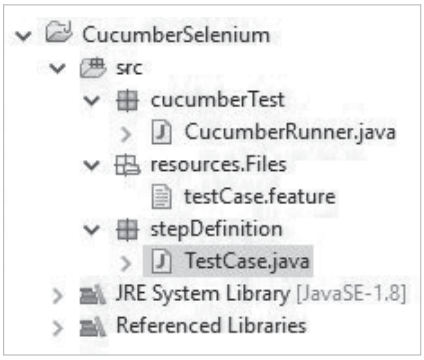


Figure 3: The file hierarchy in the project will look like this

```
1 Scenarios (?[33m1 undefined?[0m)
5 Steps (?[33m5 undefined?[0m)
0m0.000s


You can implement missing steps with the snippets below:

@Given("^\"([^\"]*)\" browser is opened$")
public void browser_is_opened(String arg1) throws Throwable {
    // Write code here that turns the phrase above into concrete actions
    throw new PendingException();
}
```

Figure 4: Methods' skeleton being generated by Cucumber by parsing the *feature* file

purposes) and the latest Java installed on your system.

***Jars required**:* The following jars/drivers need to be downloaded before starting with the configuration:

- Selenium-server-standalone (latest version)
- Selenium client for Java (latest version available)
- Cobertura-2.1.1 or above
- Cucumber-core-1.2.5 or above
- Cucumber-java-1.2.5 or above
- Cucumber-junit-1.2.5 or above
- Cucumber-jvm-deps-1.0.6 or above
- Cucumber-reporting-3.9.0 or above
- Gherkin-2.12.2
- Junit-3.4.2 or above
- Mockito-all-2.0.2-beta or above
- A driver corresponding to your browser (I will be using the Chrome driver)

After all the required files have been downloaded, follow the steps given below.

1. Launch Eclipse and create a Java project named 'CucumberSelenium' in it
2. Create three packages under the *src* folder named *cucumberTest, resources.Files* and *testScripts,* which will each contain the *runner* file, *test case feature* file and the *step definition* file.

Feature file: This file will contain our test case written in Gherkin language and will have the *feature* extension.

Runner file: Our Cucumber-Selenium framework will not have the *Main* method since we will be using JUnit to run our Java class. So this Java class will be run as a JUnit test to run our script.

 **Note:** Each feature file will have its separate runner class in this framework.

Step definition file: We have designed our *feature* file and *runner* file, but how will Cucumber get to know what code to execute for the specific test step mentioned in the *feature* file? This is taken care of by a separate Java class called *step definition* file.

3. Now create the *CucumberRunner.java* class under the *cucumberTest* package, *testCase.feature* file under the *resources.Files* package and *TestCase.java* class under the *stepDefinition* package.
4. Next, copy the following in your runner class and feature file.
   For *CucumberRunner.java*, copy:

```
package cucumberTest;

import org.junit.runner.RunWith;
import cucumber.api.CucumberOptions;
import cucumber.api.junit.Cucumber;

@RunWith(Cucumber.class)
@CucumberOptions(
       features = “src/resources/Files/testCase.feature”,
       glue = {“stepDefinition”},
       tags = {“@cucumber”}
       )

public class CucumberRunner {

}
```

For *testCase.feature,* copy:

```
Feature:To check functionality of google search page

    @cucumber
    Scenario Outline:
      Given <required> browser is opened
          When <url> is opened
          And <keyword> is searched in search box
              Then the first link in search results should be opened
              And browser is closed

    Examples:
    | required | url                 | keyword  |
| “chrome” | “http://www.google.com”    | “DevOps” |
```

5. Now run your *runner* file as a JUnit test, and you can see the empty methods being auto generated in the console output of Eclipse for each test step in the *feature* file. The significance of these methods is that Cucumber reads test steps from your *feature* file and searches for the corresponding method in the package mentioned in the *glue* option of the *runner* file. The variable values are also passed as arguments to the methods, to make use of them while scripting. So you can see how beautifully Cucumber has taken care of every minute detail while designing this framework.
6. Now copy the following code in the *step definition* file *TestCase.java*:

```
package stepDefinition;

import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import cucumber.api.java.en.Given;
import cucumber.api.java.en.Then;
import cucumber.api.java.en.When;

public class TestCase {

    WebDriver driver = null;

    @Given("^\"([^\"]*)\" browser is opened$")
    public void browser_is_opened(String arg1) throws Throwable {
      if(arg1.equals("chrome")) {
         System.setProperty("webdriver.chrome.driver", "D:\\Selenium\\
chromedriver.exe"); // file path of driver where it is stored.
         driver = new ChromeDriver();
         driver.manage().window().maximize();
         driver.manage().timeouts().implicitlyWait(30,TimeUnit.SECONDS);
      }
    }
    @When("^\"([^\"]*)\" is opened$")
    public void is_opened(String arg1) throws Throwable {
      driver.get(arg1);
    }
    @When("^\"([^\"]*)\" is searched in search box$")
    public void is_searched_in_search_box(String arg1) throws Throwable {
      driver.findElement(By.id("lst-ib")).sendKeys(arg1);
```

```
Starting ChromeDriver 2.31.488763
Only local connections are allowed.
Aug 08, 2017 8:21:40 PM org.openqa.
INFO: Detected dialect: OSS

1 Scenarios (☺[32m1 passed☺[0m)
5 Steps (☺[32m5 passed☺[0m)
0m13.935s
```

Figure 5: Output of the script depicting one scenario and five steps being passed

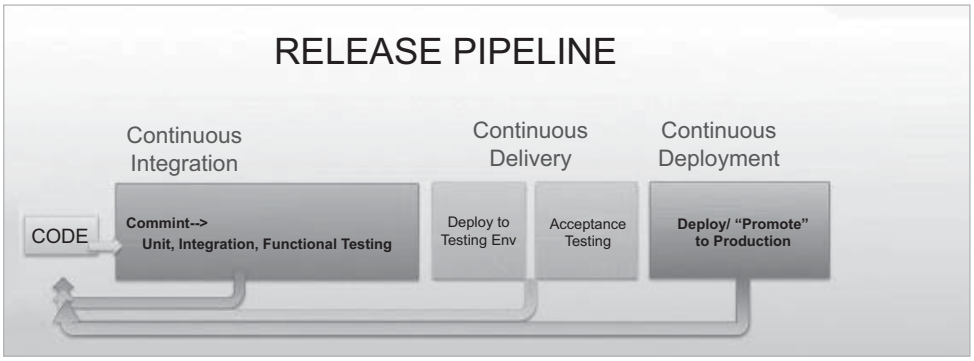


Figure 6: Complete release pipeline with CI/CD tools implemented (*Image courtesy:* Pinterest)

```
      driver.findElement(By.xpath("//*[@id='tsf']/div[2]/div[3]/center/
input[1]")).click();

    }

    @Then("^the first link in search results should be opened$")
    public void the_first_link_in_search_results_should_be_opened() throws
Throwable {
      driver.findElement(By.xpath(".//*[@id='rso']/div[3]/div/div[1]/div/div/
h3/a")).click();
    }

    @Then("^browser is closed$")
    public void browser_is_closed() throws Throwable {
        driver.close();
    }
}
```

7. Since we have defined the code, run the *runner* file and the corresponding steps will be performed according to the defined test case.
8. Once execution is complete, you can see the execution status of the steps and scenarios in the console as shown in Figure 5.
   So you can see how easily Cucumber can be configured with Selenium

Web driver to implement the BDD framework. Using this framework, you can start your DevOps journey in the testing field, within your organisation.

CI/CD tools and DevOps

This framework can also be integrated smoothly with continuous integration and continuous delivery (CI/CD) tools like Jenkins, TeamCity, Bamboo, etc, so that automated tests can be run every time developers check their code into a central repository; reports can then be published to the required stakeholders as and when required.

So we have discussed the shift from the Waterfall model to the Agile model as well as the simultaneous implementation of DevOps and the Agile methodology. Try this BDD inspired framework using Selenium to leverage your team's productivity and efficiency.

Chapter 2

An Introduction to Ansible

With this article, we begin a new series on DevOps, starting out with Ansible, which helps you to build a strong foundation. As the Ansible website proclaims, proudly, "Deploy apps. Manage systems. Crush complexity."

Ansible is an IT automation tool that is used for provisioning, configuration, deployment and managing infrastructure. The project was first released in 2012, and is written in Python. The main objective of the tool is to be simple and easy to use. It is based on an agent-less (push-based) architecture, and the playbooks are written in plain English. It also supports pull-based deployments Ansible has had pull support since 2012 and uses SSH to execute commands on remote machines. It is available under the GNU General Public License.

Installation

You can install Ansible using your GNU/Linux distribution package manager.

On Fedora, you can use Yum to install Ansible, as follows:

```
$ sudo yum install ansible
```

If you are using RHEL or CentOS, install the epel-release, and then use the Yum command to install Ansible.

On Ubuntu, you need to add the *ppa* repository before installing the tool, as shown below:

```
$ sudo apt-get install software-properties-common
$ sudo apt-add-repository ppa:ansible/ansible
```

```
$ sudo apt-get update
$ sudo apt-get install ansible
```

The Ansible documentation encourages Debian users to access the Ubuntu repository to obtain Ansible. You need to add the following line to */etc/apt/sources.list:*

```
deb http://ppa.launchpad.net/ansible/ansible/ubuntu trusty main
```

You can then install the software using the following commands:

```
$ sudo apt-get update
$ sudo apt-get install ansible
```

The Parabola GNU/Linux-libre distribution is a derivative of Arch Linux, without the binary blobs. You can install Ansible using the pacman utility:

```
$ pacman -S ansible
```

The latest Ansible version 2.2 (as of date) is what we will use in this article. Ansible is also available for BSD variants, Mac OS X, and Windows. You are encouraged to refer to the Ansible documentation for more information.

Virtualisation

Ansible can be used to provision new machines and also configure them. Instead of using bare metal machines, you can create multiple virtual machines (VMs) on your system. Lots of free and open source software (FOSS) virtualisation software is available.

QEMU is a machine emulator and virtualiser. It can also use host CPU support to run guest VMs for better performance. It is written by Fabrice Bellard, and released under the GNU General Public License (GPL). You can install it on Parabola GNU/Linux-libre, using the following command:

```
$ sudo pacman -S qemu
```

KVM or kernel-based virtual machine has direct support in the Linux kernel. It requires hardware support to be able to run guest operating systems. It is written in C, and is released under the GNU General Public License.

You need to check if your hardware first supports KVM. The *'lscpu'* command will show an entry for *'Virtualization'* if there is hardware support. For example:

```
$ lscpu
```

```
Architecture:          x86_64
CPU op-mode(s):        32-bit, 64-bit
Byte Order:            Little Endian
CPU(s):                4
On-line CPU(s) list:   0-3
Thread(s) per core:    2
Core(s) per socket:    2
Socket(s):             1
NUMA node(s):          1
Vendor ID:             GenuineIntel
CPU family:            6
Model:                 78
Model name:            Intel(R) Core(TM) i5-6200U CPU @ 2.30GHz
Stepping:              3
CPU MHz:               2275.341
CPU max MHz:           2800.0000
CPU min MHz:           400.0000
BogoMIPS:              4801.00
Virtualization:        VT-x
L1d cache:             32K
L1i cache:             32K
L2 cache:              256K
L3 cache:              3072K
NUMA node0 CPU(s):     0-3
```

```
Flags:                 fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge
mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall
nx pdpe1gb rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl
xtopology nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl
vmx est tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe
popcnt tsc_deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch
epb intel_pt tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2
smep bmi2 erms invpcid mpx rdseed adx smap clflushopt xsaveopt xsavec xgetbv1
xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
```

You can also check the */proc/cpuinfo* output as shown below:

```
$ grep -E “(vmx|svm)” --color=always /proc/cpuinfo
```

```
flags           : fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov
pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology
nonstop_tsc aperfmperf eagerfpu pni pclmulqdq dtes64 monitor ds_cpl vmx est
```

```
tm2 ssse3 sdbg fma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_
deadline_timer aes xsave avx f16c rdrand lahf_lm abm 3dnowprefetch epb intel_pt
tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust bmi1 avx2 smep bmi2
erms invpcid mpx rdseed adx smap clflushopt xsaveopt xsavec xgetbv1 xsaves
dtherm ida arat pln pts hwp hwp_notify hwp_act_window hwp_epp
```

The Libvirt project provides APIs to manage guest machines on KVM, QEMU and other virtualisation software. It is written in C, and is released under the GNU Lesser GPL. The virtual machine manager (VMM) provides a graphical user interface for managing the guest VMs and is written in Python.

You can install all this software on Parabola GNU/Linux-Libre using the following command:

```
$ sudo pacman -S libvirt virt-manager
```

A screenshot of VMM is provided in Figure 1.

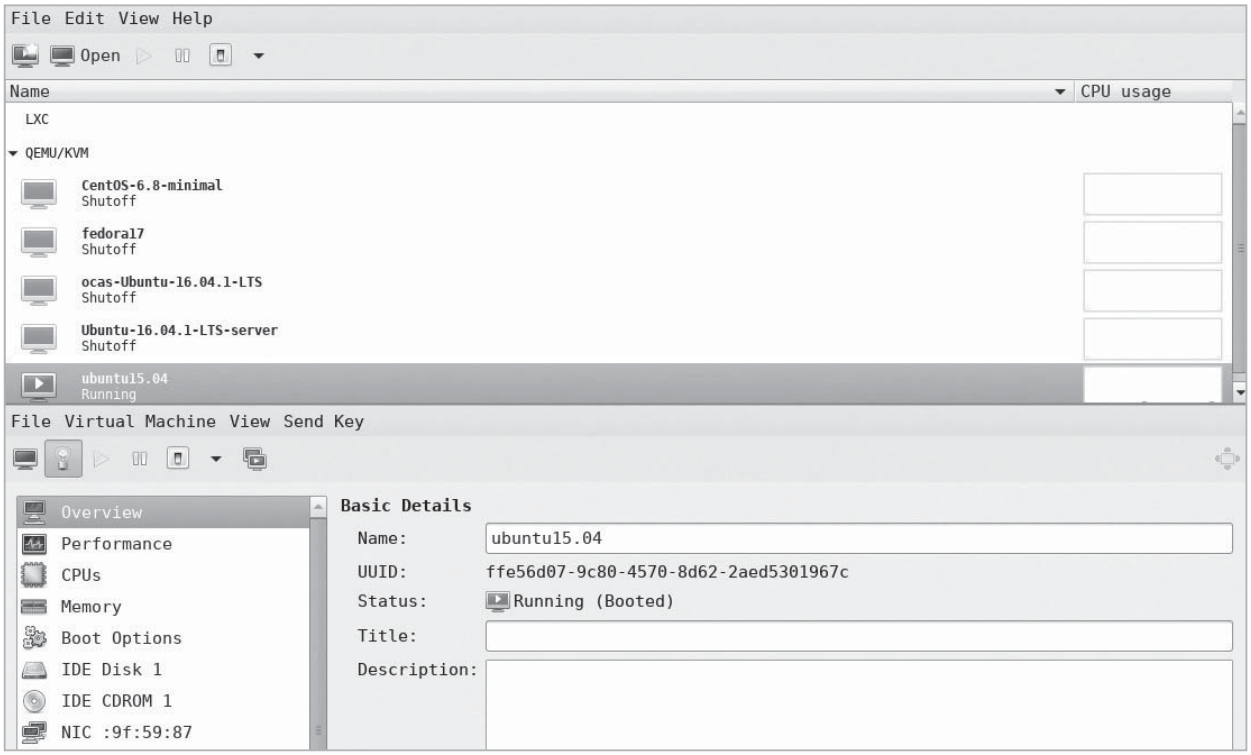


Figure 1: Virtual Machine Manager

Check your distribution documentation to install the appropriate virtualisation software packages.

You can use the VMM to create a new virtual machine, and install a GNU/Linux distribution using a *.iso* image. You can specify RAM, disk size and follow the installation steps for your particular distro. You can also import an existing *.qcow2* disk image to use it as a virtual machine.

Ansible with libvirt-VM

The version of Ansible used for this article is given below:

```
$ ansible --version
ansible 2.2.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = Default w/o overrides
```

If you have the *sshd* daemon running on your local machine, you can use Ansible to test it. For example, a ping test on the localhost is shown below:

```
$ ansible localhost -m ping
localhost | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

You can also check how long the system has been up and running using the following commands:

```
$ ansible localhost -a uptime
localhost | SUCCESS | rc=0 >>

 11:00:20 up  4:09,  0 users,  load average: 0.18, 0.14, 0.11
```

You can execute a shell command on the remote machine (localhost, in this case) as illustrated below:

```
$  ansible localhost -a “date”
localhost | SUCCESS | rc=0 >>
Sun Feb  5 11:24:53 IST 2017
```

The ‘setup’ command provides details of the remote target machine. A snippet output is provided below:

```
$  ansible localhost -m setup
localhost | SUCCESS => {
    “ansible_facts”: {
        “ansible_all_ipv4_addresses”: [
            “192.168.10.1”,
            “192.168.5.6”
        ],
```

```
        "ansible_all_ipv6_addresses": [
            "fe90::fc24:ff:feb9:cb61",
            "ff80::5846:fac1:6afc:2e30"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "06/12/2016",
        "ansible_bios_version": "R00ET45W (1.20 )",
        "ansible_cmdline": {
            "BOOT_IMAGE": "/vmlinuz-linux-libre",
            "cryptdevice": "/dev/sda1:cryptroot",
            "quiet": true,
            "root": "/dev/mapper/cryptroot",
            "rw": true
        },
        ....
```

An Ubuntu 15.04 instance with VMM is used in the following examples with Ansible. The IP address of the instance is added to */etc/hosts*:

```
192.168.122.250 ubuntu
```

The */etc/ansible/hosts* file contains the following:

```
ubuntu
```

You can now do a ping test from the host to the Ubuntu VM using the following command sequence for the user 'xetex':

```
$  ansible ubuntu -m ping -u xetex --ask-pass
SSH password:
ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

To avoid prompting for the password, you can add the localhost public SSH key to the VM, as follows:

```
$  ssh-copy-id -i ~/.ssh/id_rsa.pub xetex@ubuntu
```

```
/usr/bin/ssh-copy-id: INFO: Source of key(s) to be installed: "/home/user/.ssh/
id_rsa.pub"
```

```
/usr/bin/ssh-copy-id: INFO: attempting to log in with the new key(s), to filter
out any that are already installed
/usr/bin/ssh-copy-id: INFO: 1 key(s) remain to be installed -- if you are
prompted now it is to install the new keys
xetex@ubuntu's password:

Number of key(s) added: 1
```

Now try logging into the machine, with *ssh xetex@ubuntu* and check to make sure that only the key(s) you wanted were added.

You can now issue the following command to get the same result:

```
$ ansible ubuntu -m ping -u xetex

ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

For the Ubuntu system, you can also add the defined user in the */etc/ansible/hosts* file as follows:

```
ubuntu ansible_ssh_host=ubuntu ansible_ssh_user=xetex
```

The ping command is now simplified to:

```
$  ansible ubuntu -m ping

ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

You can now try the earlier Ansible commands on the target Ubuntu VM as illustrated below:

```
$ ansible ubuntu -a uptime

ubuntu | SUCCESS | rc=0 >>
 12:32:14 up 25 min,  3 users,  load average: 0.02, 0.07, 0.06
$ ansible ubuntu -a date

ubuntu | SUCCESS | rc=0 >>
```

```
Sun Feb  5 12:32:45 IST 2017
$  ansible ubuntu -m setup
ubuntu | SUCCESS => {
    “ansible_facts”: {
        “ansible_all_ipv4_addresses”: [
            “192.168.122.250”
        ],
        “ansible_all_ipv6_addresses”: [
            “ff20::5034:ff:fa9f:6123”
        ],
        “ansible_architecture”: “x86_64”,
        “ansible_bios_date”: “04/01/2014”,
        “ansible_bios_version”: “1.10.1-20151022_124906-anatol”,
        “ansible_cmdline”: {
            “BOOT_IMAGE”: “/boot/vmlinuz-3.19.0-15-generic”,
            “quiet”: true,
            “ro”: true,
            “root”: “UUID=f43c2c72-5bc7-4a97-9a43-12e634ae232af”,
            “splash”: true,
            “vt.handoff”: “7”
        },
        …
```

Chapter 3

Ansible Deployment of LAMP and WordPress

This is the second article in the DevOps series, and covers the installation of a LAMP stack and WordPress, using Ansible.

In this article, we are going to learn how to automate the deployment of a LAMP stack and install WordPress. LAMP stands for Linux, Apache (a Web server), MySQL (a database) and PHP (server-side scripting). It is a technology stack on which you can deploy different Web applications. We are also going to explore the installation of WordPress, which is free and open source software for creating websites and blogs.

Installing Linux

A Parabola GNU/Linux-libre x86_64 system is used as the host system. An Ubuntu 15.04 image runs as a guest OS using KVM/QEMU. Ansible is installed on the host system using the distribution package manager. You should be able to issue commands from Ansible to the guest OS. For example:

```
$ ansible ubuntu -m ping
ubuntu | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

The */etc/hosts* file already has an entry for the guest Ubuntu VM.

```
192.168.122.250 ubuntu
```

On the host system, we will create a project for our playbooks. It has the following directory structure:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /playbooks/admin/
```

An 'inventory' file is created inside the *inventory/kvm* folder that contains the following:

```
ubuntu ansible_host=192.168.122.250 ansible_connection=ssh ansible_user=xetex
```

Installing Apache

We will first install and test the Apache Web server on the guest Ubuntu system. Let's then create a *playbooks/configuration/apache.yml* file with the following content:

```
---
- name: Install Apache web server
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [web]
  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes
    - name: Install Apache
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - apache2
    - wait_for:
        port: 80
```

On the Ubuntu guest system, the playbook runs *apt-get update* and then installs the *apache2* package. The playbook finishes after the server has started, and is listening on port 80. Open a terminal, enter the *ansible/* folder, and execute the playbook as shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/apache.yml
-K
SUDO password:
```

```
PLAY [Install Apache web server] ****************************
TASK [setup] ************************************************
ok: [ubuntu]
TASK [Update the software package repository] ***************
changed: [ubuntu]
TASK [Install Apache] ***************************************
changed: [ubuntu] => (item=[u'apache2'])
TASK [wait_for] *********************************************
ok: [ubuntu]
PLAY RECAP **************************************************
ubuntu                     : ok=4    changed=2    unreachable=0    failed=0
```

The '-K' option is to prompt for the sudo password for the 'xetex' user. You can increase the level of verbosity in the Ansible output by passing '-vvvv' at the end of the *ansible-playbook* command. The more number of times 'v' occurs, the greater is the verbosity level.

If you now open *http://192.168.122.250*, you should be able to see the default Apache2 *index.html* page as shown in Figure 1.

Installing MySQL

The next step is to install the MySQL database server. The corresponding playbook is provided below:

```
---
- name: Install MySQL database server
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [database]
  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes
    - name: Install MySQL
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - mysql-server
        - mysql-client
        - python-mysqldb
    - name: Start the server
```

```
    service:
      name: mysql
      state: started
  - wait_for:
      port: 3306
  - mysql_user:
      name: guest
      password: '*F7B659FE10CA9FAC576D358A16CC1BC646762FB2'
      encrypted: yes
      priv: '*.*:ALL,GRANT'
      state: present
```

The package repository is updated and the necessary MySQL packages are installed. The database server is then started, and we wait for the server to be up and running. A 'guest' user account with 'osfy' as the password is created for use in our Web application. The chosen password is just for demonstration purposes. When used in production, please select a strong password with special characters and numerals.

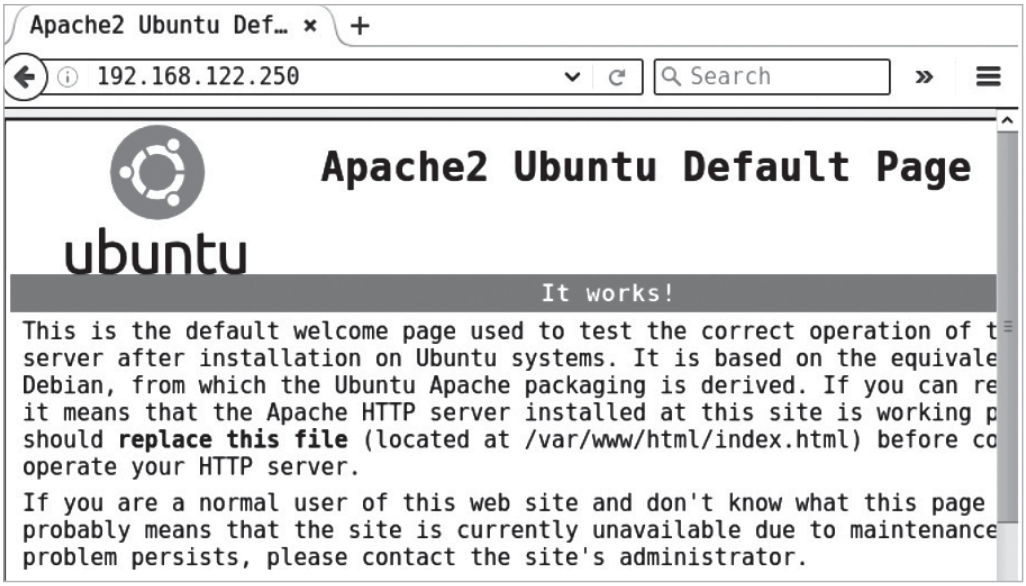


Figure 1: Apache2 Ubuntu default page

You can compute the hash for a password from the MySQL client, as shown below:

```
mysql> SELECT PASSWORD('osfy');
+-------------------------------------------+
| PASSWORD('osfy')                          |
+-------------------------------------------+
| *F7B659FE10CA9FAC576D358A16CC1BC646762FB2 |
```

```
+-----------------------------------------+
1 row in set (0.00 sec)
```

An execution run to install MySQL is as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/mysql.yml
-K
SUDO password:
PLAY [Install MySQL database server] ************************
TASK [setup] ************************************************
ok: [ubuntu]
TASK [Update the software package repository] ***************
changed: [ubuntu]
TASK [Install MySQL] ****************************************
changed: [ubuntu] => (item=[u'mysql-server', u'mysql-client', u'python-
mysqldb'])
TASK [Start the server] *************************************
ok: [ubuntu]
TASK [wait_for] *********************************************
ok: [ubuntu]
TASK [mysql_user] *******************************************
ok: [ubuntu]
PLAY RECAP **************************************************
ubuntu                     : ok=6    changed=2    unreachable=0    failed=0
```

 **Note:** The default MySQL root password is empty. You should change it after installation.

Installing PHP

PHP is a server-side programming language and stands for PHP: Hypertext Preprocessor (a recursive acronym). Although we have used PHP5 in this example, it is recommended that you use the latest PHP for security reasons. The Ansible playbook for installing PHP is given below:

```
---
- name: Install PHP
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
```

```
  tags: [web]
  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes
    - name: Install PHP
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - php5
        - php5-mysql
```

We update the software repository and install PHP5. An execution output of the Ansible playbook is shown below:

```
$  ansible-playbook -i inventory/kvm/inventory playbooks/configuration/php.yml
-K
SUDO password:
PLAY [Install PHP] ************************************************
TASK [setup] ******************************************************
ok: [ubuntu]
TASK [Update the software package repository] ***************
changed: [ubuntu]
TASK [Install PHP] ************************************************
changed: [ubuntu] => (item=[u'php5', u'php5-mysql'])
PLAY RECAP ********************************************************
ubuntu                     : ok=3    changed=2    unreachable=0    failed=0
```

Installing WordPress

As a final step, we will download, install and set up WordPress. The complete playbook is as follows:

```
---
- name: Setup Wordpress
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [database]
  vars:
```

```
    wordpress_file: "/home/{{ ansible_user }}/Downloads/wordpress-latest.zip"
    wordpress_dest: "/var/www/html"
  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes
    - name: Create a database for wordpress
      mysql_db:
        name: wordpress
        state: present
    - name: Create downloads directory
      file:
        path: "/home/{{ ansible_user }}/Downloads"
        state: directory
    - name: Create target directory
      file:
        path: "{{ wordpress_dest }}/wordpress"
        state: directory
    - name: Download latest wordpress
      get_url:
        url: https://wordpress.org/latest.zip
        dest: "{{ wordpress_file }}"
    - name: Extract to /var/www/html
      unarchive:
        src: "{{ wordpress_file }}"
        dest: "{{ wordpress_dest}}"
        remote_src: True
    - name: Copy wp-config-sample.php to wp-config.php
      command: cp "{{ wordpress_dest }}/wordpress/wp-config-sample.php" "{{
wordpress_dest }}/wordpress/wp-config.php"
    - name: Update database credentials in the file
      replace:
        dest: "{{ wordpress_dest }}/wordpress/wp-config.php"
        regexp: "{{ item.regexp }}"
        replace: "{{ item.replace }}"
      with_items:
        - { regexp: 'database_name_here', replace: 'wordpress' }
        - { regexp: 'username_here', replace: 'guest' }
        - { regexp: 'password_here', replace: 'osfy'}
    - name: Restart apache2 server
      service:
        name: apache2
        state: restarted
```

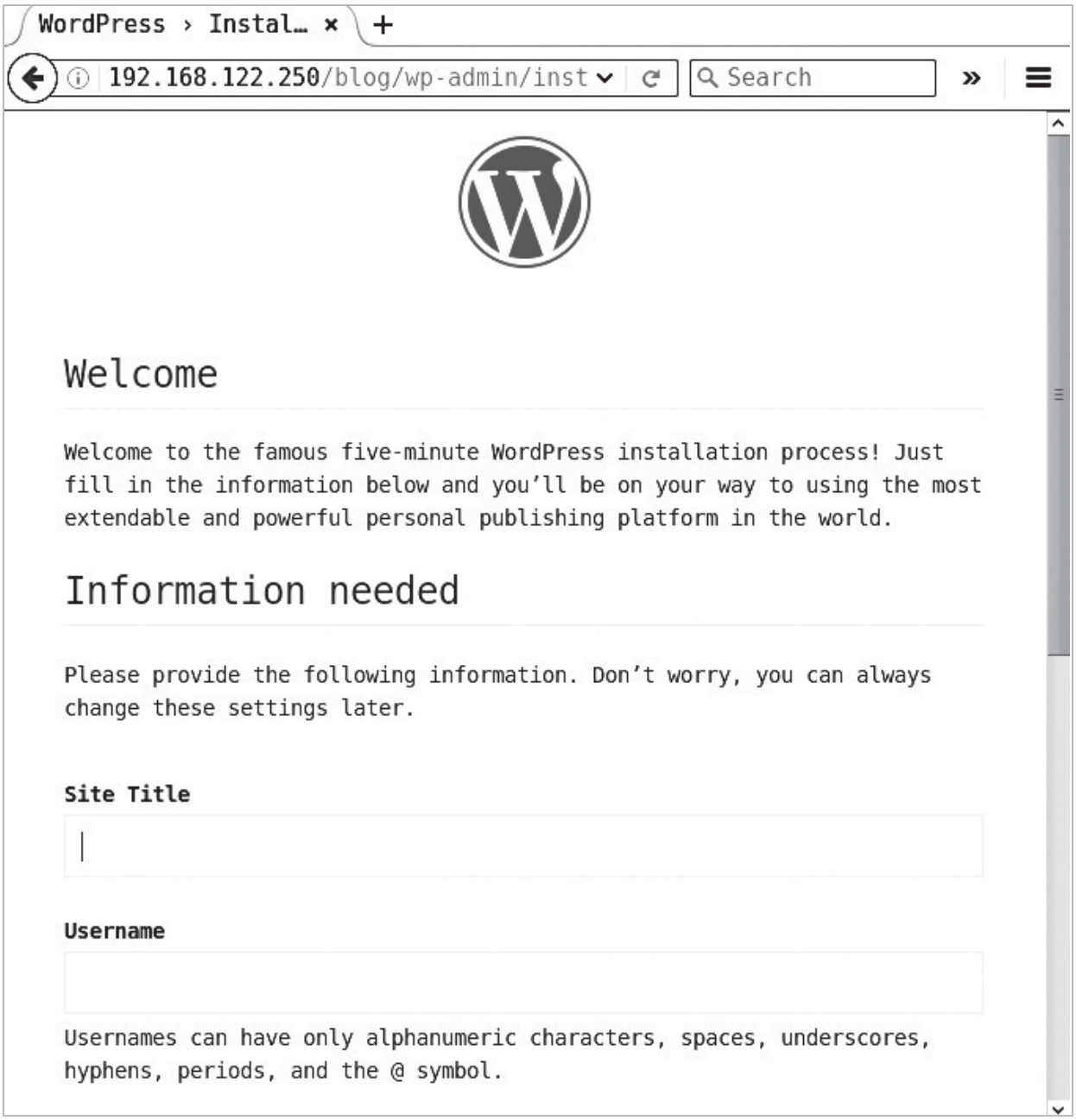


Figure 2: WordPress install page

We create variables to store the downloaded file for WordPress, and the target installation path. After updating the software repository, a database is created for the WordPress application. The download and target directories are created on the guest system, before actually downloading the latest WordPress sources. A configuration file is then created, and the database settings are updated. Although we explicitly specify the password here, the recommended practice is to store the encrypted passwords in an Ansible Vault file, and reference the same in the playbook. In future articles, I will demonstrate this use case. After completing the configuration, the Web server is restarted. An execution run of the playbook is shown below:

```
$  ansible-playbook -i inventory/kvm/inventory playbooks/configuration/
```

```
wordpress.yml -K
SUDO password:
PLAY [Setup Wordpress] ***************************************************
*
TASK [setup] *************************************************************
*
ok: [ubuntu]
TASK [Update the software package repository] ****************************
*
changed: [ubuntu]
TASK [Create a database for wordpress] ***********************************
*
changed: [ubuntu]
TASK [Create downloads directory] ****************************

ok: [ubuntu]
TASK [Create target directory] ****************************
changed: [ubuntu]
TASK [Download latest wordpress] ****************************
ok: [ubuntu]
TASK [Extract to /var/www/html] ****************************

changed: [ubuntu]
TASK [Copy wp-config-sample.php to wp-config.php] ******************************

changed: [ubuntu]
TASK [Update database credentials in the file] ***************
changed: [ubuntu] => (item={u'regexp': u'database_name_here', u'replace':
u'wordpress'})
changed: [ubuntu] => (item={u'regexp': u'username_here', u'replace': u'guest'})
changed: [ubuntu] => (item={u'regexp': u'password_here', u'replace': u'osfy'})
TASK [Restart apache2 server] ******************************

changed: [ubuntu]
PLAY RECAP ***************************************************
ubuntu                     : ok=10   changed=7    unreachable=0    failed=0
```

If you open the URL *http://192.168.122.250/wordpress* in a browser on the host system, you will see a screenshot as shown in Figure 2.

You can now proceed to complete the installation process from the browser. It is recommended that you follow the security best practices as recommended by the WordPress and PHP projects to harden this deployment.

Writing clean-up playbooks

It is essential to write clean-up playbooks to revert whatever changes you have made, so that you can roll back the system if things fail. Uninstalling should be done in the reverse order. For example, remove WordPress first, followed by PHP, MySQL and Apache.

The removal of WordPress could depend on your data retention policy. You might want to back up your PHP files, or you may decide to discard them. You might also want to retain the database. A complete removal of WordPress and the LAMP stack in the playbooks/admin folder is provided below for reference:

```
---
- name: Uninstall Wordpress
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [web]
  vars:
    wordpress_dest: "/var/www/html"
  tasks:
    - name: Delete wordpress folder
      file:
        path: "{{ wordpress_dest }}/wordpress"
        state: absent
    - name: Drop database
      mysql_db:
        name: wordpress
        state: absent
---
- name: Uninstall PHP
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [web]
  tasks:
    - name: Uninstall PHP packages
      package:
        name: "{{ item }}"
        state: absent
      with_items:
        - php5-mysql
```

```
        - php5
---
- name: Uninstall MySQL
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [database]
  tasks:
    - name: Stop the database server
      service:
        name: mysql
        state: stopped
    - name: Uninstall MySQL packages
      package:
        name: “{{ item }}”
        state: absent
      with_items:
        - python-mysqldb
        - mysql-client
        - mysql-server
---
- name: Uninstall Apache web server
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [server]
s  tasks:
    - name: Stop the web server
      service:
        name: apache2
        state: stopped
    - name: Uninstall apache2
      package:
        name: “{{ item }}”
        state: absent
      with_items:
        - apache2
```

The entire suite of playbooks is also available in my GitHub project (*https://github.com/shakthimaan/introduction-to-ansible*) for your reference.

Chapter 4

Using Ansible to Deploy Cacti for Monitoring

In this third article in the DevOps series, we will install and set up Cacti, a free and open source Web-based network monitoring and graphing tool, using Ansible.

Cacti is written in PHP and uses the MySQL database as a backend. It uses the RRDtool (Round-Robin Database tool) to handle time series data and has built-in SNMP support. Cacti has been released under the GNU General Public License.

Setting up Cacti

We will use a CentOS 6.8 virtual machine (VM) running on KVM to set up Cacti. Just for this demonstration, we will disable SELinux. You will need to set the following in */etc/selinux/config* and reboot the VM:

```
SELINUX=disabled
```

When used in production, it is essential that you enable SELinux. You should then test for Internet connectivity from within the VM.

The Ansible version used on the host Parabola GNU/Linux-libre x86_64 is 2.2.1.0. The *ansible/inventory/kvm/* directory structure is shown below:

```
ansible/inventory/kvm/inventory
ansible/inventory/kvm/group_vars/all/all.yml
```

The IP address of the guest CentOS 6.8 VM is provided in the inventory file as shown below:

```
centos ansible_host=192.168.122.98 ansible_connection=ssh ansible_user=root
```

```
ansible_password=password
```

Add an entry for 'centos' in the */etc/hosts* file as indicated below:

```
192.168.122.98 centos
```

The contents of the *all.yml* for use with the playbook are as follows:

```
---
mysql_cacti_password_hash: "{{ vault_mysql_cacti_password_hash }}"
```

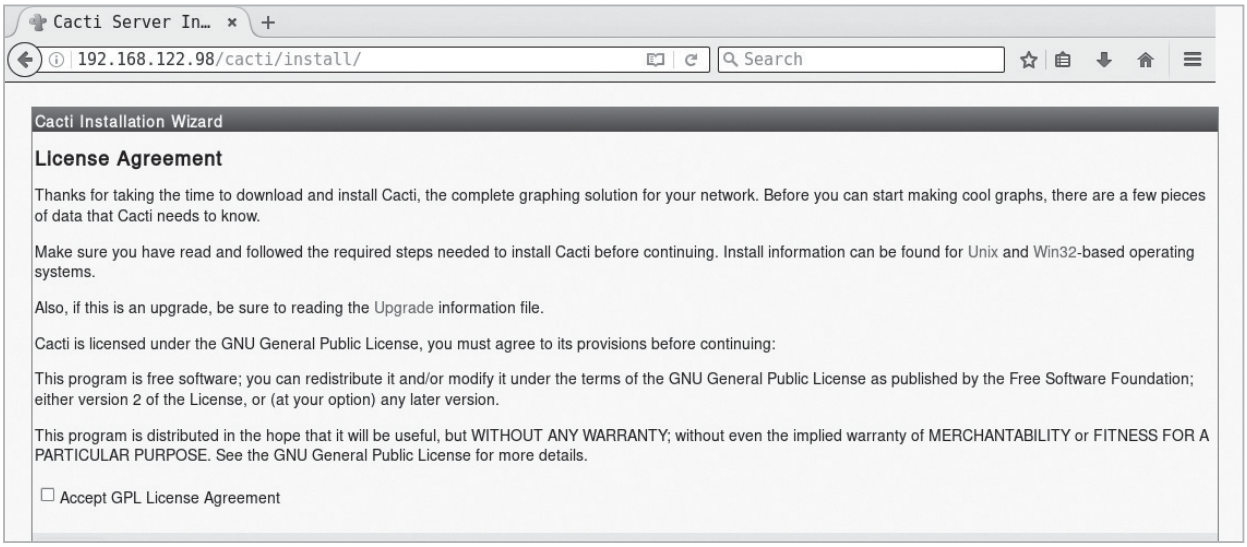


Figure 1: License agreement

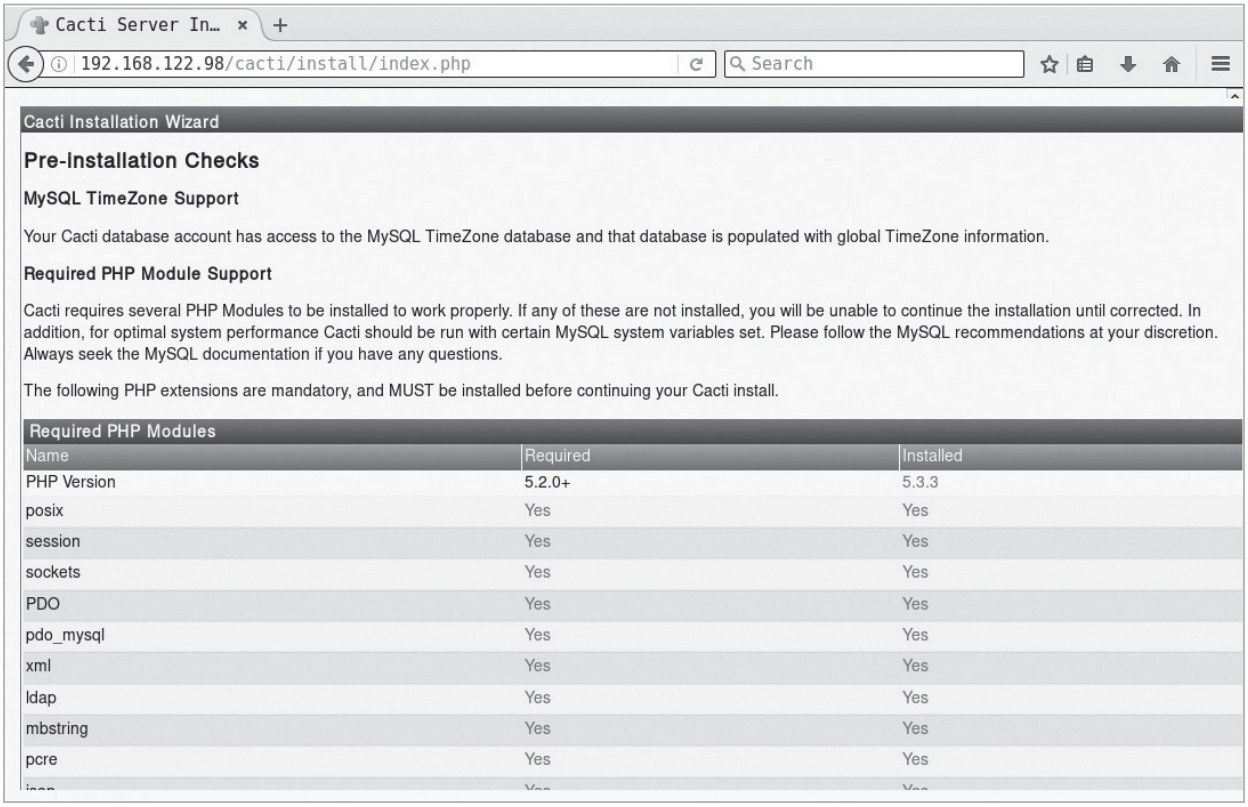


Figure 2: Pre-installation checks

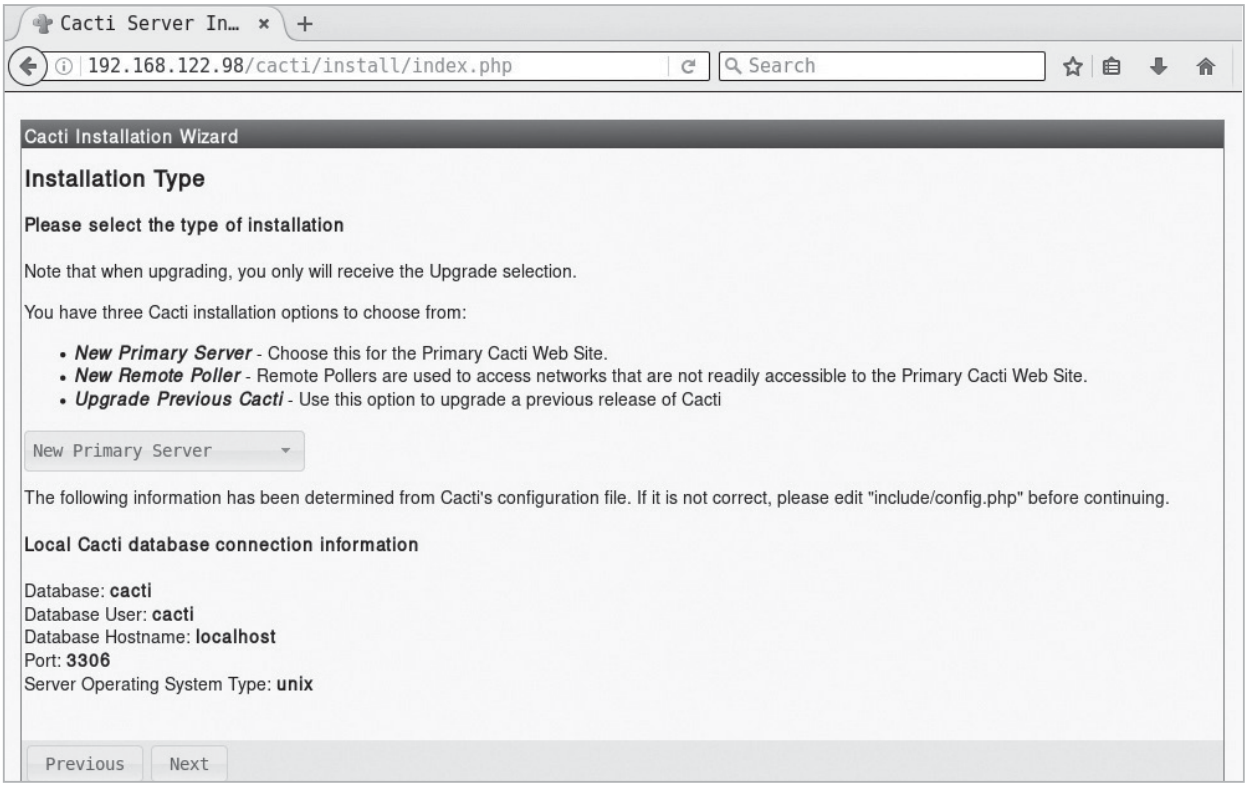


Figure 3: Installation type

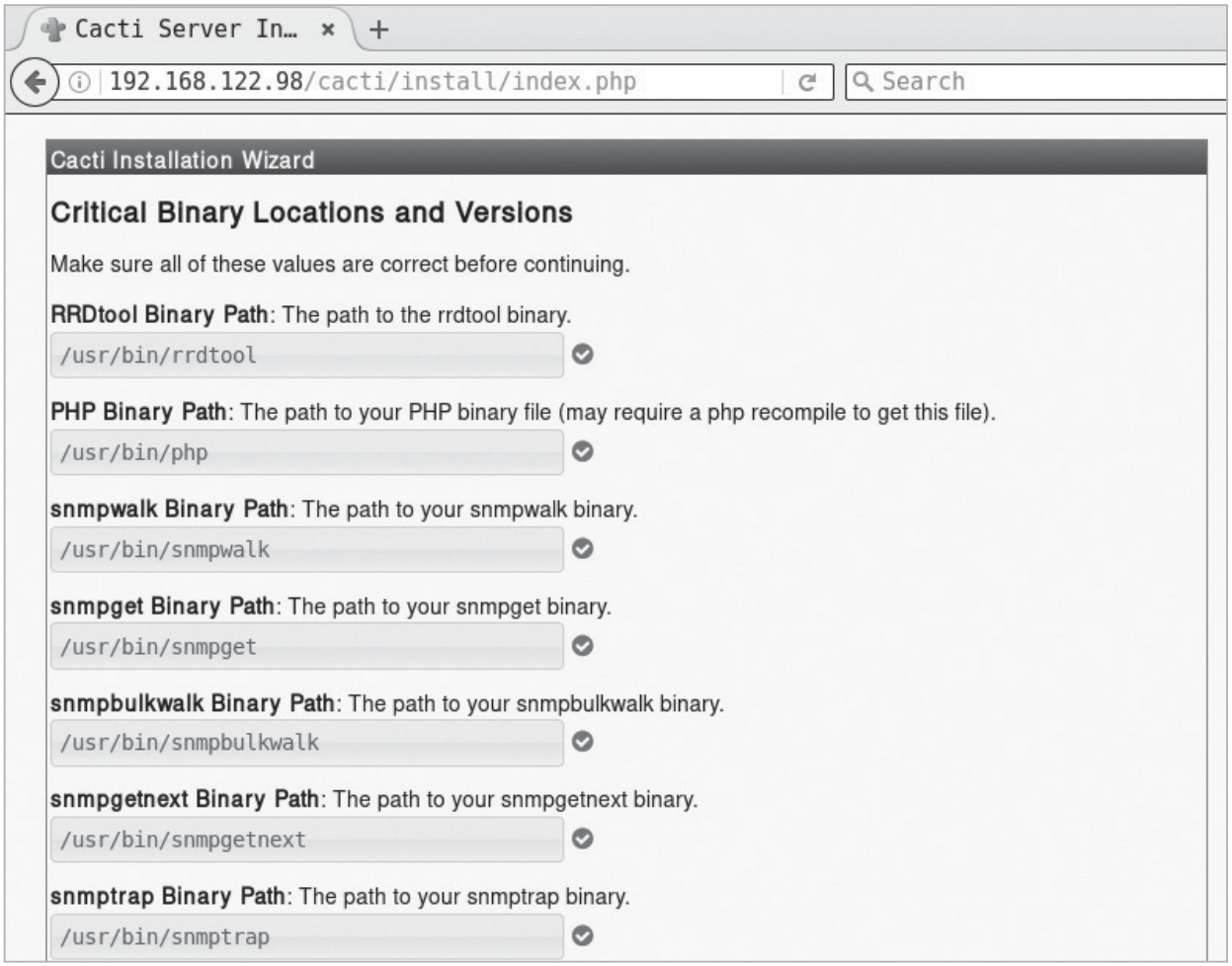


Figure 4: Binary location and version

```
mysql_username: "{{ vault_mysql_user }}"
mysql_password: "{{ vault_mysql_password }}"
```

The *Cacti.yml* playbook is located in the *ansible/playbooks/configuration* folder.

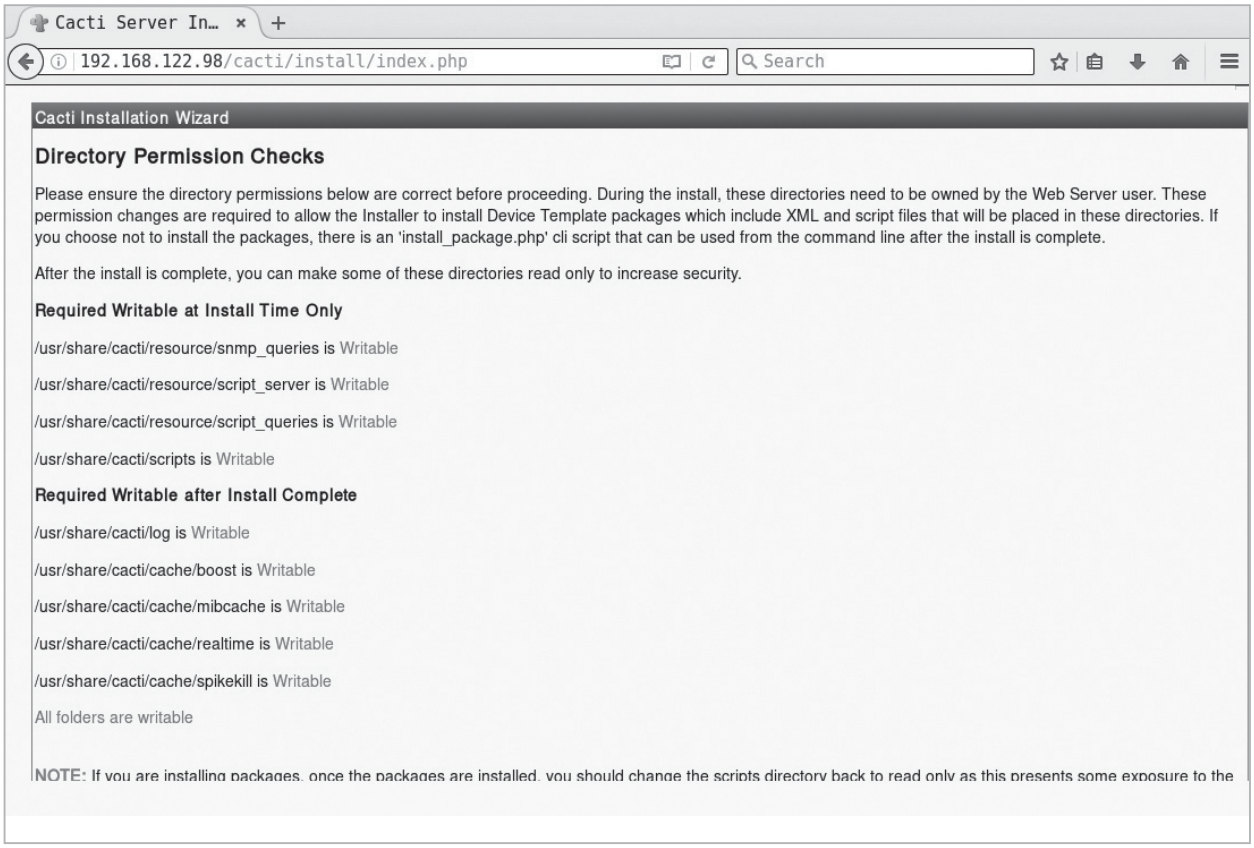


Figure 5: Directory permission checks

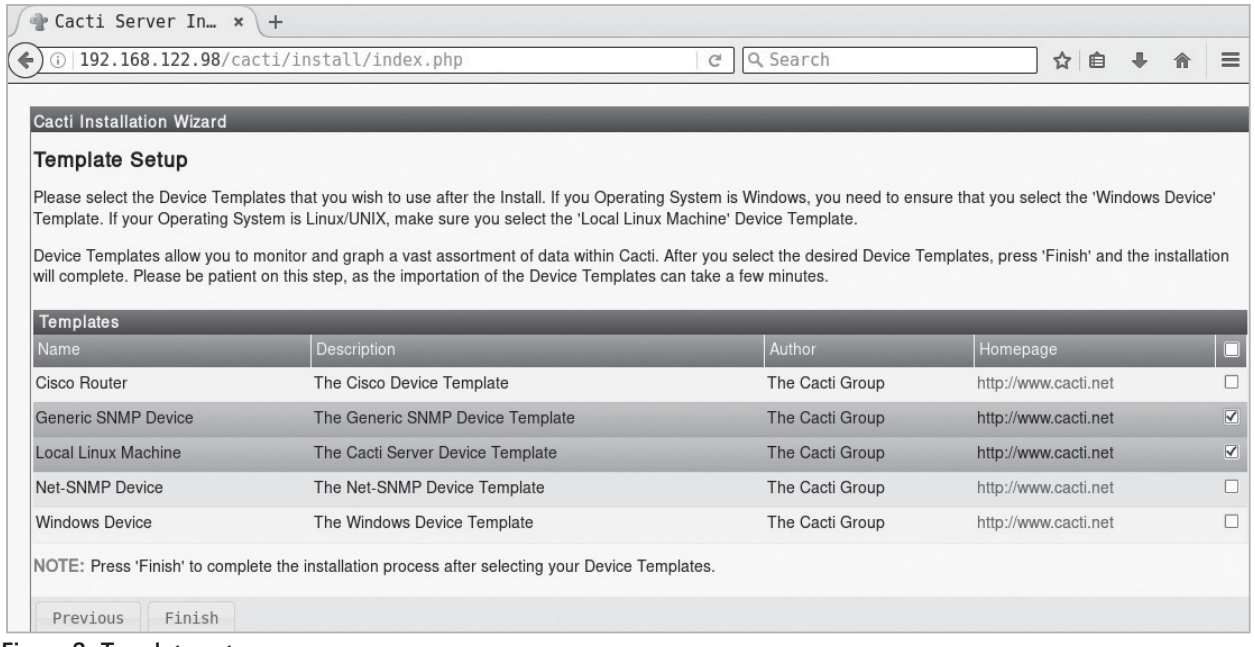


Figure 6: Template set-up

Vault

Ansible provides the Vault feature, which allows you to store sensitive information like passwords in encrypted files. You can set the EDITOR environment variable to the text editor of your choice, as shown below:

```
$ export EDITOR=nano
```

In order to store our MySQL database credentials, we will create a *vault.yml* file as indicated below:

```
$ ansible-vault create inventory/kvm/group_vars/all/vault.yml
```

Provide a password when prompted, following which, the Nano text editor will open. You can enter the following credentials and save the file:

```
---
vault_mysql_cacti_password_hash: "*528573A4E6FE4F3E8B455F2F060EB6F63ECBECAA"

vault_mysql_user: "cacti"
vault_mysql_password: "cacti123"
```

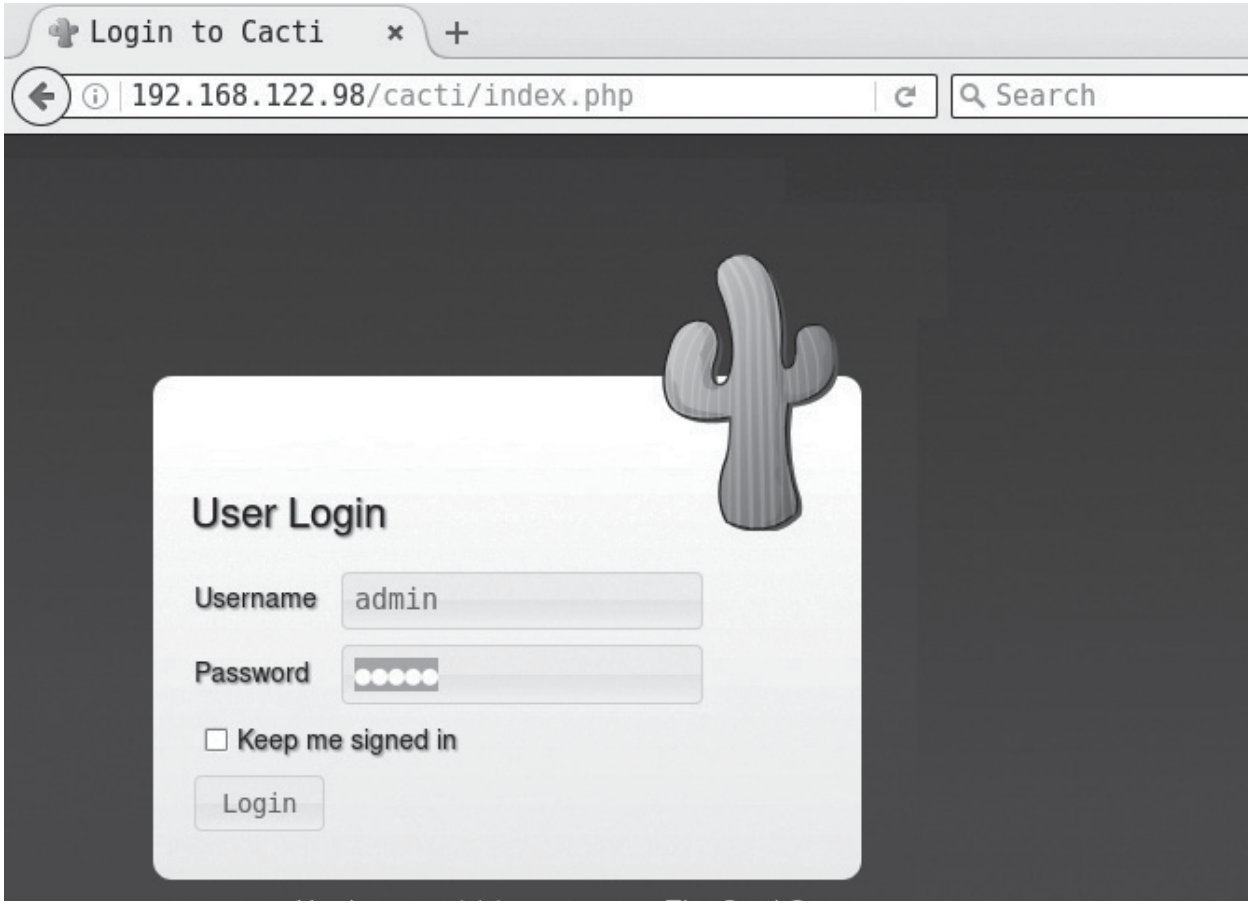


Figure 7: User login

You can edit the same file, if you wish, using the following command:

```
$ ansible-vault edit inventory/kvm/group_vars/all/vault.yml
```

It will prompt you for a password, and on successful authentication, your text editor will open with the decrypted file contents for editing.

Apache

Cacti has many dependency packages, and the first software that we will install is the Apache HTTP server.

```
---
- name: Install web server
  hosts: centos
  gather_facts: true
  tags: [httpd]

  tasks:
    - name: Update the software package repository
      yum:
        name: '*'
        update_cache: yes

    - name: Install HTTP packages
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - wget
        - nano
        - httpd
        - httpd-devel

    - name: Start the httpd server
      service:
        name: httpd
        state: started

    - wait_for:
        port: 80
```

A 'yum update' is first performed to sync with the package repositories. The *httpd* Web server and a few other packages are then installed. The

server is started, and the Ansible playbook waits for the server to listen on port 80.

MySQL and PHP

The MySQL, PHP and RRDTool packages are then installed, following which the SNMP and MySQL servers are started as shown below:

```
- name: Install MySQL, PHP packages
  hosts: centos
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [database-web]

  tasks:
    - name: Install database/web packages
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - mysql
        - mysql-server
        - MySQL-python
        - php-mysql
        - php-pear
        - php-common
        - php-gd
        - php-devel
        - php
        - php-mbstring
        - php-cli
        - php-process
        - php-snmp
        - net-snmp-utils
        - net-snmp-libs
        - rrdtool

    - name: Start snmpd server
      service:
        name: snmpd
        state: started

    - name: Start mysqld server
```

```
    service:
      name: mysqld
      state: started

  - wait_for:
      port: 3306
```

Cacti

Cacti is available in the EPEL repository for CentOS. The GPG key for the CentOS repositories is enabled before installing the EPEL repository. A'yum update' is performed and the Cacti package is installed. A *Cacti* user is then created in the MySQL database.

```
- name: Install Cacti
  hosts: centos
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [cacti]

  tasks:
    - name: Import EPEL GPG key
      rpm_key:
        key: http://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-6
        state: present

    - name: Add YUM repo
      yum_repository:
        name: epel
        description: EPEL YUM repo
        baseurl: https://dl.fedoraproject.org/pub/epel/$releasever/$basearch/
        gpgcheck: yes

    - name: Update the software package repository
      yum:
        name: '*'
        update_cache: yes

    - name: Install cacti
      package:
        name: "{{ item }}"
        state: latest
      with_items:
```

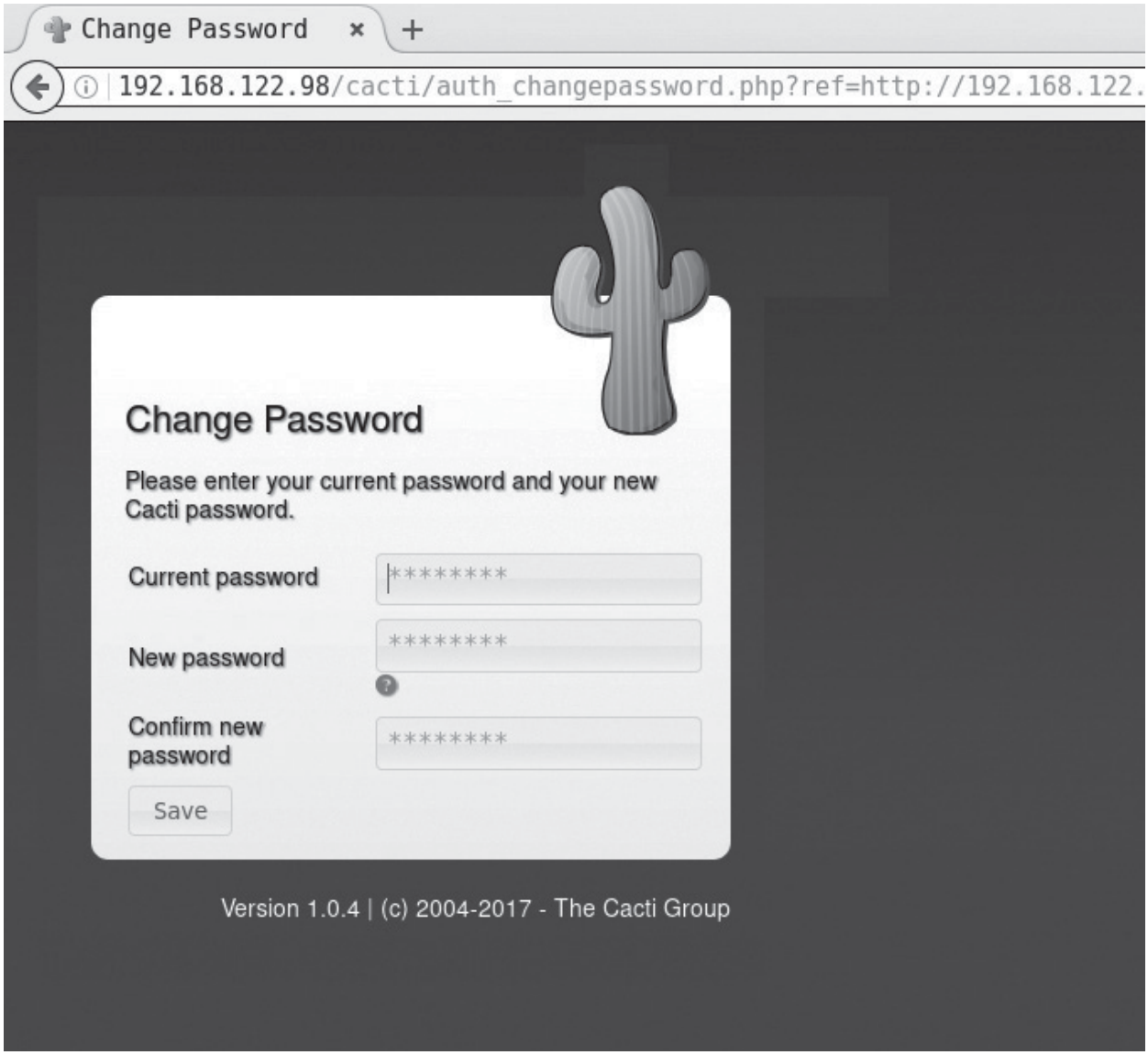


Figure 8: Changing the password

```
      - cacti
```

```
  - name: Create cacti database user
    mysql_user:
      name: cacti
      password: "{{ mysql_cacti_password_hash }}"
      encrypted: yes
      priv: '*.*:ALL,GRANT'
      state: present
```

Fixing a bug

The time zone data is missing in this MySQL version (5.1.73-8). In order to resolve this bug, the *mysql_test_data_timezone.sql* file needs to be imported and the 'cacti' user needs to be given the SELECT privilege to do this.

```
- name: For bug https://github.com/Cacti/cacti/issues/242
  hosts: centos
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [bug]

  tasks:
    - name: Import mysql_test_data_timezone.sql
      mysql_db:
        state: import
        name: mysql
        target: /usr/share/mysql/mysql_test_data_timezone.sql

    - name: Grant privileges
      mysql_user:
        name: cacti
        append_privs: true
        priv: 'mysql.time_zone_name:SELECT'
        state: present
```

It is a good practice to have a separate playbook for such exceptional cases. In future, when you upgrade to newer versions that have bug fixes, you can simply skip this step.

Configuration

The last step involves configuring Cacti.

```
- name: Configuration
  hosts: centos
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [config]

  tasks:
    - name: Create a database for cacti
      mysql_db:
        name: cacti
        state: present

    - name: Import cacti.sql
```

```
      mysql_db:
        state: import
        name: cacti
        target: /usr/share/doc/cacti-1.0.4/cacti.sql

    - name: Update database credentials in config file
      lineinfile:
        dest: /etc/cacti/db.php
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
      with_items:
        - { regexp: '^\$database_username', line: "$database_username = '{{
mysql_username }}';" }
        - { regexp: '^\$database_password', line: "$database_password = '{{
mysql_password }}';" }

    - name: Allow port 80
      shell: iptables -I INPUT 5 -p tcp --dport 80 -m state --state
NEW,ESTABLISHED -j ACCEPT

    - name: Update access in cacti.conf for httpd
      replace:
        dest: /etc/httpd/conf.d/cacti.conf
        regexp: "{{ item.regexp }}"
        replace: "{{ item.replace }}"
      with_items:
        - { regexp: 'Require host localhost', replace: 'Require all granted' }
        - { regexp: 'Allow from localhost', replace: 'Allow from all' }

    - lineinfile:
        dest: /etc/cron.d/cacti
        regexp: '^#(.*)$'
        line: '\1'
        backrefs: yes

    - name: Start mysqld server
      service:
        name: mysqld
        state: restarted

    - wait_for:
        port: 3306
```

```
    - name: Start the httpd server
      service:
        name: httpd
        state: restarted

    - wait_for:
        port: 80
```

A database called 'cacti' is created for the application, and the *cacti.sql* file is imported into it. The database credentials are updated for the Cacti application. The firewall rules are then updated to allow incoming HTTP requests for port 80. The periodic *cron poller* is then enabled in */etc/cron.d/cacti*:

```
*/5 * * * *     cacti   /usr/bin/php /usr/share/cacti/poller.php > /dev/null
2>&1
```

The MySQL and HTTP servers are then restarted.

The result

The entire playbook can now be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/cacti.yml
--ask-vault-pass
```

It will prompt you for the Vault password, following which all the

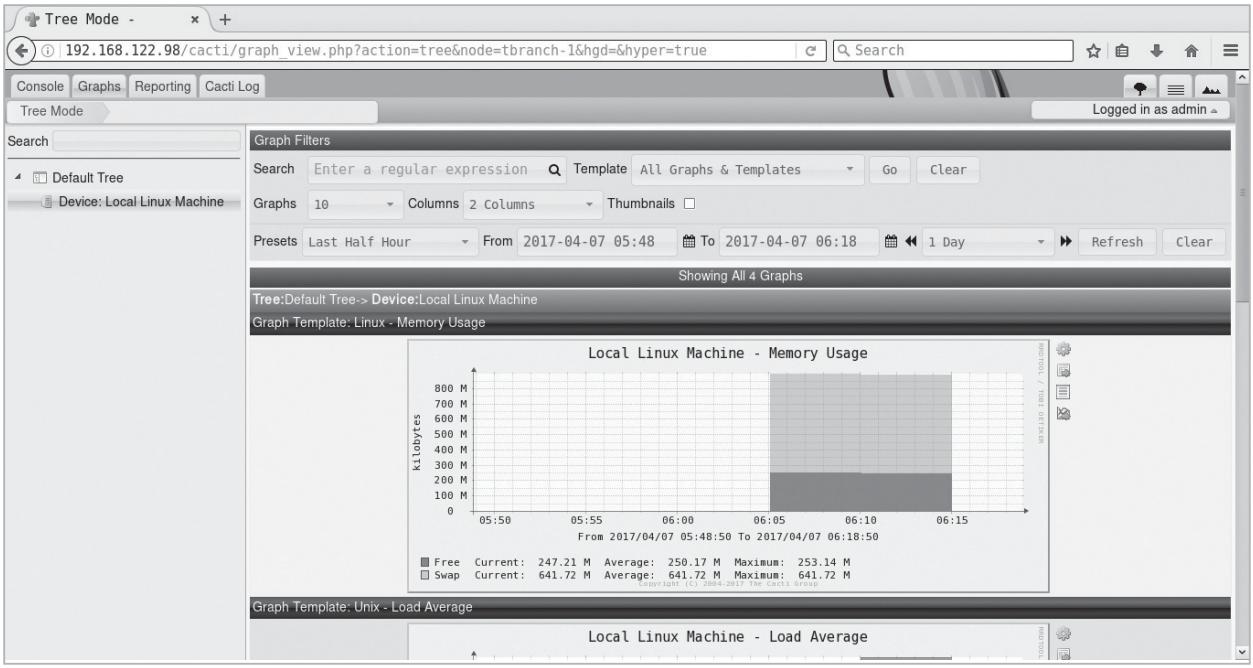


Figure 9: Cacti Web UI

playbooks will be completed. You can then open *http://192.168.122.98/cacti* to accept the GNU General Public License agreement. After you agree to the terms of the licence, click 'Next'. The Cacti installation wizard shows the pre-installation checks, which should not have any errors. This is followed by the selection of the installation type, binary location, version, and the directory permission checks. You can then decide on the templates you would like to set up, following which a user login is provided. The default user name and password is 'admin:admin' and you will be immediately prompted to change the password after logging in. You can then proceed to log in to the Cacti dashboard. Figures 1 to 8 give the screenshots of the Cacti Web UI installation for reference.

A screenshot of Cacti graphing for memory usage is shown in Figure 9.

Chapter 5

Ansible Deployment of RabbitMQ

RabbitMQ, which is free and open source, is the world's most widely deployed message broker. It is used by several big companies like Ford, Instagram, Cisco, etc. Being easy to deploy, it can be used in situ or on the cloud.

In this fourth article in the DevOps series, we will learn to install RabbitMQ using Ansible. RabbitMQ is a free and open source message broker system that supports a number of protocols such as the Advanced Message Queuing Protocol (AMQP), Streaming Text Oriented Messaging Protocol (STOMP) and Message Queue Telemetry Transport (MQTT). The software has support for a large number of client libraries for different programming languages. RabbitMQ is written using the Erlang programming language and is released under the Mozilla Public License.

Setting it up

A CentOS 6.8 virtual machine (VM) running on KVM is used for the installation. Do make sure that the VM has access to the Internet. The Ansible version used on the host (Parabola GNU/Linux-libre x86_64) is 2.2.1.0. The ansible/folder contains the following files:

```
ansible/inventory/kvm/inventory
ansible/playbooks/configuration/rabbitmq.yml
ansible/playbooks/admin/uninstall-rabbitmq.yml
```

The IP address of the guest CentOS 6.8 VM is added to the *inventory* file as shown below:

```
rabbitmq ansible_host=192.168.122.161 ansible_connection=ssh ansible_user=root
ansible_password=password
```

Also, add an entry for the *rabbitmq* host in the */etc/hosts* file as indicated below:

```
192.168.122.161 rabbitmq
```

Installation

RabbitMQ requires the Erlang environment, and uses the Open Telecom Platform (OTP) framework. There are multiple sources for installing Erlang — the EPEL repository, Erlang Solutions, and the zero-dependency Erlang provided by RabbitMQ. In this article, we will use the EPEL repository for installing Erlang.

```
---
- name: Install RabbitMQ server
  hosts: rabbitmq
  gather_facts: true
  tags: [server]
  tasks:
    - name: Import EPEL GPG key
      rpm_key:
        key: http://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-6
        state: present

    - name: Add YUM repo
      yum_repository:
        name: epel
        description: EPEL YUM repo
        baseurl: https://dl.fedoraproject.org/pub/epel/$releasever/$basearch/
        gpgcheck: yes

    - name: Update the software package repository
      yum:
        name: '*'
        update_cache: yes

    - name: Install RabbitMQ server
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - rabbitmq-server

    - name: Start the RabbitMQ server
```

```
    service:
      name: rabbitmq-server
      state: started
  - wait_for:
      port: 5672
```

Figure 1: RabbitMQ login

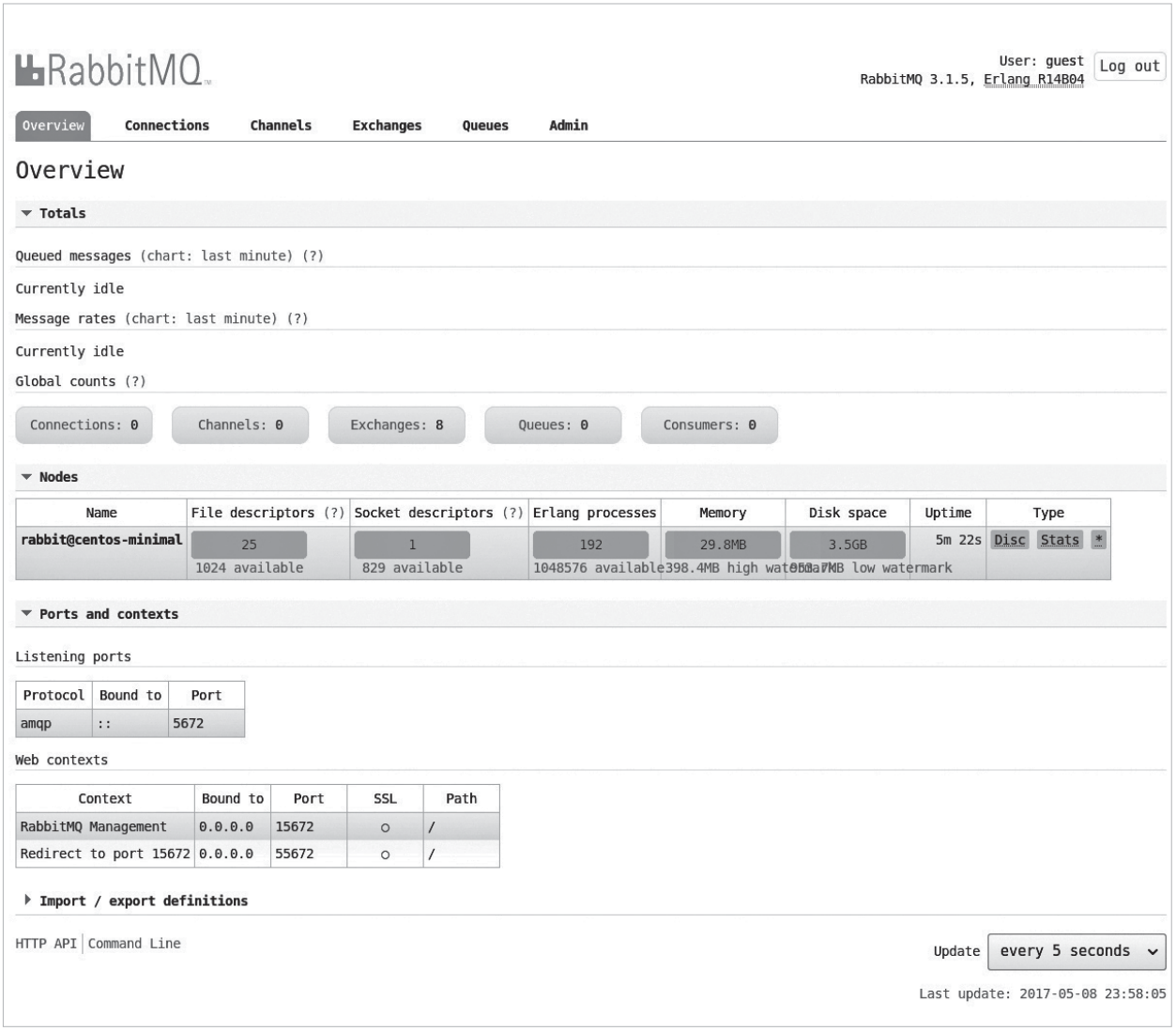


Figure 2: RabbitMQ overview

After importing the EPEL GPG key and adding the EPEL repository to the system, the *yum update* command is executed. The RabbitMQ server and its dependencies are then installed. We wait for the RabbitMQ server to start and listen on port 5672. The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/rabbitmq.
yml --tags “server”
Dashboard
```

The RabbitMQ management user interface (UI) is available through plugins.

```
- name: Start RabbitMQ Management UI
  hosts: rabbitmq
  gather_facts: true
  tags: [ui]

  tasks:
    - name: Start management UI
      command: /usr/lib/rabbitmq/bin/rabbitmq-plugins enable rabbitmq_
management

    - name: Restart RabbitMQ server
      service:
        name: rabbitmq-server
        state: restarted

    - wait_for:
        port: 15672

    - name: Allow port 15672
      shell: iptables -I INPUT 5 -p tcp --dport 15672 -m state --state
NEW,ESTABLISHED -j ACCEPT
```

After enabling the management plugin, the server needs to be restarted. Since we are running it inside the VM, we need to allow the management user interface (UI) port 15672 through the firewall. The playbook invocation to set up the management UI is given below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/rabbitmq.
yml --tags “ui”
```

The default user name and password for the dashboard are 'guest:guest'. From your host system, you can start a browser and open *http://192.168.122.161:15672* to view the login page as shown in Figure 1. The default 'Overview' page is shown in Figure 2.

Ruby

We will use a Ruby client example to demonstrate that our installation of RabbitMQ is working fine. The Ruby Version Manager (RVM) will be used to install Ruby as shown below:

```
- name: Ruby client
  hosts: rabbitmq
  gather_facts: true
  tags: [ruby]

  tasks:
    - name: Import key
      command: gpg2 --keyserver hkp://keys.gnupg.net --recv-keys
409B6B1796C275462A1703113804BB82D39DC0E3

    - name: Install RVM
      shell: curl -sSL https://get.rvm.io | bash -s stable

    - name: Install Ruby
      shell: source /etc/profile.d/rvm.sh && rvm install ruby-2.2.6

    - name: Set default Ruby
      command: rvm alias create default ruby-2.2.6

    - name: Install bunny client
      shell: gem install bunny --version ">= 2.6.4"
```

After importing the required GPG keys, RVM and Ruby 2.2.6 are installed on the CentOS 6.8 VM. The *bunny* Ruby client for RabbitMQ is then installed. The Ansible playbook to set up Ruby is given below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/rabbitmq.
yml --tags "ruby"
```

We shall create a 'temperature' queue to send the values in Celsius. The *consumer.rb* code to receive the values from the queue is given below:

```
#!/usr/bin/env ruby

require “bunny”

conn = Bunny.new(:automatically_recover => false)
conn.start

chan  = conn.create_channel
queue = chan.queue(“temperature”)

begin
  puts “ ... waiting. CTRL+C to exit”
  queue.subscribe(:block => true) do |info, properties, body|
    puts “ Received #{body}”
  end
rescue Interrupt => _
  conn.close

  exit(0)
end
```

The *producer.rb* code to send a sample of five values in degree Celsius is as follows:

```
#!/usr/bin/env ruby

require “bunny”

conn = Bunny.new(:automatically_recover => false)
conn.start

chan   = conn.create_channel
queue   = chan.queue(“temperature”)

values = [“33.5”, “35.2”, “36.7”, “37.0”, “36.4”]

values.each do |v|
  chan.default_exchange.publish(v, :routing_key => queue.name)
end
puts “Sent five temperature values.”

conn.close
```

As soon as you start the consumer, you will get the following output:

```
$ ruby consumer.rb
 ... waiting. CTRL+C to exit
```

You can then run the *producer.rb* script that writes the values to the queue:

```
$ ruby producer.rb
```

```
Sent five temperature values.
```

The received values at the consumer side are printed out as shown below:

```
$ ruby consumer.rb
```

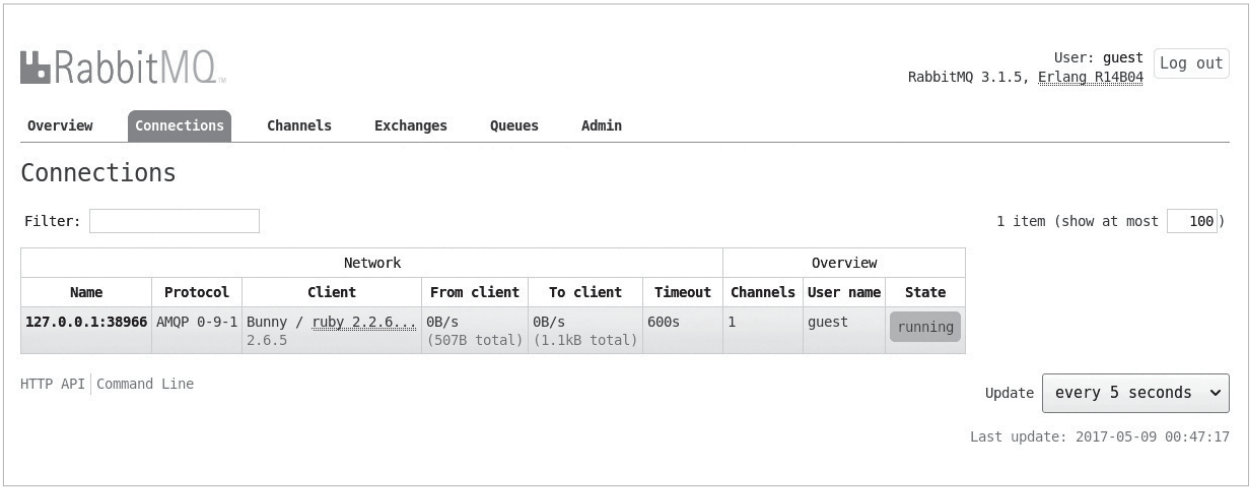


Figure 3: RabbitMQ connections

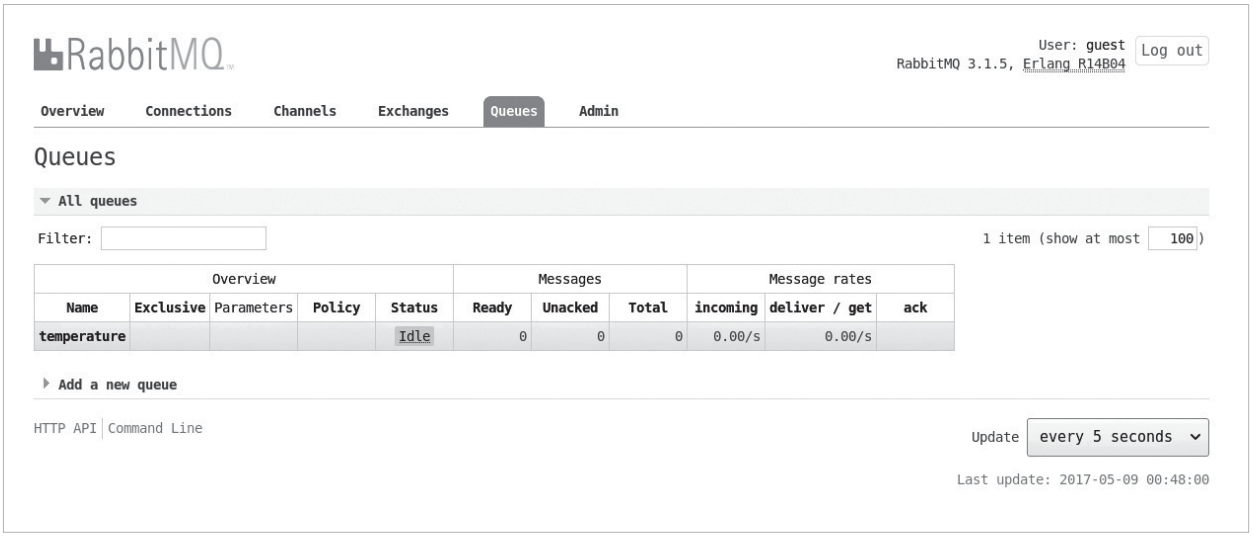


Figure 4: RabbitMQ queues

```
... waiting. CTRL+C to exit
Received 33.5
Received 35.2
Received 36.7
Received 37.0
Received 36.4
```

We can observe the available connections and the created queue in the management user interface as shown in Figure 3 and Figure 4, respectively.

Uninstall

It is good to have an uninstall script to remove the RabbitMQ server for administrative purposes. The Ansible playbook for the same is available in the *playbooks/admin* folder and is shown below:

```
---
- name: Uninstall RabbitMQ server
  hosts: rabbitmq
  gather_facts: true
  tags: [remove]

  tasks:
    - name: Stop the RabbitMQ server
      service:
        name: rabbitmq-server
        state: stopped

    - name: Uninstall rabbitmq
      package:
        name: "{{ item }}"
        state: absent
      with_items:
        - rabbitmq-server
```

The script can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-
rabbitmq.yml
```

You are encouraged to read the detailed documentation at *https://www.rabbitmq.com/documentation.html* to know more about the usage, configuration, client libraries and plugins available for RabbitMQ.

Chapter 6

Deploying Graphite Using Ansible

In this fifth article in the DevOps series we will learn to install and set up Graphite using Ansible.

Graphite is a monitoring tool that was written by Chris Davis in 2006. It has been released under the Apache 2.0 licence and comprises three components:

1. Graphite-Web
2. Carbon
3. Whisper

Graphite-Web is a Django application and provides a dashboard for monitoring. Carbon is a server that listens to time-series data, while Whisper is a database library for storing the data.

Setting it up

A CentOS 6.8 virtual machine (VM) running on KVM is used for the installation. Please make sure that the VM has access to the Internet. The Ansible version used on the host (Parabola GNU/Linux-libre x86_64) is 2.2.1.0. The *ansible/* folder contains the following files:

```
ansible/inventory/kvm/inventory
ansible/playbooks/configuration/graphite.yml
ansible/playbooks/admin/uninstall-graphite.yml
```

The IP address of the guest CentOS 6.8 VM is added to the *inventory* file as shown below:

```
graphite ansible_host=192.168.122.120 ansible_connection=ssh ansible_user=root
ansible_password=password
```

Also, add an entry for the *graphite* host in the */etc/hosts* file as indicated below:

```
192.168.122.120 graphite
```

Graphite

The playbook to install the Graphite server is given below:

```
---
- name: Install Graphite software
  hosts: graphite
  gather_facts: true
  tags: [graphite]

  tasks:
    - name: Import EPEL GPG key
      rpm_key:
        key: http://dl.fedoraproject.org/pub/epel/RPM-GPG-KEY-EPEL-6
        state: present

    - name: Add YUM repo
      yum_repository:
        name: epel
        description: EPEL YUM repo
        baseurl: https://dl.fedoraproject.org/pub/epel/$releasever/$basearch/
        gpgcheck: yes

    - name: Update the software package repository
      yum:
        name: '*'
        update_cache: yes

    - name: Install Graphite server
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - graphite-web
```

We first import the keys for the Extra Packages for Enterprise Linux (EPEL) repository and update the software package list. The 'graphite-web' package is then installed using Yum. The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graphite.
yml --tags “graphite”
```

MySQL

A backend database is required by Graphite. By default, the SQLite3 database is used, but we will install and use MySQL as shown below:

```
- name: Install MySQL
  hosts: graphite
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [database]

  tasks:
    - name: Install database
      package:
        name: “{{ item }}”
        state: latest
      with_items:
        - mysql
        - mysql-server
        - MySQL-python
        - libselinux-python

    - name: Start mysqld server
      service:
        name: mysqld
        state: started

    - wait_for:
        port: 3306

    - name: Create graphite database user
      mysql_user:
        name: graphite
        password: graphite123
        priv: ‘*.*:ALL,GRANT’
        state: present

    - name: Create a database
      mysql_db:
```

```
        name: graphite
        state: present

    - name: Update database configuration
      blockinfile:
        path: /etc/graphite-web/local_settings.py
        block: |
          DATABASES = {
            'default': {
            'NAME': 'graphite',
            'ENGINE': 'django.db.backends.mysql',
            'USER': 'graphite',
            'PASSWORD': 'graphite123',
           }
          }

    - name: syncdb
      shell: /usr/lib/python2.6/site-packages/graphite/manage.py syncdb
--noinput

    - name: Allow port 80
      shell: iptables -I INPUT -p tcp --dport 80 -m state --state
NEW,ESTABLISHED -j ACCEPT

    - name:
      lineinfile:
        path: /etc/httpd/conf.d/graphite-web.conf
        insertafter: '           # Apache 2.2'
        line: '           Allow from all'

    - name: Start httpd server
      service:
        name: httpd
        state: started
```

As a first step, let's install the required MySQL dependency packages and the server itself. We can then start the server and wait for it to listen on port 3306. A graphite user and database is created for use with the Graphite Web application. For this example, the password is provided as plain text. In production, use an encrypted Ansible Vault password.

The database configuration file is then updated to use the MySQL credentials. Since Graphite is a Django application, the *manage.py* script with *syncdb* needs to be executed to create the necessary tables.

We then allow port 80 through the firewall in order to view the Graphite dashboard. The *graphite-web.conf* file is updated to allow read access, and the Apache Web server is started.

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graphite.
yml --tags "database"
```

Carbon and Whisper

The Carbon and Whisper Python bindings need to be installed before starting the *carbon-cache* script.

```
- name: Install Carbon and Whisper
  hosts: graphite
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [carbon]

  tasks:
    - name: Install carbon and whisper
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - python-carbon
        - python-whisper

    - name: Start carbon-cache
      shell: /etc/init.d/carbon-cache start
```

The above playbook is invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graphite.
yml --tags "carbon"
```

The Graphite dashboard

You can open *http://192.168.122.120* in the browser on the host to view the Graphite dashboard. A screenshot of the Graphite Web application is shown in Figure 1.

Uninstalling Graphite

An uninstall script to remove the Graphite server and its dependency

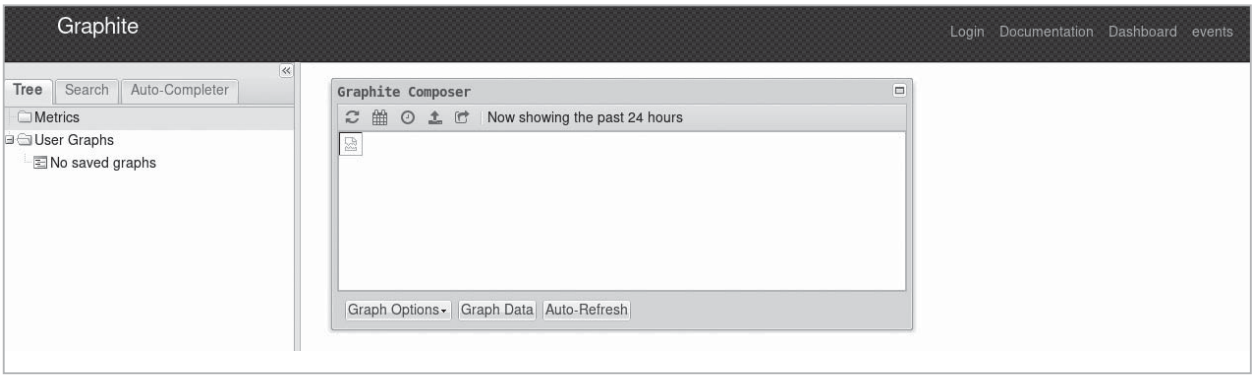


Figure 1: Graphite Web

packages is required for administration. The Ansible playbook for the same is available in the *playbooks/admin* folder and is given below:

```
---
- name: Uninstall Graphite and dependencies
  hosts: graphite
  gather_facts: true
  tags: [remove]

  tasks:
    - name: Stop the carbon-cache server
      shell: /etc/init.d/carbon-cache stop

    - name: Uninstall carbon and whisper
      package:
        name: "{{ item }}"
        state: absent
      with_items:
        - python-whisper
        - python-carbon

    - name: Stop httpd server
      service:
        name: httpd
        state: stopped

    - name: Stop mysqld server
      service:
        name: mysqld
        state: stopped

    - name: Uninstall database packages
      package:
```

```
        name: "{{ item }}"
        state: absent
      with_items:
        - libselinux-python
        - MySQL-python
        - mysql-server
        - mysql
        - graphite-web
```

The script can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-
graphite.yml
```

Chapter 7

Ansible Deployment of Jenkins

DevOps, formed by combining the words 'development' and 'operations', is a software development process. Its area of focus is the communication and collaboration between product management, software development and operations professionals. This article is the sixth in the series.

In this article, we will install Jenkins using Ansible and set up a continuous integration (CI) build for a project that uses Git. Jenkins is free and open source automation server software that is used to build, deploy and automate projects. It is written in Java and released under the MIT licence. There are a number of plugins available to integrate Jenkins with other tools such as version control systems, APIs and databases.

Setting it up

A CentOS 6.8 virtual machine (VM) running on KVM will be used for the installation. Internet access should be available from the guest machine. The Ansible version used on the host (Parabola GNU/Linux-libre x86_64) is 2.3.0.0. The *ansible/ folder* contains the following files:

```
ansible/inventory/kvm/inventory
ansible/playbooks/configuration/jenkins.yml
ansible/playbooks/admin/uninstall-jenkins.yml
```

The IP address of the guest CentOS 6.8 VM is added to the *inventory* file as shown below:

```
jenkins ansible_host=192.168.122.120 ansible_connection=ssh ansible_user=root
ansible_password=password
```

An entry for the Jenkins host is also added to the */etc/hosts* file as indicated below:

```
192.168.122.120 jenkins
```

Installation

The playbook to install the Jenkins server on the CentOS VM is given below:

```
---
- name: Install Jenkins software
  hosts: jenkins
  gather_facts: true
  become: yes
  become_method: sudo
  tags: [jenkins]

  tasks:
    - name: Update the software package repository
      yum:
        name: '*'
        update_cache: yes

    - name: Install dependencies
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - java-1.8.0-openjdk
        - git
        - texlive-latex
        - wget

    - name: Download jenkins repo
      command: wget -O /etc/yum.repos.d/jenkins.repo http://pkg.jenkins-ci.org/
redhat/jenkins.repo

    - name: Import Jenkins CI key
      rpm_key:
        key: http://pkg.jenkins-ci.org/redhat/jenkins-ci.org.key
        state: present

    - name: Install Jenkins
      package:
        name: "{{ item }}"
```

```
        state: latest
      with_items:
        - jenkins

    - name: Allow port 8080
      shell: iptables -I INPUT -p tcp --dport 8080 -m state --state
NEW,ESTABLISHED -j ACCEPT

    - name: Start the server
      service:
        name: jenkins
        state: started

    - wait_for:
        port: 8080
```

The playbook first updates the Yum repository and installs the Java OpenJDK software dependency required for Jenkins. The Git and Tex Live LaTeX packages are required to build our project *github.com/shakthimaan/di-git-ally-managing-love-letters*. We then download the Jenkins repository file and import the repository GPG key. The Jenkins server is then installed, port 8080 is allowed through the firewall, and the script waits for the server to listen on port 8080. The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/jenkins.
yml -vv
```

Figure 1: Unlocking Jenkins

Configuration

You can now open *http://192.168.122.120:8080* in the browser on the host to start configuring Jenkins. The Web page will prompt you to enter the initial administrator password from */var/lib/jenkins/secrets/initialAdminPassword* to proceed further. This is shown in Figure 1.

The second step is to install the plugins. For this demonstration, you can select the 'Install suggested plugins' option, and later install any of the plugins that you require. Figure 2 displays the selected option.

After you select the 'Install suggested plugins' option, the plugins will get installed as shown in Figure 3.

An admin user is required for managing Jenkins. After installing the plugins, a form is displayed for you to enter the user name, password, name and e-mail address of the administrator. A screenshot of this is shown in Figure 4.

Once the administrator credentials are stored, a 'Jenkins is ready!' page will be displayed, as depicted in Figure 5.

You can now click on the 'Start using Jenkins' button to open the default Jenkins dashboard shown in Figure 6.

An example of a new project

Let's now create a new build for the *github.com/shakthimaan/di-git-ally-managing-love-letters* project. Provide a name in the 'Enter an item name' text box and select the 'Freestyle project'. Figure 7 shows the screenshot for creating a new project.

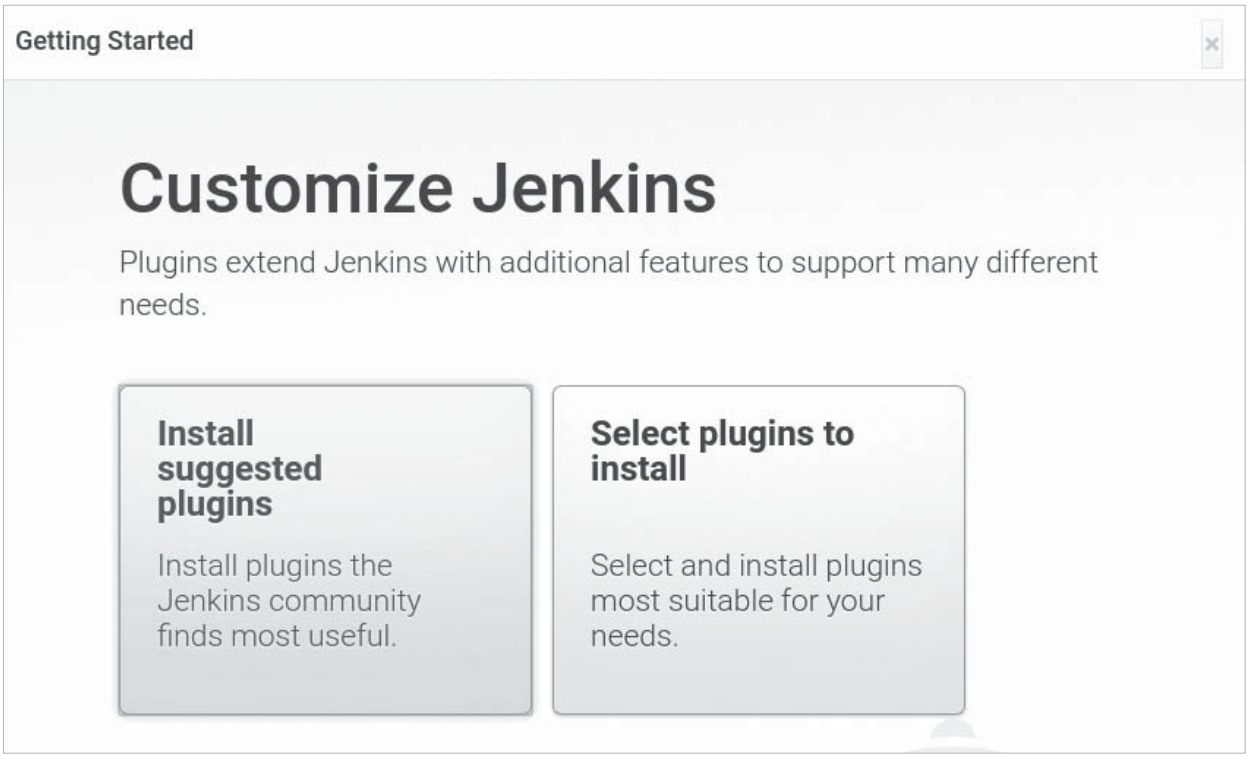


Figure 2: Customising Jenkins

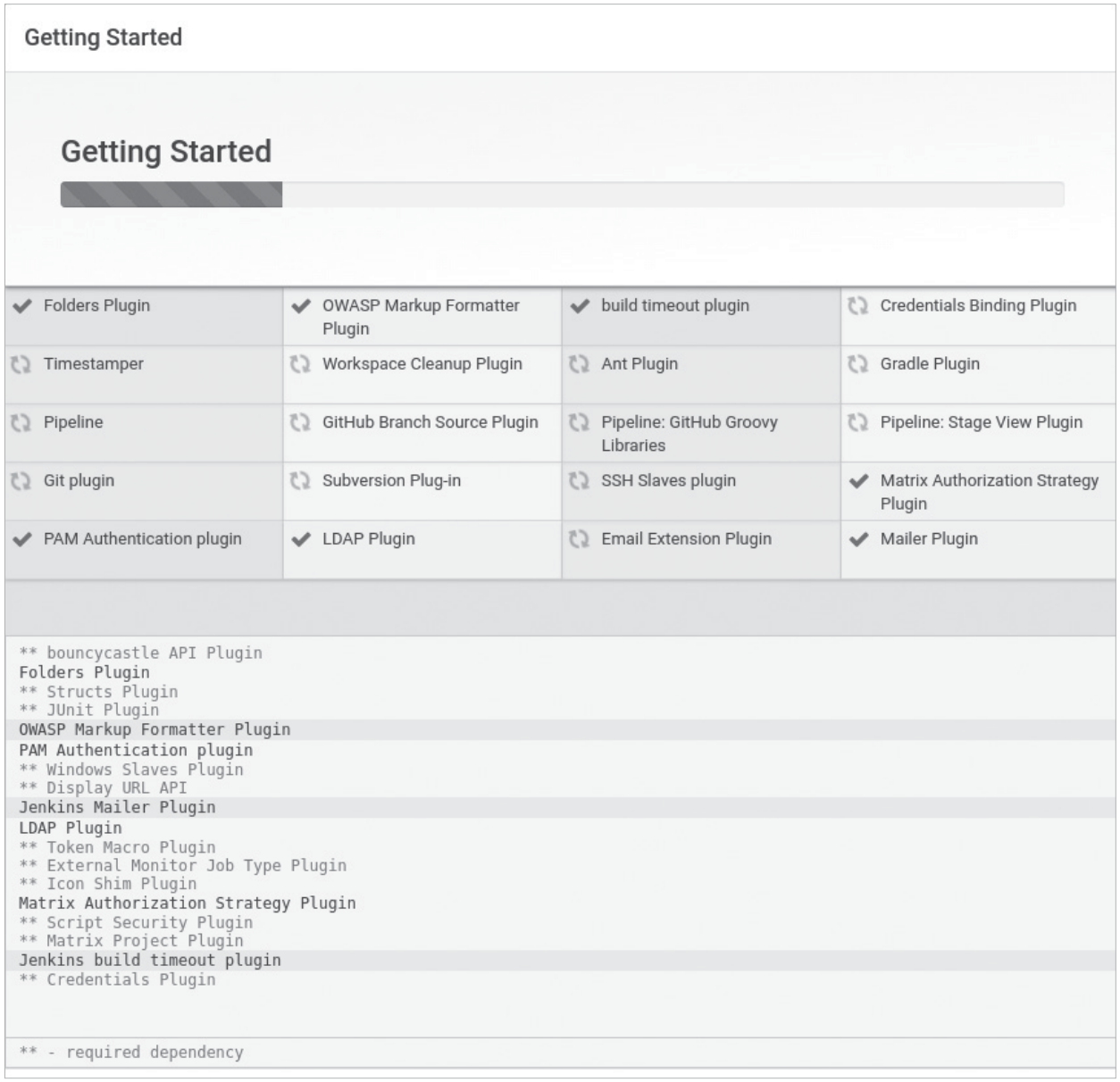


Figure 3: Getting started

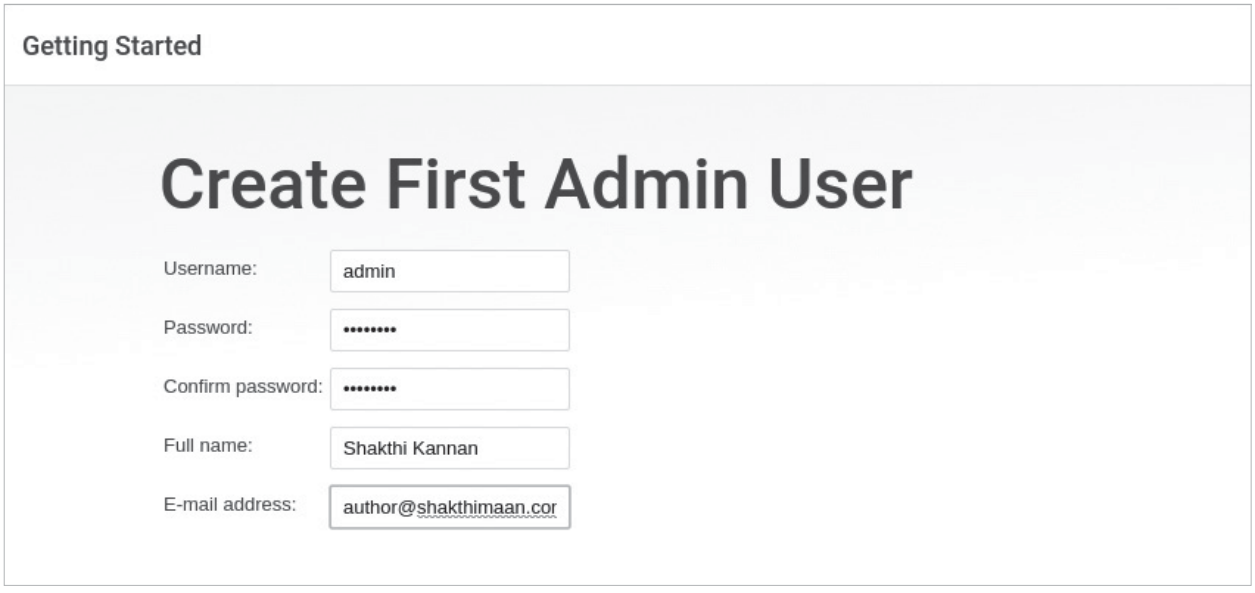


Figure 4: Creating the first admin user

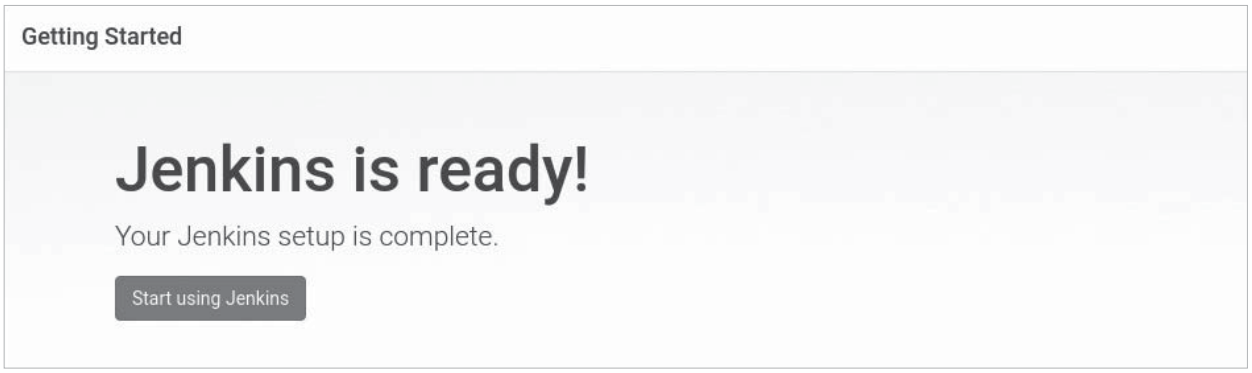


Figure 5: 'Jenkins is ready!'

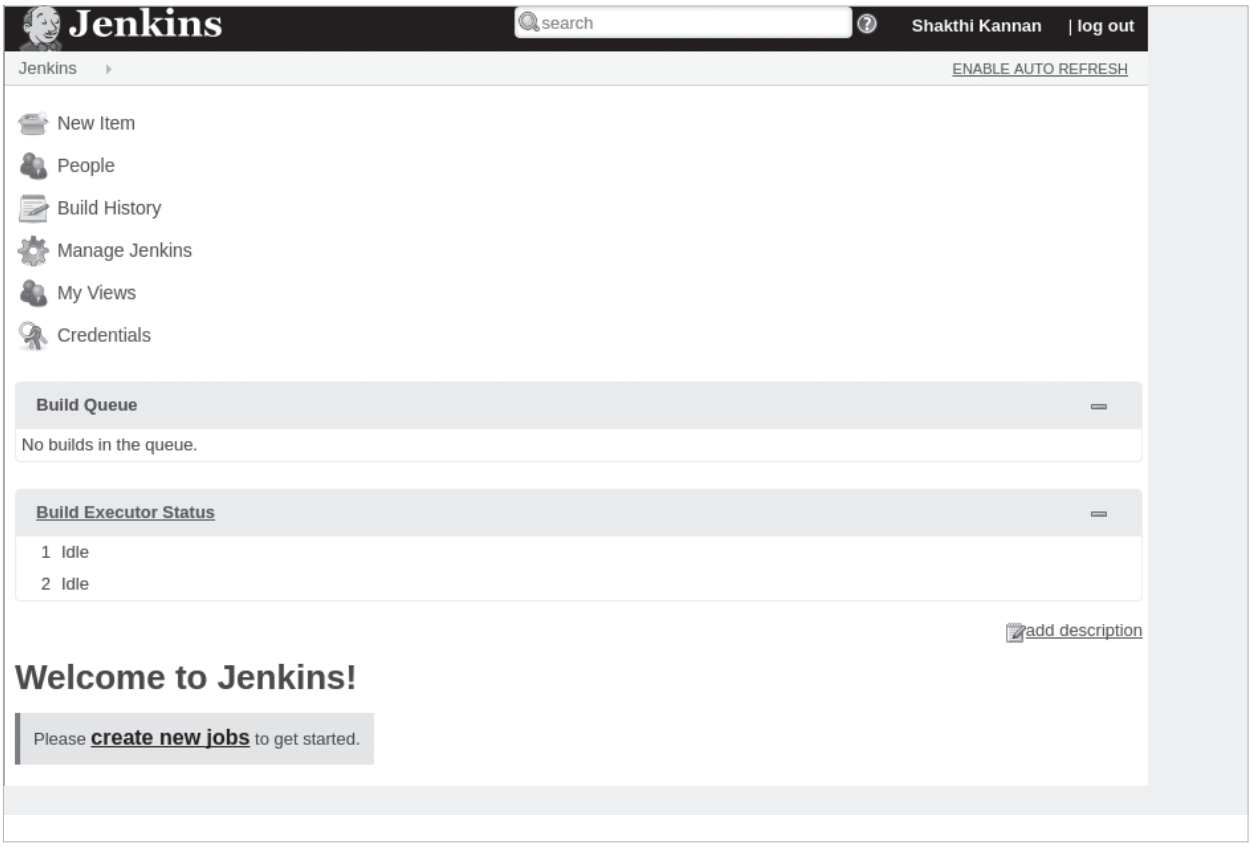


Figure 6: The Jenkins dashboard

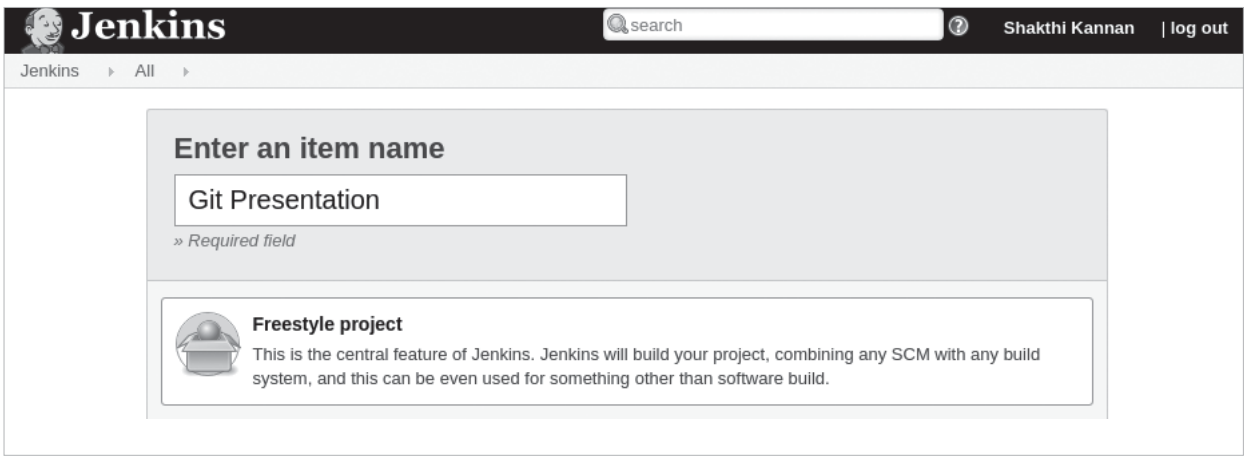


Figure 7: Enter an item name

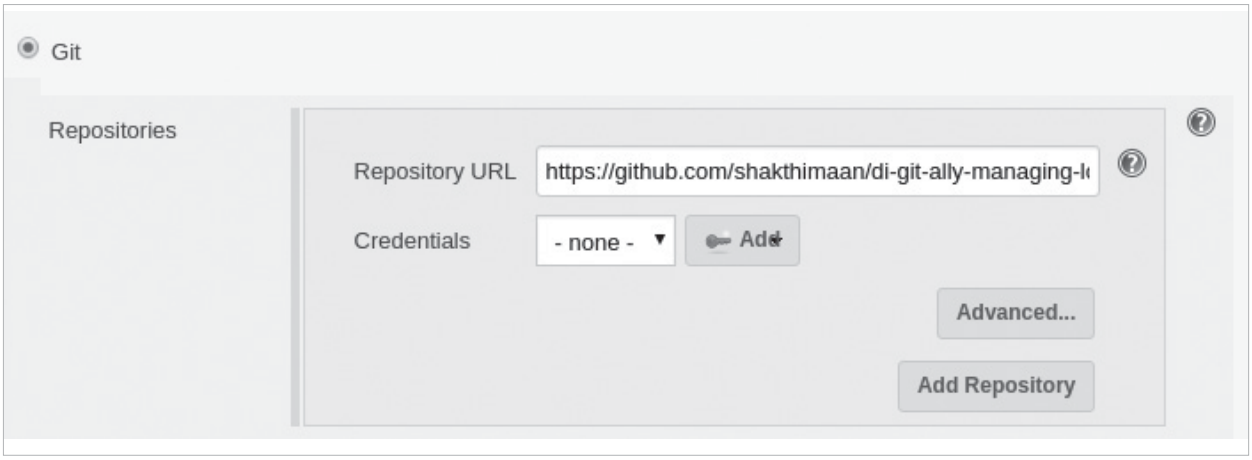


Figure 8: Add the GitHub repo

Figure 9: The *Build* step

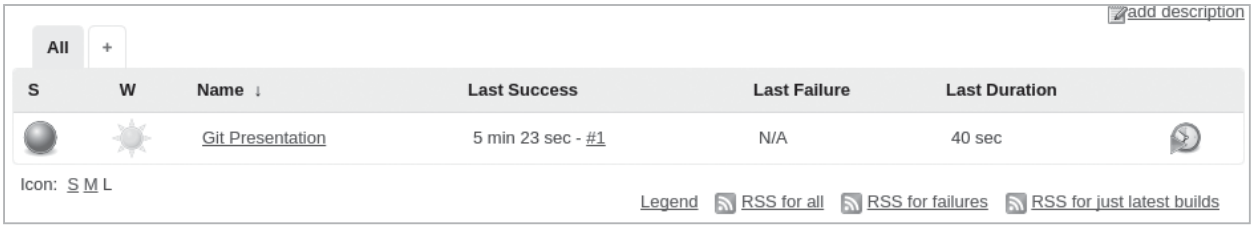


Figure 10: Build success

The next step is to add the GitHub repo to the *Repositories* section. The GitHub HTTPS URL is provided as we are not going to use any credentials in this example. By default, the master branch will be built. The form to enter the GitHub URL is shown in Figure 8.

A Makefile is available in the project source code, and hence we can simply invoke *make* to build the project. The *Execute shell* option is chosen in the *Build* step, and the *make clean; make* command is added to the *Build* step, as shown in Figure 9.

From the left panel, you can click on the *Build Now* link for the project to trigger a build. After a successful build, you should see a screenshot

similar to Figure 10.

Uninstalling

An uninstall script to remove the Jenkins server is available in the *playbooks/admin* folder. It is given below for reference:

```
---
---
- name: Uninstall Jenkins
  hosts: jenkins
  gather_facts: true
  become: yes
  become_method: sudo
  tags: [remove]

  tasks:
    - name: Stop Jenkins server
      service:
        name: jenkins
        state: stopped

    - name: Uninstall packages
      package:
        name: "{{ item }}"
        state: absent
      with_items:
        - jenkins
```

The script can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-
jenkins.yml
```

Chapter 8

Creating a Virtual Machine for Erlang/OTP

This seventh article in the DevOps series is a tutorial on how to create a test virtual machine (VM) to compile, build, and test Erlang/OTP from its source code. You can then adapt the method to create different VMs for various Erlang releases.

Erlang is a programming language designed by Ericsson primarily for soft real-time systems. The Open Telecom Platform (OTP) consists of libraries, applications and tools to be used with Erlang to implement services that require high availability. In this article, we will create a test virtual machine (VM) to compile, build, and test Erlang/OTP from its source code. This allows you to create VMs with different Erlang release versions for testing.

The Erlang programming language was developed by Joe Armstrong, Robert Virding and Mike Williams in 1986 and released as free and open source software in 1998. It was initially designed to work with telecom switches, but is widely used today in large scale, distributed systems. Erlang is a concurrent and functional programming language, and is released under the Apache License 2.0.

Setting it up

A CentOS 6.8 virtual machine (VM) running on KVM is used for the installation. Internet access should be available from the guest machine. The VM should have at least 2GB of RAM allotted to build the Erlang/OTP documentation. The Ansible version used on the host (Parabola GNU/Linux-libre x86_64) is 2.3.0.0. The *ansible/* folder contains the following files:

```
ansible/inventory/kvm/inventory
```

```
ansible/playbooks/configuration/erlang.yml
```

The IP address of the guest CentOS 6.8 VM is added to the *inventory* file as shown below:

```
erlang ansible_host=192.168.122.150 ansible_connection=ssh ansible_user=bravo
ansible_password=password
```

An entry for the *erlang* host is also added to the */etc/hosts* file as indicated below:

```
192.168.122.150 erlang
```

A 'bravo' user account is created on the test VM, and is added to the 'wheel' group. The */etc/sudoers* file also has the following line uncommented, so that the 'bravo' user will be able to execute sudo commands:

```
## Allows people in group wheel to run all commands
%wheel ALL=(ALL) ALL
```

We can obtain the Erlang/OTP sources from a stable tarball, or clone the Git repository. The steps involved in both these cases are discussed below.

Building from the source tarball

The Erlang/OTP stable releases are available at *http://www.erlang.org/downloads.* The build process is divided into many steps, and we shall go through each one of them. The *version* of Erlang/OTP can be passed as an argument to the playbook. Its default value is the release 19.0, and is defined in the variable section of the playbook as shown below:

```
vars:
  ERL_VERSION: "otp_src_{{ version | default('19.0') }}"
  ERL_DIR: "{{ ansible_env.HOME }}/installs/erlang"
  ERL_TOP: "{{ ERL_DIR }}/{{ ERL_VERSION }}"
  TEST_SERVER_DIR: "{{ ERL_TOP }}/release/tests/test_server"
```

The *ERL_DIR* variable represents the directory where the tarball will be downloaded, and the *ERL_TOP* variable refers to the top-level directory location containing the source code. The path to the test directory from where the tests will be invoked is given by the *TEST_SERVER_DIR* variable.

Erlang/OTP has mandatory and optional package dependencies.

Let's first update the software package repository, and then install the required dependencies as indicated below:

```
tasks:
  - name: Update the software package repository
    become: true
    yum:
      name: '*'
      update_cache: yes

  - name: Install dependencies
    become: true
    package:
      name: "{{ item }}"
      state: latest
    with_items:
      - wget
      - make
      - gcc
      - perl
      - m4
      - ncurses-devel
      - sed
      - libxslt
      - fop
```

The Erlang/OTP sources are written using the 'C' programming language. The GNU C Compiler (GCC) and GNU Make are used to compile the source code. The 'libxslt' and 'fop' packages are required to generate the documentation. The build directory is then created, the source tarball is downloaded and it is extracted to the directory mentioned in *ERL_DIR*.

```
- name: Create destination directory
  file: path="{{ ERL_DIR }}" state=directory

- name: Download and extract Erlang source tarball
  unarchive:
    src: "http://erlang.org/download/{{ ERL_VERSION }}.tar.gz"
    dest: "{{ ERL_DIR }}"
    remote_src: yes
```

The 'configure' script is available in the sources, and it is used to generate the Makefile based on the installed software. The 'make'

command will build the binaries from the source code.

```
- name: Build the project
  command: "{{ item }} chdir={{ ERL_TOP }}"
  with_items:
    - ./configure
    - make
  environment:
    ERL_TOP: "{{ ERL_TOP }}"
```

After the 'make' command finishes, the 'bin' folder in the top-level sources directory will contain the Erlang 'erl' interpreter. The Makefile also has targets to run tests to verify the built binaries. We are remotely invoking the test execution from Ansible and hence *-noshell -noinput* are passed as arguments to the Erlang interpreter, as shown in the *.yaml* file.

```
- name: Prepare tests
  command: "{{ item }} chdir={{ ERL_TOP }}"
  with_items:
    - make release_tests
  environment:
    ERL_TOP: "{{ ERL_TOP }}"
```

```
- name: Execute tests
  shell: "cd {{ TEST_SERVER_DIR }} && {{ ERL_TOP }}/bin/erl -noshell -noinput
-s ts install -s ts smoke_test batch -s init stop"
```

You need to verify that the tests have passed successfully by checking the *$ERL_TOP/release/tests/test_server/index.html* page in a browser. A screenshot of the test results is shown in Figure 1.

The built executables and libraries can then be installed on the system using the *make install* command. By default, the install directory is */usr/local*.

```
- name: Install
  command: "{{ item }} chdir={{ ERL_TOP }}"
  with_items:
    - make install
  become: true
  environment:
    ERL_TOP: "{{ ERL_TOP }}"
```

The documentation can also be generated and installed as shown below:

```
- name: Make docs
  shell: "cd {{ ERL_TOP }} && make docs"
  environment:
    ERL_TOP: "{{ ERL_TOP }}"
    FOP_HOME: "{{ ERL_TOP }}/fop"
    FOP_OPTS: "-Xmx2048m"

- name: Install docs
  become: true
  shell: "cd {{ ERL_TOP }} && make install-docs"
  environment:
    ERL_TOP: "{{ ERL_TOP }}"
```

The total available RAM (2GB) is specified in the *FOP_OPTS* environment variable. The complete playbook to download, compile, execute the tests, and also generate the documentation is given below:

Test Results

ALL RUNS

Test Name	Label	Test Run Started	Ok	Failed	Skipped (User/Auto)	Missing Suites	Node	CT Log	Old Runs
tests.emulator_test.crypto_SUITE.t_md5	-	Wed Aug 09 2017 05:51:54	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.emulator_test.float_SUITE.cases	-	Wed Aug 09 2017 05:51:54	2	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.emulator_test.smoke_test_SUITE	-	Wed Aug 09 2017 05:51:54	3	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.emulator_test.time_SUITE	-	Wed Aug 09 2017 05:51:54	15	2	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.erl_interface_test.ei_decode_encode_SUITE	-	Wed Aug 09 2017 05:53:22	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.ic_test.ic_SUITE	-	Wed Aug 09 2017 05:53:26	35	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.jinterface_test.jinterface_SUITE.java_erlang_send_receive	-	Wed Aug 09 2017 05:53:25	0	0	1 (1/0)	0	test_server@centos-minimal	CT Log	none
tests.kernel_test.gen_tcp_echo_SUITE.active_echo	-	Wed Aug 09 2017 05:53:01	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.kernel_test.heart_SUITE.reboot	-	Wed Aug 09 2017 05:53:01	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.kernel_test.inet_SUITE.cases	-	Wed Aug 09 2017 05:53:01	2	0	4 (0/4)	0	test_server@centos-minimal	CT Log	none
tests.kernel_test.inet_res_SUITE.cases	-	Wed Aug 09 2017 05:53:01	0	0	5 (1/4)	0	test_server@centos-minimal	CT Log	none
tests.os_mon_test.cpu_sup_SUITE.cases	-	Wed Aug 09 2017 05:53:33	2	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.os_mon_test.disksup_SUITE.api	-	Wed Aug 09 2017 05:53:33	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.os_mon_test.memsup_SUITE.api	-	Wed Aug 09 2017 05:53:33	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
tests.system_test.ethread_SUITE	-	Wed Aug 09 2017 05:53:38	13	0	1 (1/0)	0	test_server@centos-minimal	CT Log	none
tests.system_test.otp_SUITE.undefined_functions	-	Wed Aug 09 2017 05:53:38	1	0	0 (0/0)	0	test_server@centos-minimal	CT Log	none
Total			**79**	**2**	**11 (3/8)**	**0**			

Figure 1: Test results

```
---
- name: Setup Erlang build
  hosts: erlang
  gather_facts: true
  tags: [release]

  vars:
    ERL_VERSION: "otp_src_{{ version | default('19.0') }}"
    ERL_DIR: "{{ ansible_env.HOME }}/installs/erlang"
    ERL_TOP: "{{ ERL_DIR }}/{{ ERL_VERSION }}"
    TEST_SERVER_DIR: "{{ ERL_TOP }}/release/tests/test_server"

  tasks:
    - name: Update the software package repository
      become: true
      yum:
        name: '*'
        update_cache: yes

    - name: Install dependencies
      become: true
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - wget
        - make
        - gcc
        - perl
        - m4
        - ncurses-devel
        - sed
        - libxslt
        - fop

    - name: Create destination directory
      file: path="{{ ERL_DIR }}" state=directory

    - name: Download and extract Erlang source tarball
      unarchive:
        src: "http://erlang.org/download/{{ ERL_VERSION }}.tar.gz"
        dest: "{{ ERL_DIR }}"
```

```
      remote_src: yes

  - name: Build the project
    command: "{{ item }} chdir={{ ERL_TOP }}"
    with_items:
      - ./configure
      - make
    environment:
      ERL_TOP: "{{ ERL_TOP }}"

  - name: Prepare tests
    command: "{{ item }} chdir={{ ERL_TOP }}"
    with_items:
      - make release_tests
    environment:
      ERL_TOP: "{{ ERL_TOP }}"

  - name: Execute tests
    shell: "cd {{ TEST_SERVER_DIR }} && {{ ERL_TOP }}/bin/erl -noshell
-noinput -s ts install -s ts smoke_test batch -s init stop"

  - name: Install
    command: "{{ item }} chdir={{ ERL_TOP }}"
    with_items:
      - make install
    become: true
    environment:
      ERL_TOP: "{{ ERL_TOP }}"

  - name: Make docs
    shell: "cd {{ ERL_TOP }} && make docs"
    environment:
      ERL_TOP: "{{ ERL_TOP }}"
      FOP_HOME: "{{ ERL_TOP }}/fop"
      FOP_OPTS: "-Xmx2048m"

  - name: Install docs
    become: true
    shell: "cd {{ ERL_TOP }} && make install-docs"
    environment:
      ERL_TOP: "{{ ERL_TOP }}"
```

The playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/erlang.yml
-e "version=19.0" --tags "release" -K
```

Building from the Git repository

We can build the Erlang/OTP sources from the Git repository too. The complete playbook is given below for reference:

```
- name: Setup Erlang Git build
  hosts: erlang
  gather_facts: true
  tags: [git]

  vars:
    GIT_VERSION: "otp"
    ERL_DIR: "{{ ansible_env.HOME }}/installs/erlang"
    ERL_TOP: "{{ ERL_DIR }}/{{ GIT_VERSION }}"
    TEST_SERVER_DIR: "{{ ERL_TOP }}/release/tests/test_server"

  tasks:
    - name: Update the software package repository
      become: true
      yum:
        name: '*'
        update_cache: yes

    - name: Install dependencies
      become: true
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - wget
        - make
        - gcc
        - perl
        - m4
        - ncurses-devel
        - sed
        - libxslt
        - fop
        - git
        - autoconf
```

```
  - name: Create destination directory
    file: path="{{ ERL_DIR }}" state=directory

  - name: Clone the repository
    git:
      repo: "https://github.com/erlang/otp.git"
      dest: "{{ ERL_DIR }}/otp"

  - name: Build the project
    command: "{{ item }} chdir={{ ERL_TOP }}"
    with_items:
      - ./otp_build autoconf
      - ./configure
      - make
    environment:
      ERL_TOP: "{{ ERL_TOP }}"
```

The 'git' and 'autoconf' software packages are required for downloading and building the sources from the Git repository. The Ansible Git module is used to clone the remote repository. The source directory provides an *otp_build* script to create the *configure* script. You can invoke the above playbook as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/erlang.yml
--tags "git" -K
```

You are encouraged to read the complete installation documentation at *https://github.com/erlang/otp/blob/master/HOWTO/INSTALL.md.*

Chapter 9

Using Docker with Ansible

This article is the eighth in the DevOps series. This month, we shall learn to set up Docker in the host system and use it with Ansible.

Docker provides operating system level virtualisation in the form of containers. These containers allow you to run standalone applications in an isolated environment. The three important features of Docker containers are isolation, portability and repeatability. All along we have used Parabola GNU/Linux-libre as the host system, and executed Ansible scripts on target virtual machines (VM) such as CentOS and Ubuntu.

Docker containers are extremely lightweight and fast to launch. You can also specify the amount of resources that you need such as the CPU, memory and network. The Docker technology was launched in 2013, and released under the Apache 2.0 licence. It is implemented using the Go programming language. A number of frameworks have been built on top of Docker for managing these clusters of servers. The Apache Mesos project, Google's Kubernetes, and the Docker Swarm project are popular examples. These are ideal for running stateless applications and help you to easily scale horizontally.

Setting it up

The Ansible version used on the host system (Parabola GNU/Linux-libre x86_64) is 2.3.0.0. Internet access should be available on the host system. The *ansible/* folder contains the following file:

```
ansible/playbooks/configuration/docker.yml
```

Installation

The following playbook is used to install Docker on the host system:

```
---
- name: Setup Docker
```

```
  hosts: localhost
  gather_facts: true
  become: true
  tags: [setup]

  tasks:
    - name: Update the software package repository
      pacman:
        update_cache: yes

    - name: Install dependencies
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - python2-docker
        - docker

    - service:
        name: docker
        state: started
    - name: Run the hello-world container
      docker_container:
        name: hello-world
        image: library/hello-world
```

The Parabola package repository is updated before proceeding to install the dependencies. The *python2-docker* package is required for use with Ansible. Hence, it is installed along with the *docker* package. The Docker daemon service is then started and the *library/hello-world* container is fetched and executed. A sample invocation and execution of the above playbook is shown below:

```
$ ansible-playbook playbooks/configuration/docker.yml -K --tags=setup
SUDO password:

PLAY [Setup Docker] ************************************

TASK [Gathering Facts] *********************************
ok: [localhost]

TASK [Update the software package repository] ***************
```

```
changed: [localhost]

TASK [Install dependencies] ********************************
ok: [localhost] => (item=python2-docker)
ok: [localhost] => (item=docker)

TASK [service] *********************************************
ok: [localhost]

TASK [Run the hello-world container] ***********************
changed: [localhost]

PLAY RECAP *************************************************
localhost                  : ok=5    changed=2    unreachable=0    failed=0
```

With the verbose '-v' option to ansible-playbook, you will see an entry for LogPath, such as */var/lib/docker/containers/<container-id>/<container-id>-json.log*. In this log file, you will see the output of the execution of the *hello-world* container. This output is the same when you run the container manually as shown below:

```
$ sudo docker run hello-world
Hello from Docker!
```

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the *hello-world* image from the Docker Hub.
3. The Docker daemon created a new container from that image, which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

You can share images, automate workflows, and more with a free Docker ID at *https://cloud.docker.com/*.

For more examples and ideas, do visit *https://docs.docker.com/engine/userguide/*.

An example

A deep learning (DL) Docker project is available *(https://github.com/floydhub/dl-docker)* with support for frameworks, libraries and software tools. We can use Ansible to build the entire DL container from the source code of the tools. The base OS of the container is Ubuntu 14.04, and will include the following software packages:

- TensorFlow
- Caffe
- Theano
- Keras
- Lasagne
- Torch
- iPython/Jupyter Notebook
- Numpy
- SciPy
- Pandas
- Scikit Learn
- Matplotlib
- OpenCV

The playbook to build the DL Docker image is given below:

```
- name: Build the dl-docker image
  hosts: localhost
  gather_facts: true
  become: true
  tags: [deep-learning]

  vars:
    DL_BUILD_DIR: "/tmp/dl-docker"
    DL_DOCKER_NAME: "floydhub/dl-docker"

  tasks:
    - name: Download dl-docker
      git:
        repo: https://github.com/saiprashanths/dl-docker.git
        dest: "{{ DL_BUILD_DIR }}"

    - name: Build image and with buildargs
      docker_image:
         path: "{{ DL_BUILD_DIR }}"
         name: "{{ DL_DOCKER_NAME }}"
         dockerfile: Dockerfile.cpu
         buildargs:
```

```
          tag: "{{ DL_DOCKER_NAME }}:cpu"
```

We first clone the deep learning Docker project sources. The *docker_image* module in Ansible helps us to build, load and pull images. We then use the *Dockerfile.cpu* file to build a Docker image targeting the CPU. If you have a GPU in your system, you can use the *Dockerfile.gpu* file. The above playbook can be invoked using the following command:

```
$ ansible-playbook playbooks/configuration/docker.yml -K --tags=deep-learning
```

Depending on the CPU and RAM you have, it will take a considerable amount of time to build the image with all the software. So be patient!

Jupyter Notebook

The built *dl-docker* image contains Jupyter Notebook, which can be launched when you start the container. An Ansible playbook for the same is provided below:

```
- name: Start Jupyter notebook
  hosts: localhost
  gather_facts: true
  become: true
  tags: [notebook]

  vars:
    DL_DOCKER_NAME: "floydhub/dl-docker"

  tasks:
    - name: Run container for Jupyter notebook
      docker_container:
        name: "dl-docker-notebook"
        image: "{{ DL_DOCKER_NAME }}:cpu"
        state: started
        command: sh run_jupyter.sh
```

You can invoke the playbook using the following command:

```
$ ansible-playbook playbooks/configuration/docker.yml -K --tags=notebook
```

The Dockerfile already exposes the port 8888, and hence you do not need to specify the same in the above *docker_container* configuration. After you run the playbook, using the 'docker ps' command on the host system, you can obtain the container ID as indicated below:

```
$ sudo docker ps
CONTAINER ID        IMAGE      COMMAND       CREATED          STATUS
PORTS           NAMES
a876ad5af751        floydhub/dl-docker:cpu   “sh run_jupyter.sh”   11 minutes
ago      Up 4 minutes        6006/tcp, 8888/tcp   dl-docker-notebook
```

Figure 1: Jupyter Notebook

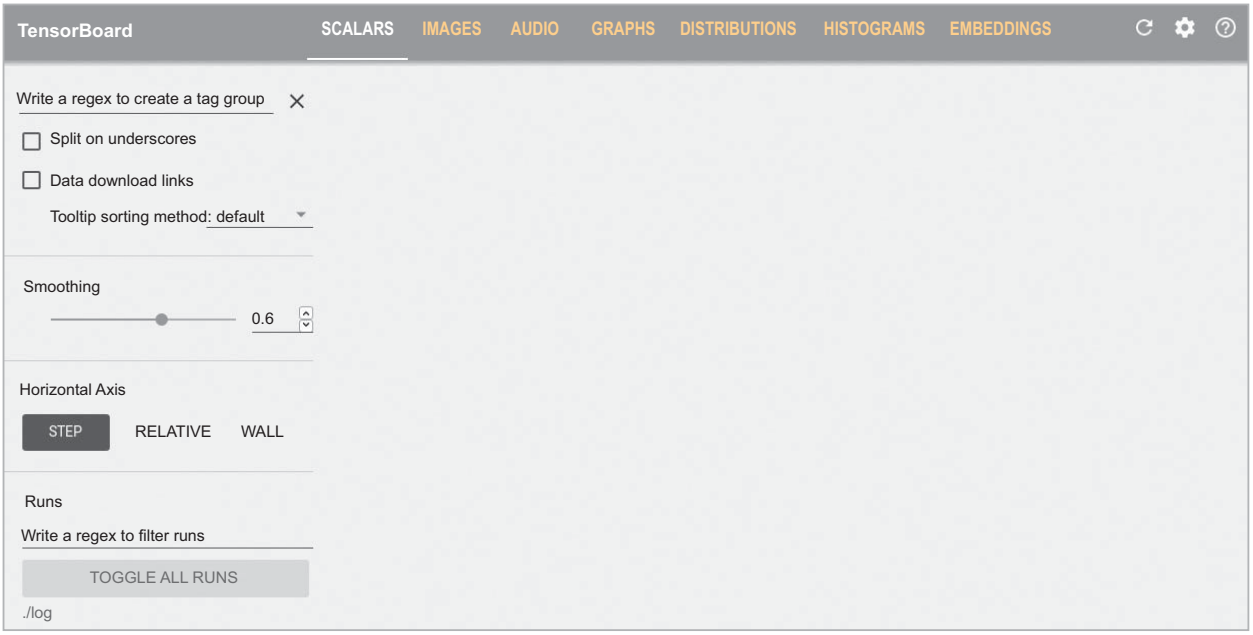


Figure 2: TensorBoard

You can now log in to the running container using the following command:

```
$ sudo docker exec -it a876 /bin/bash
```

You can then run an 'ifconfig' command to find the local IP address ('172.17.0.2' in this case), and then open *http://172.17.0.2:8888* in a browser on your host system to see the Jupyter Notebook. A screenshot is shown in Figure 1.

TensorBoard

TensorBoard consists of a suite of visualisation tools to understand the TensorFlow programs. It is installed and available inside the Docker container. After you log in to the Docker container, at the root prompt, you can start TensorBoard by passing it a log directory as shown below:

```
# tensorboard --logdir=./log
```

You can then open *http://172.17.0.2:6006/* in a browser on your host system to see the TensorBoard dashboard as shown in Figure 2.

Docker image facts

The *docker_image_facts* Ansible module provides useful information about a Docker image. We can use it to obtain the image facts for our *dl-docker* container as shown below:

```
- name: Get Docker image facts
  hosts: localhost
  gather_facts: true
  become: true
  tags: [facts]

  vars:
    DL_DOCKER_NAME: "floydhub/dl-docker"

  tasks:
    - name: Get image facts
      docker_image_facts:
        name: "{{ DL_DOCKER_NAME }}:cpu"
```

The above playbook can be invoked as follows:

```
$ ANSIBLE_STDOUT_CALLBACK=json ansible-playbook playbooks/configuration/docker.
yml -K --tags=facts
```

The ANSIBLE_STDOUT_CALLBACK environment variable is set to 'json' to produce a JSON output for readability. Some important image facts from the invocation of the above playbook are shown below:

```
"Architecture": "amd64",
"Author": "Sai Soundararaj <saip@outlook.com>",

"Config": {

"Cmd": [
  "/bin/bash"
],

"Env": [
  "PATH=/root/torch/install/bin:/root/caffe/build/tools:/root/caffe/python:/
usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin",
  "CAFFE_ROOT=/root/caffe",
  "PYCAFFE_ROOT=/root/caffe/python",
  "PYTHONPATH=/root/caffe/python:",
  "LUA_PATH=/root/.luarocks/share/lua/5.1/?.lua;/root/.luarocks/share/
lua/5.1/?/init.lua;/root/torch/install/share/lua/5.1/?.lua;/root/torch/install/
share/lua/5.1/?/init.lua;./?.lua;/root/torch/install/share/luajit-2.1.0-
beta1/?.lua;/usr/local/share/lua/5.1/?.lua;/usr/local/share/lua/5.1/?/init.
lua",
  "LUA_CPATH=/root/torch/install/lib/?.so;/root/.luarocks/lib/lua/5.1/?.so;/
root/torch/install/lib/lua/5.1/?.so;./?.so;/usr/local/lib/lua/5.1/?.so;/usr/
local/lib/lua/5.1/loadall.so",
  "LD_LIBRARY_PATH=/root/torch/install/lib:",
  "DYLD_LIBRARY_PATH=/root/torch/install/lib:"
],

"ExposedPorts": {
  "6006/tcp": {},
  "8888/tcp": {}
},
"Created": "2016-06-13T18:13:17.247218209Z",
"DockerVersion": "1.11.1",
"Os": "linux",

"task": { "name": "Get image facts" }
```


You are encouraged to read the 'Getting Started with Docker' user guide available at *http://docs.ansible.com/ansible/latest/guide_docker.html* to know more about using Docker with Ansible.

Chapter 10

Provisioning with Ansible

Ansible is the simplest way to automate apps and IT infrastructure. It meshes well with DevOps to deploy apps. In this ninth article in the series on DevOps, we explore the use of Ansible for launching Docker containers and provisioning virtual machines.

Provisioning is the first step in an application's deployment process. In a cloud environment, software can be run from a Docker container, virtual machine or bare metal, and Ansible can be used for provisioning such systems. In this article, we explore how to use Ansible to launch Docker containers and provision virtual machines.

Setting it up

Let's create an Ansible playbook for the 'Get started with Docker Compose' composetest example available at *https://docs.docker.com/compose/gettingstarted/*. The Ansible version used on the host system (Ubuntu x86_64) is 2.2.0.0. You will need to install Docker CE and docker-compose on Ubuntu. Follow the installation guide provided at *https://docs.docker.com/engine/ installation/linux/docker-ce/ubuntu/#install-using-the-repository* to install Docker CE. You can then install docker-compose using the APT package manager:

```
$ sudo apt-get install docker-compose
```

The *composetest/* folder consists of the following files:

```
composetest/app.py
composetest/docker-compose.yml
composetest/Dockerfile
composetest/provision.yml
composetest/requirements.txt
```

The *app.py* file contains a basic Flask application that communicates with a backend Redis database server. Its file contents are as follows:

```
from flask import Flask
from redis import Redis

app = Flask(__name__)
redis = Redis(host='redis', port=6379)

@app.route('/')
def hello():
    count = redis.incr('hits')
    return 'Hello World! I have been seen {} times.\n'.format(count)

if __name__ == "__main__":
    app.run(host="0.0.0.0", debug=True)
```

An HTTP request to the Flask application returns the text string 'Hello World! I have been seen N times.' This will be run inside a Docker container. The *requirements.txt* file is provided to list the dependencies required for the project:

```
flask
redis
```

Now let's provision a minimalistic Docker container that has support for Python and is based on Alpine (a security-oriented, lightweight GNU/Linux distribution). The Dockerfile for the application is provided below for reference:

```
FROM python:3.4-alpine
ADD . /code
WORKDIR /code
RUN pip install -r requirements.txt
CMD ["python", "app.py"]
```

The *Docker-compose.yml* file is used to create the images and will also be used by Ansible. It defines the services that will be deployed in the containers:

```
version: '2'
services:
```

```
  web:
    build: .
    ports:
     - "5000:5000"
  redis:
    image: "redis:alpine"
    ports:
      - "6379:6379"
```

Provisioning

The Python Web application will be running on port 5000, whereas the Redis database server will be listening on port 6379. We will first build the application using the following code:

```
$ docker-compose up

Creating composetest_web_1
Creating composetest_redis_1
Attaching to composetest_web_1, composetest_redis_1
redis_1  | 1:C 05 Oct 11:40:49.067 # oO0OoO0OoO0Oo Redis is starting
oO0OoO0OoO0Oo

redis_1  | 1:C 05 Oct 11:40:49.067 # Redis version=4.0.2, bits=64,
commit=00000000, modified=0, pid=1, just started

redis_1  | 1:C 05 Oct 11:40:49.067 # Warning: no config file specified, using the
default config. In order to specify a config file use redis-server /path/to/redis.
conf

redis_1  | 1:M 05 Oct 11:40:49.070 * Running mode=standalone, port=6379.

redis_1  | 1:M 05 Oct 11:40:49.070 # WARNING: The TCP backlog setting of 511
cannot be enforced because /proc/sys/net/core/somaxconn is set to the lower
value of 128.

redis_1  | 1:M 05 Oct 11:40:49.070 # Server initialized

redis_1  | 1:M 05 Oct 11:40:49.070 # WARNING overcommit_memory is set to 0!
Background save may fail under low memory condition. To fix this issue add 'vm.
overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command
'sysctl vm.overcommit_memory=1' for this to take effect.
```

```
redis_1  | 1:M 05 Oct 11:40:49.070 # WARNING you have Transparent Huge Pages
(THP) support enabled in your kernel. This will create latency and memory usage
issues with Redis. To fix this issue run the command 'echo never > /sys/kernel/
mm/transparent_hugepage/enabled' as root, and add it to your /etc/rc.local in
order to retain the setting after a reboot. Redis must be restarted after THP
is disabled.

redis_1  | 1:M 05 Oct 11:40:49.070 * Ready to accept connections
web_1    |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1    |  * Restarting with stat
web_1    |  * Debugger is active!
web_1    |  * Debugger PIN: 100-456-831
```

If you start a browser on the host system and open the URL *http://0.0.0.0:5000*, you will see the text from the Flask application. You can continue to refresh the page making requests to the application, and you will see the count increasing in the text: 'Hello World! I have been seen N times.' Pressing *Ctrl+c* in the above terminal will stop the application. Let's now create an Ansible playbook to launch these containers:

```
- name: Provision Flask application
  hosts: localhost
  connection: local
  become: true
  gather_facts: true
  tags: [setup]

  tasks:
  - docker_service:
      project_name: composetest
      definition:
        version: '2'
        services:
          web:
            build: "{{ playbook_dir }}/."
            ports:
              - "5000:5000"
          redis:
            image: "redis:alpine"
    register: output

  - debug:
      var: output
```

```
  - assert:
      that:
        - "web.composetest_web_1.state.running"
- "redis.composetest_redis_1.state.running"
```

The above playbook can be invoked as follows:

```
$ sudo ansible-playbook provision.yml --tags setup
```

The *docker_service* module is used to compose the services—a Web application and a Redis database server. The output of launching the containers is stored in a variable and is used to ensure that both the backend services are up and running. You can verify that the containers are running using the *docker ps* command output as shown below:

```
$ docker ps
CONTAINER ID        IMAGE               COMMAND                  CREATED
STATUS              PORTS                    NAMES
03f6f6a3d48f        composetest_web     "python app.py"          18 seconds ago
Up 17 seconds       0.0.0.0:5000->5000/tcp   composetest_web_1
fa00c70da13a        redis:alpine        "docker-entrypoint..."   18 seconds ago
Up 17 seconds       6379/tcp                 composetest_redis_1
```

Scaling

You can use the *docker_service* Ansible module to increase the number of Web services to two, as shown in the following Ansible playbook:

```
- name: Scale the web services to 2
  hosts: localhost
  connection: local
  become: true
  gather_facts: true
  tags: [scale]

  tasks:
    - docker_service:
        project_src: "/home/guest/composetest"
        scale:
          web: 2
      register: output

    - debug:
```

```
      var: output

  - name: Start container two
    docker_container:
      name: composetest_web_2
      image: composetest_web
      state: started
      ports:
        - "5001:5000"
      network_mode: bridge
      networks:
        - name: composetest_default
          ipv4_address: "172.19.0.12"
```

The above playbook can be invoked as follows:

```
$ sudo ansible-playbook provision.yml --tags scale
```

The execution of the playbook will create one more Web application server, and this will listen on Port 5001. You can verify the running containers as follows:

```
$ docker ps
CONTAINER ID   IMAGE            COMMAND      CREATED             STATUS
PORTS                  NAMES
66b59eb163c3   composetest_web "python app.py" 9 seconds ago       Up 8 seconds
0.0.0.0:5001->5000/tcp   composetest_web_2
4e8a37344598   redis:alpine  "docker-entrypoint..." 11 seconds ago    Up 10
seconds   0.0.0.0:6379->6379/tcp   composetest_redis_1
03f6f6a3d48f composetest_web   "python app.py" 55 seconds ago      Up 54
seconds  0.0.0.0:5000->5000/tcp   composetest_web_1
```

You can open another tab in the browser with the URL *http://localhost:5001* on the host system, and the text count will continue to increase if you keep refreshing the page repeatedly.

Cleaning up

You can stop and remove all the running instances. First, stop the newly created Web application, as follows:

```
$ docker stop 66b
```

You can use the following Ansible playbook to stop the containers that were started using Docker compose:

```
- name: Stop all!
  hosts: localhost
  connection: local
  become: true
  gather_facts: true
  tags: [stop]

  tasks:
    - docker_service:
        project_name: composetest
        project_src: "{{ playbook_dir }}/."
        state: absent
```

The above playbook can be invoked using the following command:

```
$ sudo ansible-playbook provision.yml --tags stop
```

You can also verify that there are no containers running in the system, as follows:

```
$ docker ps
CONTAINER ID    IMAGE         COMMAND             CREATED             STATUS
PORTS         NAMES
```

Refer to the Ansible *docker_service* module's documentation at *http://docs.ansible.com/ansible/ latest/docker_service_module.html* for more examples and options.

Vagrant and Ansible

Vagrant is free and open source software (FOSS) that helps to build and manage virtual machines. It allows you to create machines using different backend providers such as VirtualBox, Docker, libvirt, etc. It is developed by HashiCorp and is written in the Ruby programming language. It was first released in 2010 under an MIT licence. The Vagrantfile describes the virtual machine using a Ruby DSL, and an Ansible playbook can be executed as part of the provisioning process.

The following dependencies need to be installed on the host Ubuntu system:

```
$ sudo apt-get build-dep vagrant ruby-libvirt
$ sudo apt-get install qemu libvirt-bin ebtables dnsmasq virt-manager
$ sudo apt-get libxslt-dev libxml2-dev libvirt-dev zlib1g-dev ruby-dev
```

Vagrant 1.8.7 is then installed on Ubuntu using a *.deb* package obtained from the *https://www.vagrantup.com/* website. Issue the following command to install the *vagrant-libvirt* provider:

```
$ vagrant plugin install vagrant-libvirt
```

The *firewalld* daemon is then started on the host system, as follows:

```
$ sudo systemctl start firewalld
```

A simple Vagrantfile is created inside a test directory to launch an Ubuntu guest system. Its file contents are given below for reference:

```
# -*- mode: ruby -*-
# vi: set ft=ruby :

Vagrant.configure("2") do |config|
  config.vm.define :test_vm do |test_vm|
    test_vm.vm.box = "sergk/xenial64-minimal-libvirt"
  end

  config.vm.provision "ansible" do |ansible|
    ansible.playbook = "playbook.yml"
  end
end
```

When provisioning the above Vagrantfile, a minimalistic Xenial 64-bit Ubuntu image is downloaded, started and the Ansible playbook is executed after the instance is launched. The contents of the *playbook.yml* file are as follows:

```
---
- hosts: all
  become: true
  gather_facts: no

  pre_tasks:
    - name: Install python2
```

```
    raw: sudo apt-get -y install python-simplejson

  tasks:
    - name: Update apt cache
      apt: update_cache=yes

    - name: Install Apache
      apt: name=apache2 state=present
```

The minimal Ubuntu machine has Python 3 by default, and the Ansible that we use requires Python 2. Hence, we install Python 2, update the APT repository and install the Apache Web server. A sample debug execution of the above playbook from the test directory is given below:

```
$ VAGRANT_LOG=debug sudo vagrant up --provider=libvirt

Bringing machine 'test_vm' up with 'libvirt' provider...
==> test_vm: Creating image (snapshot of base box volume).
==> test_vm: Creating domain with the following settings...
==> test_vm:  -- Name:              vagrant-libvirt-test_test_vm
==> test_vm:  -- Domain type:       kvm
==> test_vm:  -- Cpus:              1
==> test_vm:  -- Feature:           acpi
==> test_vm:  -- Feature:           apic
==> test_vm:  -- Feature:           pae
==> test_vm:  -- Memory:            512M
==> test_vm:  -- Management MAC:
==> test_vm:  -- Loader:
==> test_vm:  -- Base box:          sergk/xenial64-minimal-libvirt

==> test_vm:  -- Storage pool:      default
==> test_vm:  -- Image:             /var/lib/libvirt/images/vagrant-libvirt-
test_test_vm.img (100G)
==> test_vm:  -- Volume Cache:      default
==> test_vm:  -- Kernel:
==> test_vm:  -- Initrd:
==> test_vm:  -- Graphics Type:     vnc
==> test_vm:  -- Graphics Port:     5900
==> test_vm:  -- Graphics IP:       127.0.0.1
==> test_vm:  -- Graphics Password: Not defined
==> test_vm:  -- Video Type:        cirrus
==> test_vm:  -- Video VRAM:        9216
```

```
==> test_vm:  -- Sound Type:
==> test_vm:  -- Keymap:            en-us
==> test_vm:  -- TPM Path:
==> test_vm:  -- INPUT:             type=mouse, bus=ps2
==> test_vm: Creating shared folders metadata...
==> test_vm: Starting domain.
==> test_vm: Waiting for domain to get an IP address...
==> test_vm: Waiting for SSH to become available...
    test_vm:

    test_vm: Vagrant insecure key detected. Vagrant will automatically replace

    test_vm: this with a newly generated keypair for better security.

    test_vm:
    test_vm: Inserting generated public key within guest...

test_vm: Removing insecure key from the guest if it's present...

    test_vm: Key inserted! Disconnecting and reconnecting using new SSH key...
==> test_vm: Configuring and enabling network interfaces...
==> test_vm: Running provisioner: ansible...
    test_vm: Running ansible-playbook...

PLAY ***************************************************

TASK [Install python2] *********************************
ok: [test_vm]

TASK [Update apt cache] ********************************
ok: [test_vm]

TASK [Install Apache] **********************************
changed: [test_vm]

PLAY RECAP *********************************************
test_vm      : ok=3    changed=1    unreachable=0    failed=0
```

Since you have installed virt-manager, you can now open up the Virtual Machine Manager to see the instance running. You can also log in to the instance using the following command from the test directory:

```
$ vagrant ssh
```

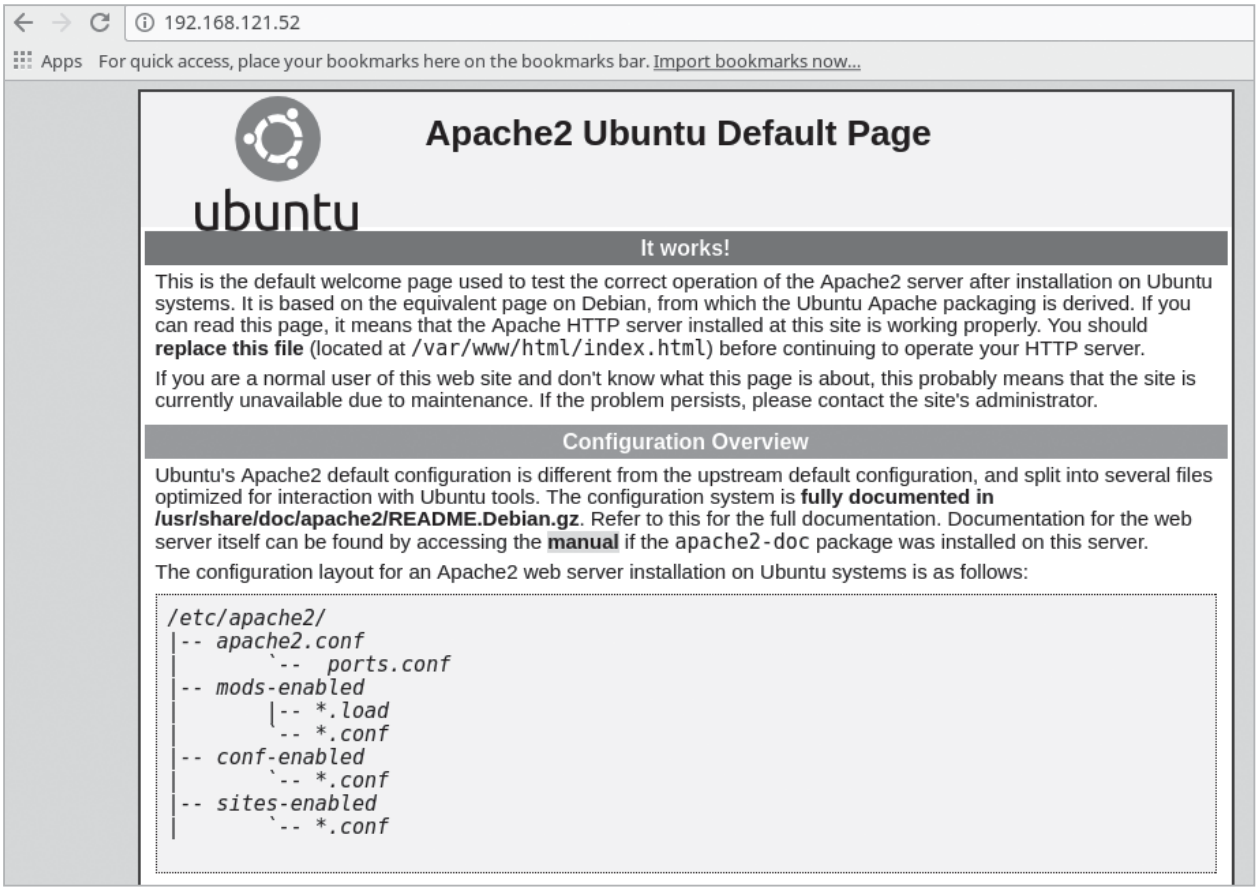


Figure 1: Apache Web server page

After logging into the guest machine, you will find its IP address using the *ifconfig* command. You can then open a browser on the host system with this IP address to see the default Apache Web server home page, as shown in Figure 1.

Chapter 11

Using Ansible to Deploy a Piwigo Photo Gallery

Ansible is the simplest way to automate apps and IT infrastructure. It meshes well with DevOps to deploy apps. In this ninth article in the series on DevOps, we explore the use of Ansible for launching Docker containers and provisioning virtual machines.

Piwigo requires a MySQL database for its back-end and has a number of extensions and plugins developed by the community. You can install it on any shared Web hosting service provider or install it on your own GNU/Linux server. It basically uses the (G)LAMP stack. In this article, we will use Ansible to install and configure a Piwigo instance, which is released under the GNU General Public License (GPL).

You can add photos using the Piwigo Web interface or use an FTP client to synchronise the photos with the server. Each photo is made available in nine sizes, ranging from XXS to XXL. A number of responsive UI themes are available that make use of these different photo sizes, depending on whether you are viewing the gallery on a phone, tablet or computer. The software also allows you to add a watermark to your photos, and you can create nested albums. You can also tag your photos, and Piwigo stores metadata about the photos too. You can even use access control to make photos and albums private. My Piwigo gallery is available at *https://www.shakthimaan.in/gallery/*.

Linux

The Piwigo installation will be on an Ubuntu 15.04 image running as a guest OS using KVM/QEMU. The host system is a Parabola GNU/Linux-libre x86_64 system. Ansible is installed on the host system using the distribution package manager. The version of Ansible used is:

```
$ ansible --version
ansible 2.4.1.0
```

```
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/shakthi/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.14 (default, Sep 20 2017, 01:25:59) [GCC 7.2.0]
```

The */etc/hosts* file should have an entry for the guest "ubuntu" VM as indicated below:

```
192.168.122.4 ubuntu
```

You should be able to issue commands from Ansible to the guest OS. For example:

```
$ ansible ubuntu -m ping
```

```
ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

On the host system, we will create a project directory structure to store the Ansible playbooks:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /playbooks/admin/
```

An 'inventory' file is created inside the *inventory/kvm* folder that contains the following:

```
ubuntu ansible_host=192.168.122.4 ansible_connection=ssh ansible_user=xetex
ansible_password=pass
```

Apache

The Apache Web server needs to be installed first on the Ubuntu guest VM. The Ansible playbook for the same is as follows:

```
- name: Install Apache web server
  hosts: ubuntu
  become: yes
```

```
  become_method: sudo
  gather_facts: true
  tags: [web]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install Apache
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - apache2


    - wait_for:
        port: 80
```

The Ansible playbook updates the software package repository by running *apt-get update* and then proceeds to install the Apache2 package. The playbook waits for the server to start and listen on port 80. An execution of the playbook is shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/piwigo.yml
--tags web -K
SUDO password:

PLAY [Install Apache web server] ******************
TASK [setup] *********************************
ok: [ubuntu]
TASK [Update the software package repository] *************
changed: [ubuntu]

TASK [Install Apache] ************************************ changed: [ubuntu]
=> (item=[u'apache2'])

TASK [wait_for] ************************************************
ok: [ubuntu]
PLAY RECAP *****************************************************
ubuntu                     : ok=4    changed=2    unreachable=0    failed=0
```

The verbosity in the Ansible output can be achieved by passing 'v' multiple times in the invocation. The more number of times that 'v' is present, the greater is the verbosity level. The *-K* option will prompt for the *sudo* password for the *xetex* user. If you now open *http://192.168.122.4,* you should be able to see the default Apache2 *index.html* page as shown in Figure 1.

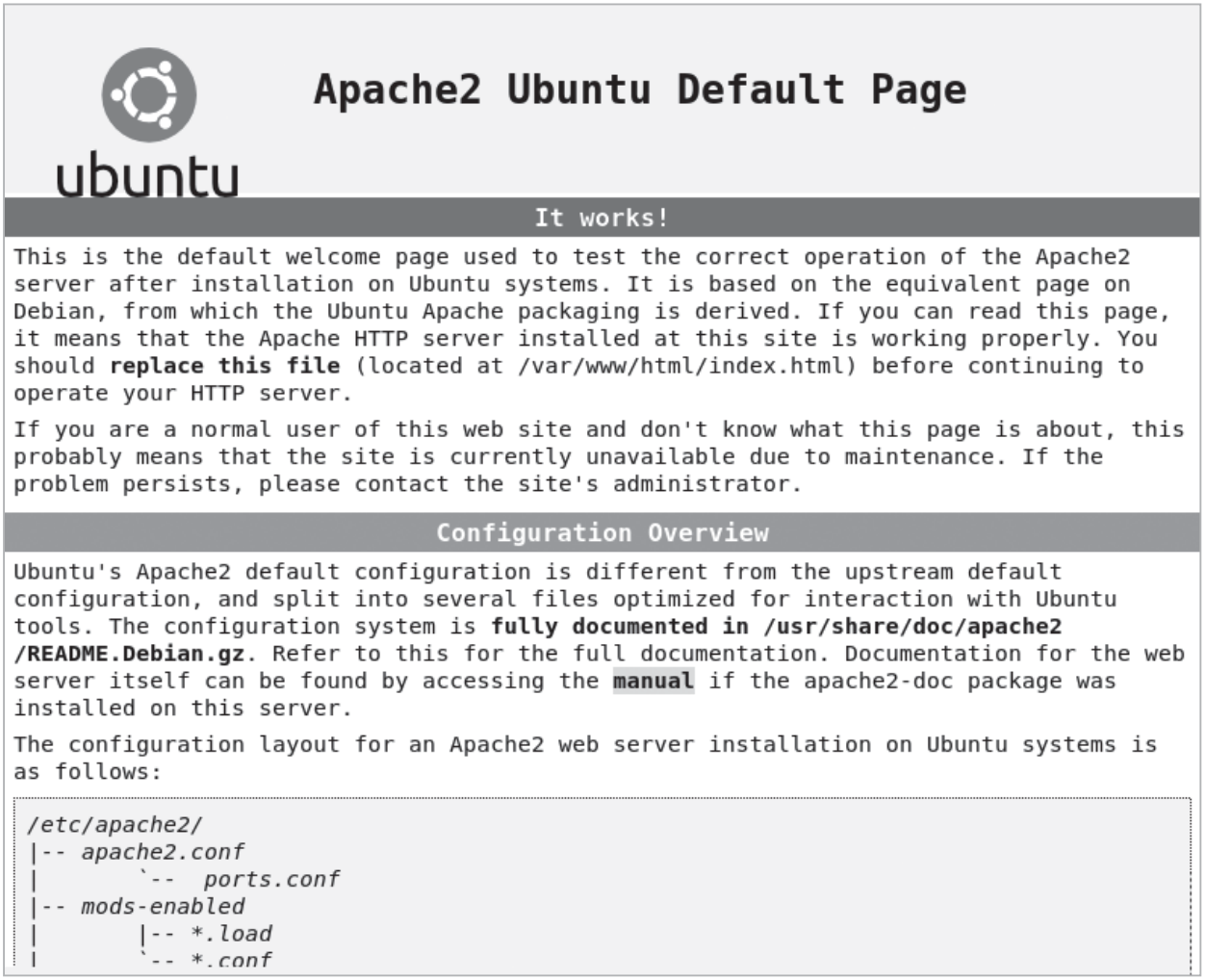


Figure 1: Apache2 default index page

MySQL

Piwigo requires a MySQL database server for its back-end, and at least version 5.0. As the second step, you can install the same using the following Ansible playbook:

```
- name: Install MySQL database server
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
```

```
  tags: [database]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install MySQL
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - mysql-server
        - mysql-client
        - python-mysqldb

    - name: Start the server
      service:
        name: mysql
        state: started

    - wait_for:
        port: 3306

    - mysql_user:
        name: guest
        password: '*F7B659FE10CA9FAC576D358A16CC1BC646762FB2'
        encrypted: yes
        priv: '*.*:ALL,GRANT'
        state: present
```

The APT software repository is updated first and the required MySQL packages are then installed. The database server is started, and the Ansible playbook waits for the server to listen on port 3306. For this example, a *guest* database user account with *osfy* as the password is chosen for the gallery Web application. In production, please use a stronger password. The hash for the password can be computed from the MySQL client as indicated below:

```
mysql> SELECT PASSWORD('osfy');
+-------------------------------------------+
| PASSWORD('osfy')                          |
```

```
+-------------------------------------------+
| *F7B659FE10CA9FAC576D358A16CC1BC646762FB2 |
+-------------------------------------------+
1 row in set (0.00 sec)
```

Also, the default MySQL root password is empty. You should change it after installation. The playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/piwigo.yml
--tags database -K
```

PHP

Piwigo is written using PHP (PHP Hypertext Preprocessor), and it requires at least version 5.0 or later. The documentation website recommends version 5.2. The Ansible playbook to install PHP is given below:

```
- name: Install PHP
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [php]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install PHP
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - php5
- php5-mysql
```

Update the software package repository, and install PHP5 and the php5-mysql database connectivity package. The Ansible playbook for this can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/piwigo.yml
```

```
--tags php -K
```

Piwigo

The final step is to download, install and configure Piwigo. The playbook for this is given below:

```
- name: Setup Piwigo
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [piwigo]

  vars:
    piwigo_dest: "/var/www/html"

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Create a database for piwigo
      mysql_db:
        name: piwigo
        state: present

    - name: Create target directory
      file:
        path: "{{ piwigo_dest }}/gallery"
        state: directory

    - name: Download latest piwigo
      get_url:
        url: http://piwigo.org/download/dlcounter.php?code=latest
        dest: "{{ piwigo_dest }}/piwigo.zip"

    - name: Extract to /var/www/html/gallery
      unarchive:
        src: "{{ piwigo_dest }}/piwigo.zip"
        dest: "{{ piwigo_dest }}/gallery"
        remote_src: True

    - name: Restart apache2 server
```

```
    service:
      name: apache2
      state: restarted
```

The *piwigo_dest* variable stores the location of the default Apache hosting directory. The APT software package repository is then updated. Next, an exclusive MySQL database is created for this Piwigo installation. A target folder *gallery* is then created under */var/www/html* to store the Piwigo PHP files. Next, the latest version of Piwigo is downloaded (2.9.2, as on date) and extracted under the gallery folder. The Apache Web server is then restarted.

You can invoke the above playbook as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/piwigo.yml
--tags piwigo -K
```

If you open the URL *http://192.168.122.4/gallery* in a browser on the host system, you will see the screenshot given in Figure 2 to start the installation of Piwigo.

Figure 2: Piwigo install page

Figure 3: Piwigo install success page

After entering the database credentials and creating an admin user account, you should see the 'success' page, as shown in Figure 3.

You can then go to *http://192.168.122.4/gallery* to see the home page of Piwigo, as shown in Figure 4.

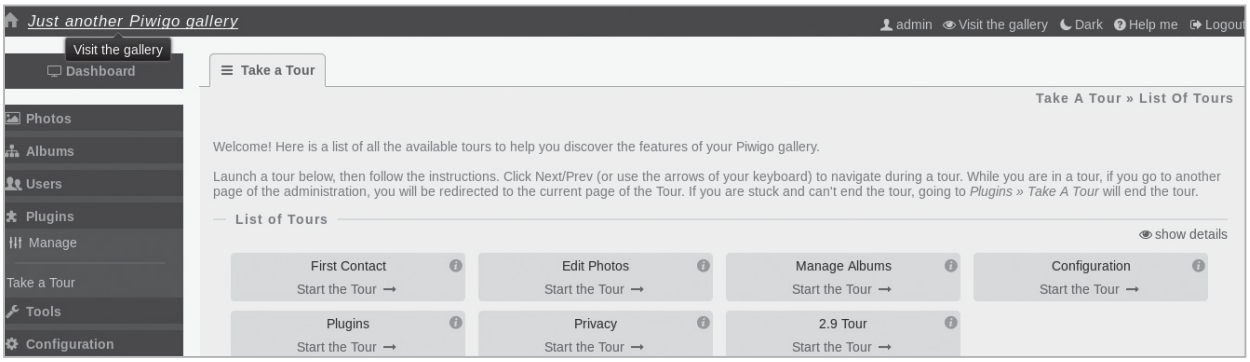


Figure 4: Piwigo home page

Backup

The Piwigo data is present in both the installation folder and in the MySQL database. It is thus important to periodically make backups, so that you can use these archive files to restore data, if required. The following Ansible playbook creates a target backup directory, makes a tarball of the installation folder, and dumps the database contents to a *.sql* file. The epoch timestamp is used in the filename. The backup folder can be rsynced to a different system or to secondary backup.

```
- name: Backup Piwigo
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [backup]

  vars:
    piwigo_dest: "/var/www/html"
```

```
  tasks:
    - name: Create target directory
      file:
        path: "{{ piwigo_dest }}/gallery/backup"
        state: directory

    - name: Backup folder
      archive:
        path: "{{ piwigo_dest }}/gallery/piwigo"
        dest: "{{ piwigo_dest }}/gallery/backup/piwigo-backup-{{ ansible_date_
time.epoch }}.tar.bz2"

    - name: Dump database
      mysql_db:
        name: piwigo
        state: dump
        target: "{{ piwigo_dest }}/gallery/backup/piwigo-{{ ansible_date_time.
epoch }}.sql"
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/piwigo.yml
--tags backup -K
```

Two backup files that were created from executing the above playbook are *piwigo-1510053932.sql* and *piwigo-backup-1510053932.tar.bz2.*

Cleaning up

You can uninstall the entire Piwigo installation using an Ansible playbook. This has to happen in the reverse order. You have to remove Piwigo first, followed by PHP, MySQL and Apache. A playbook to do this is included in the *playbooks/admin* folder and given below for reference:

```
---
- name: Uninstall Piwigo
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [uninstall]

  vars:
    piwigo_dest: "/var/www/html"
```

```
tasks:
  - name: Delete piwigo folder
    file:
      path: "{{ piwigo_dest }}/gallery"
      state: absent

  - name: Drop database
    mysql_db:
      name: piwigo
      state: absent

  - name: Uninstall PHP packages
    package:
      name: "{{ item }}"
      state: absent
    with_items:
      - php5-mysql
      - php5

  - name: Stop the database server
    service:
      name: mysql
      state: stopped

  - name: Uninstall MySQL packages
    package:
      name: "{{ item }}"
      state: absent
    with_items:
      - python-mysqldb
      - mysql-client
      - mysql-server

  - name: Stop the web server
    service:
      name: apache2
      state: stopped

  - name: Uninstall apache2
    package:
      name: "{{ item }}"
      state: absent
```

```
      with_items:
- apache2
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-piwigo.
yml -K
```

You can visit *http://piwigo.org/* for more documentation.

Chapter 12

Deploying Graylog Using Ansible

This 11th article in the DevOps series is a tutorial on installing Graylog software using Ansible.

Graylog is a free and open source log management software that allows you to store and analyse all your logs from a central location. It requires MongoDB (a document-oriented, NoSQL database) to store meta information and configuration information. The actual log messages are stored in Elasticsearch. It is written using the Java programming language and released under the GNU General Public License (GPL) v3.0.

Access control management is built into the software, and you can create roles and user accounts with different permissions. If you already have an LDAP server, its user accounts can be used with the Graylog software. It also provides a REST API, which allows you to fetch data to build your own dashboards. You can create alerts to take actions based on the log messages, and also forward the log data to other output streams. In this article, we will install the Graylog software and its dependencies using Ansible.

GNU/Linux

An Ubuntu 16.04.3 LTS guest virtual machine (VM) instance will be used to set up Graylog using KVM/QEMU. The host system is a Parabola GNU/Linux-libre x86_64 system. Ansible is installed on the host system using the distribution package manager. The version of Ansible used is:

```
$ ansible --version
ansible 2.4.1.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/shakthi/.ansible/plugins/modules', u'/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
```

```
  executable location = /usr/bin/ansible
  python version = 2.7.14 (default, Sep 20 2017, 01:25:59) [GCC 7.2.0]
```

Add an entry to the */etc/hosts* file for the guest 'ubuntu' VM as indicated below:

```
192.168.122.25 ubuntu
```

On the host system, let's create a project directory structure to store the Ansible playbooks:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /playbooks/admin/
```

An 'inventory' file is created inside the *inventory/kvm* folder that contains the following code:

```
ubuntu ansible_host=192.168.122.25 ansible_connection=ssh ansible_user=ubuntu
ansible_password=password
```

You should be able to issue commands using Ansible to the guest OS. For example:

```
$ ansible -i inventory/kvm/inventory ubuntu -m ping
```

```
ubuntu | SUCCESS => {
    "changed": false,
    "failed": false,
    "ping": "pong"
}
```

Pre-requisites

The Graylog software has a few dependency packages that need to be installed as pre-requisites. The APT package repository is updated and

Figure 1: Graylog login page

upgraded before installing the pre-requisite software packages.

```
---
- name: Pre-requisites
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [prerequisite]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Update all the packages
      apt:
        upgrade: dist

    - name: Install pre-requisite packages
      package:
```

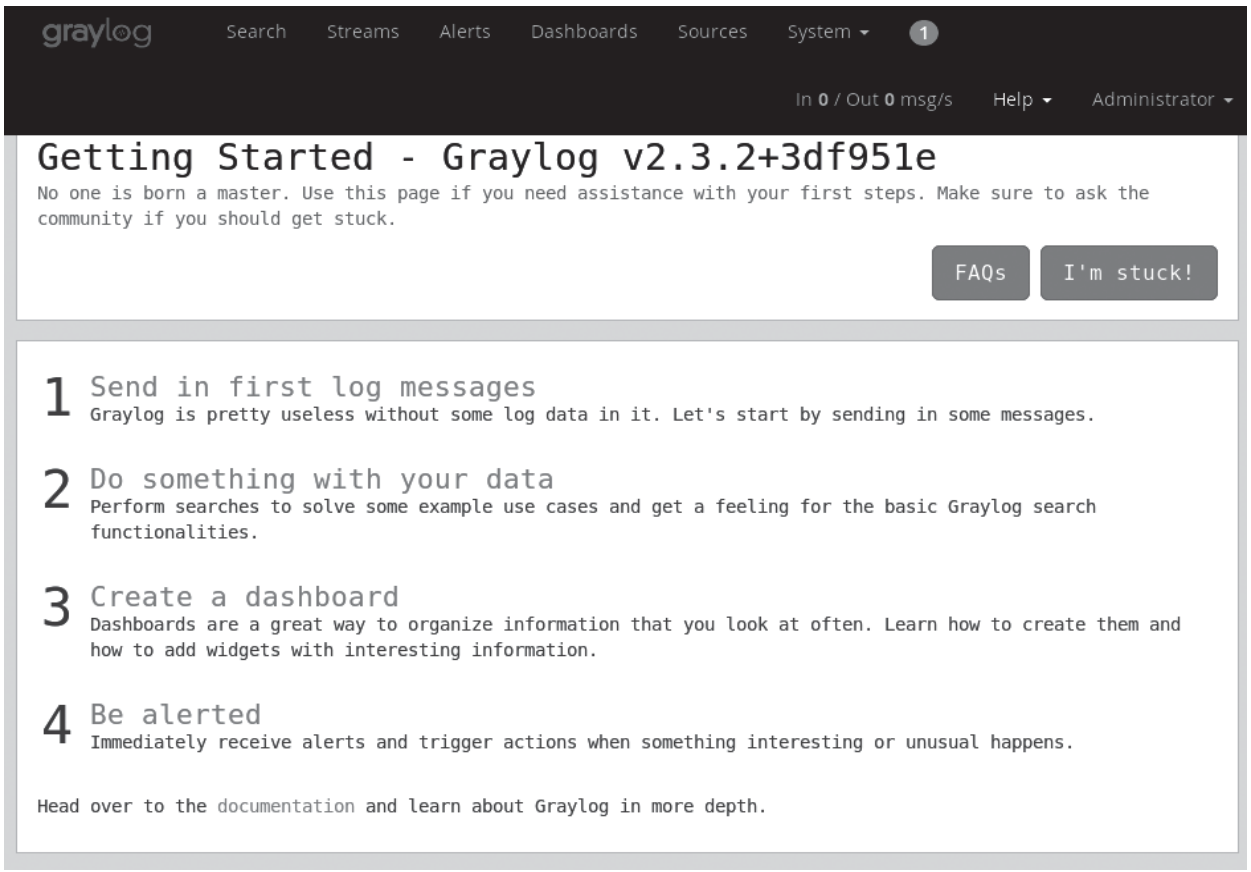


Figure 2: Graylog home page

```
        name: "{{ item }}"
        state: latest
      with_items:
        - apt-transport-https
        - openjdk-8-jre-headless
        - uuid-runtime
- pwgen
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graylog.
yml --tags prerequisite -K
```

The '-K' option prompts for the sudo password for the 'ubuntu' user. You can append multiple '-v' to the end of the playbook invocation to get a more verbose output.

MongoDB

Graylog uses MongoDB to store meta information and configuration changes. The MongoDB software package that ships with Ubuntu 16.04 is supported by the latest Graylog software. The Ansible playbook to install the same is as follows:

```
- name: Install Mongodb
```

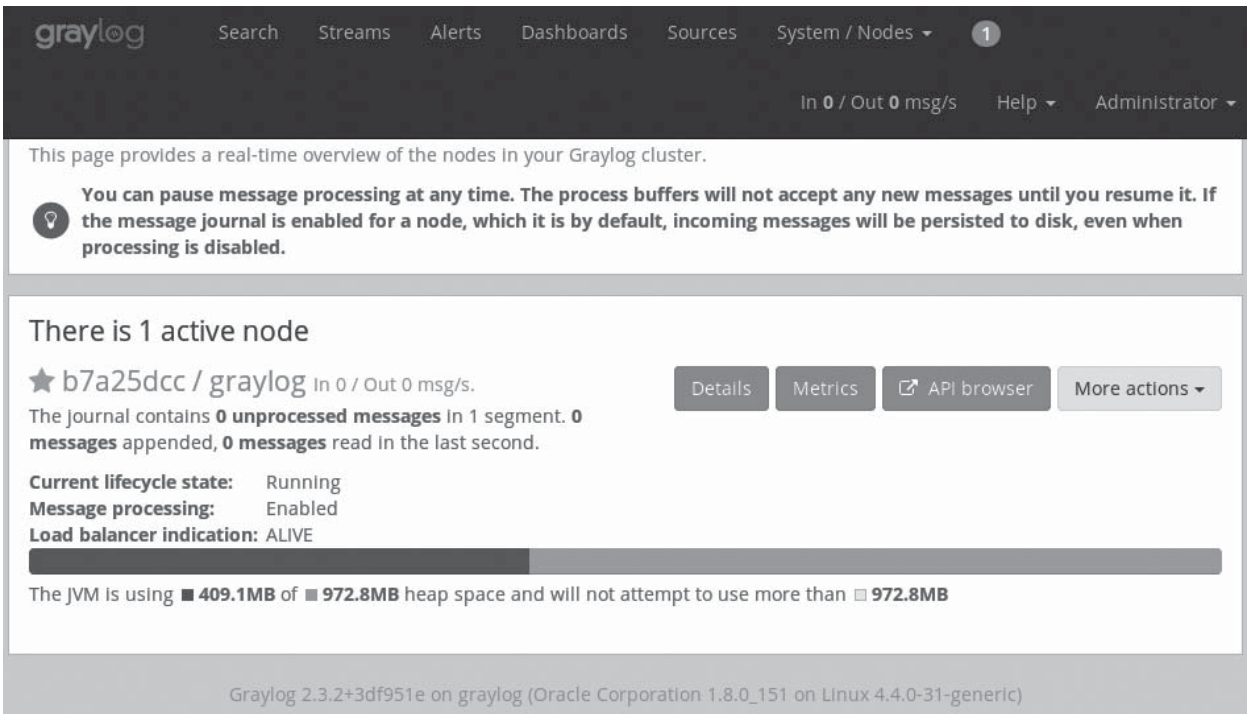


Figure 3: Graylog node activated

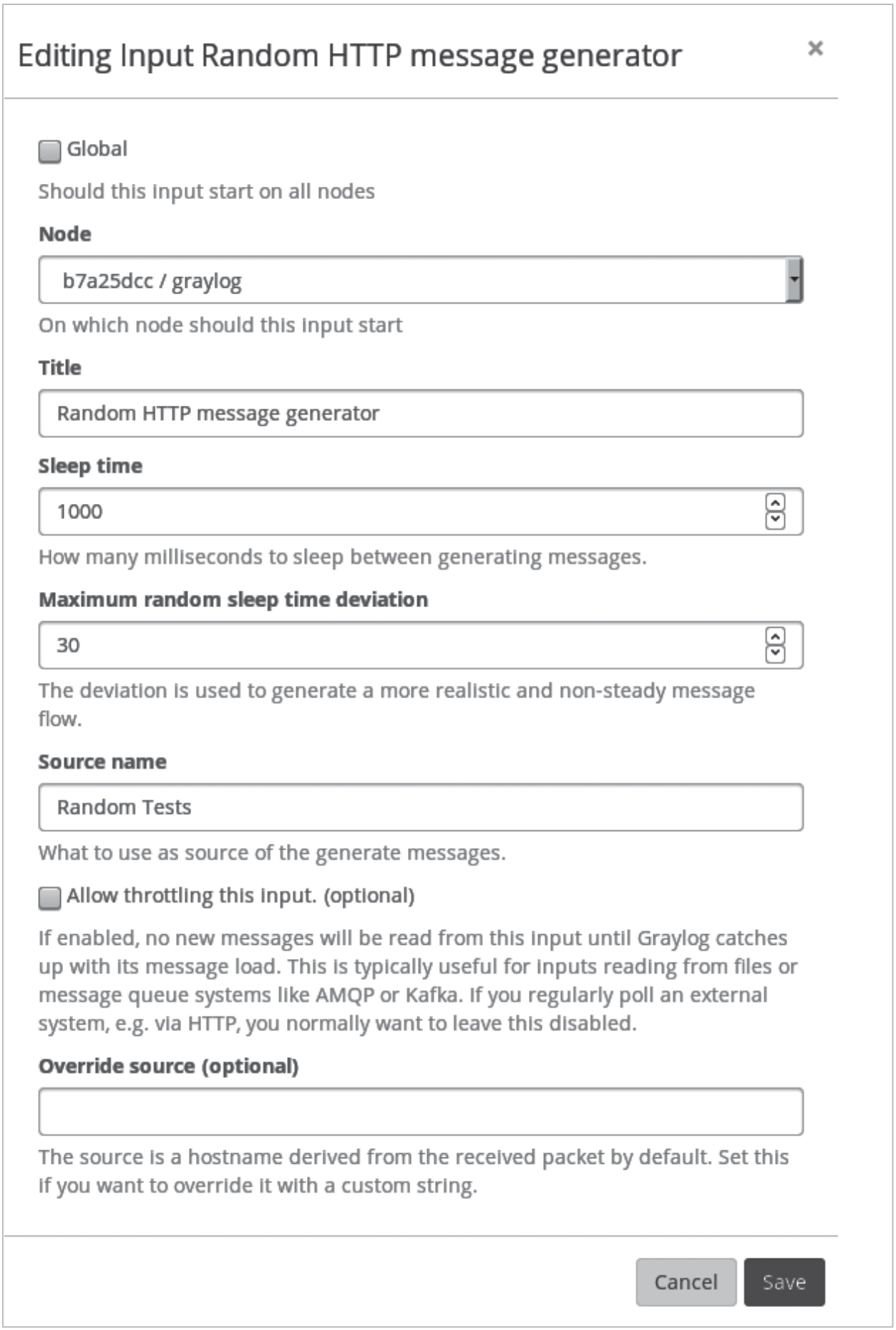


Figure 4: Random HTTP message generator

```
hosts: ubuntu
become: yes
become_method: sudo
gather_facts: true
tags: [mongodb]

tasks:
  - name: Install MongoDB
    package:
      name: mongodb-server
      state: latest

  - name: Start the server
    service:
      name: mongodb
      state: started

  - wait_for:
      port: 27017
```

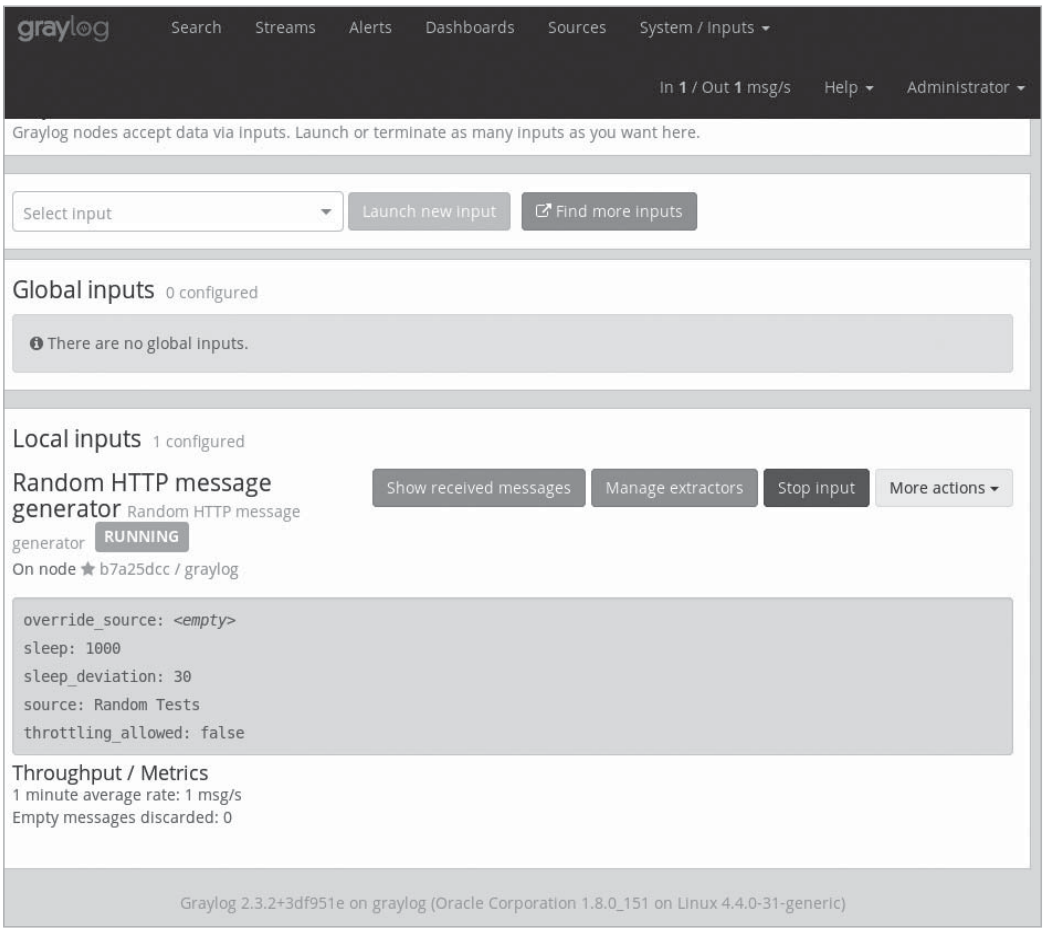


Figure 5: Graylog input random HTTP message generator

The Ubuntu software package for MongoDB is called the 'mongodb-server'. It is installed, and the database server is started. The Ansible playbook waits for the MongoDB server to start and listen on the default port 27017. The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graylog.
yml --tags mongodb -K
```

Elasticsearch

Elasticsearch is a search engine that is written in Java and released under the Apache licence. It is based on Lucene (an information retrieval software library) and provides a full-text search feature. The *elastic.co* website provides *.deb* packages that can be used to install the same on Ubuntu. The Ansible playbook for this is provided below:

```
- name: Install Elasticsearch
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [elastic]

  tasks:
    - name: Add key
      apt_key:
        url: https://artifacts.elastic.co/GPG-KEY-elasticsearch
        state: present

    - name: Add elastic deb sources
      lineinfile:
        path: /etc/apt/sources.list.d/elastic-5.x.list
        create: yes
        line: 'deb https://artifacts.elastic.co/packages/5.x/apt stable main'

    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install Elasticsearch
      package:
        name: elasticsearch
```

```
      state: latest

  - name: Update cluster name
    lineinfile:
      path: /etc/elasticsearch/elastisearch.yml
      create: yes
      regexp: '^#cluster.name: my-application'
      line: 'cluster.name: graylog'

  - name: Daemon reload
    systemd: daemon_reload=yes

  - name: Start elasticsearch service
    service:
      name: elasticsearch.service
      state: started

  - wait_for:
```

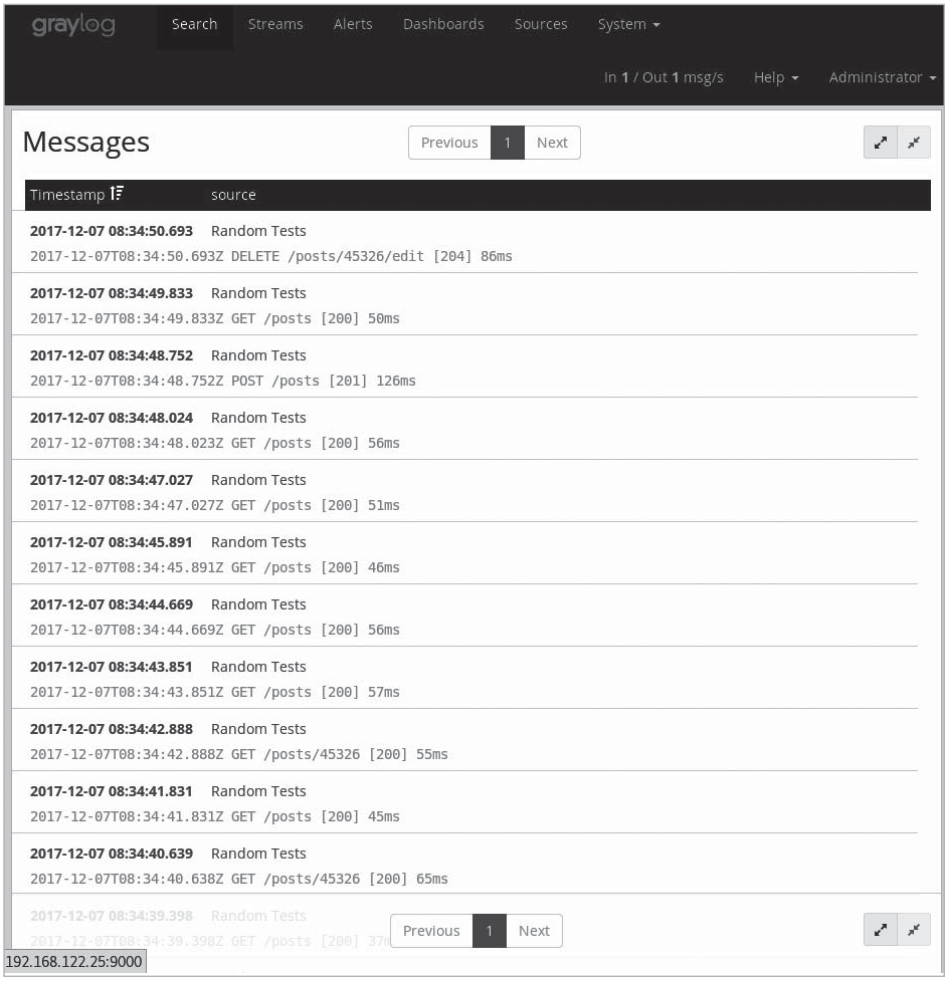


Figure 6: Graylog random HTTP messages

```
        port: 9200
```

```
    - name: Test Curl query
      shell: curl -XGET 'localhost:9200/?pretty'
```

The stable *elastic.co* repository package is installed before installing Elasticsearch. The cluster name is then updated in the */etc/elasticsearch/elasticsearch.yml* configuration file. The system daemon services are reloaded, and the Elasticsearch service is started. The Ansible playbook waits for the service to run and listen on port 9200.

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graylog.
yml --tags elastic -K
```

You can perform a manual query to verify that Elasticsearch is running using the following Curl command:

```
$ curl -XGET 'localhost:9200/?pretty'
```

```
{
  "name" : "cFn-3YD",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "nuBTSlFBTk6PDGyrfDCr3A",
  "version" : {
    "number" : "5.6.5",
    "build_hash" : "6a37571",
    "build_date" : "2017-12-04T07:50:10.466Z",
    "build_snapshot" : false,
    "lucene_version" : "6.6.1"
  },
  "tagline" : "You Know, for Search"
}
```

Graylog

The final step is to install Graylog itself. The *.deb* package available from the *graylog2.org* website is installed and then the actual 'graylog-server' package is installed. The configuration file is updated with credentials for the 'admin' user with a hashed string for the password 'osfy'. The Web interface is also enabled with the default IP address of the guest VM. The Graylog service is finally started. The Ansible playbook to install Graylog is as follows:

```
- name: Install Graylog
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [graylog]

  tasks:
    - name: Install Graylog repo deb
      apt:
        deb: https://packages.graylog2.org/repo/packages/graylog-2.3-
repository_latest.deb

    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install Graylog
      package:
        name: graylog-server
        state: latest

    - name: Update database credentials in the file
      replace:
        dest: "/etc/graylog/server/server.conf"
        regexp: "{{ item.regexp }}"
        replace: "{{ item.replace }}"
      with_items:
        - { regexp: 'password_secret =', replace: 'password_secret =
QXHg3Eqvsu PmFxUY2aKlgimUF05plMPXQ Hy1stUiQ1uaxgIG27 K3t2MviRiFLNot09U1ako
T30njK3G69KIzqIoYqdY3oLUP' }
        - { regexp: '#root_username = admin', replace: 'root_username = admin'
}
        - { regexp: 'root_password_sha2 =', replace: 'root_password_sha2 =
eabb9bb2efa089223 d4f54d55bf2333ebf04a29094bff00753536d7488629399'}
        - { regexp: '#web_enable = false', replace: 'web_enable = true' }
        - { regexp: '#web_listen_uri = http://127.0.0.1:9000/', replace: "web_
listen_uri = http://{{ ansible_default_ipv4.address }}:9000/" }
        - { regexp: 'rest_listen_uri = http://127.0.0.1:9000/api/', replace:
"rest_listen_uri = http://{{ ansible_default_ipv4.address }}:9000/api/" }

    - name: Start graylog service
```

```
    service:
      name: graylog-server.service
      state: started
```

The above playbook can be run using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/graylog.
yml --tags graylog -K
```

Web interface

You can now open the URL *http://192.168.122.25:9000* in a browser on the host system to see the default Graylog login page as shown in Figure 1.

The user name is 'admin' and the password is 'osfy'. You will then be taken to the Graylog home page as shown in Figure 2.

The guest VM is a single node, and hence if you traverse to *System -> Nodes*, you will see this node information as illustrated in Figure 3.

You can now test the Graylog installation by adding a data source as input by traversing *System -> Input* in the Web interface. The 'random HTTP message generator' is used as a local input, as shown in Figure 4.

The newly created input source is now running and visible as a local input in the Web page as shown in Figure 5.

After a few minutes, you can observe the created messages in the *Search* link as shown in Figure 6.

Uninstalling Graylog

An Ansible playbook to stop the different services, and to uninstall Graylog and its dependency software packages, is given below for reference:

```
---
- name: Uninstall Graylog
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [uninstall]

  tasks:
    - name: Stop the graylog service
      service:
        name: graylog-server.service
        state: stopped

    - name: Uninstall graylog server
```

```
      package:
        name: graylog-server
        state: absent

    - name: Stop the Elasticsearch server
      service:
        name: elasticsearch.service
        state: stopped

    - name: Uninstall Elasticsearch
      package:
        name: elasticsearch
        state: absent

    - name: Stop the MongoDB server
      service:
        name: mongodb
        state: stopped

    - name: Uninstall MongoDB
      package:
        name: mongodb-server
        state: absent

    - name: Uninstall pre-requisites
      package:
        name: "{{ item }}"
        state: absent
      with_items:
        - pwgen
        - uuid-runtime
        - openjdk-8-jre-headless
        - apt-transport-https
```

The above playbook can be invoked using:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-
graylog.yml -K
```

Chapter 13

Ansible Deployment of Nginx with SSL

This is the 12th article in the DevOps series. It is a tutorial on installing Nginx with SSL. Nginx is a high performance Web server and can be used as a load balancer.

Nginx is a Web server written in C by Igor Sysoev. It can be used as a load balancer, reverse proxy and HTTP cache server. Nginx was designed to handle over 10,000 client connections and has support for TLS (transport layer security) and SSL (secure sockets layer). It requires a very low memory footprint and is IPv6-compatible. Nginx can also be used as a mail server proxy. It was first released in 2004 under a BSD-like licence.

The OpenSSL project provides a free and open source software security library that implements the SSL and TLS protocols. This library is used by applications to secure communication between machines in a computer network. The library is written in C and Assembly, and uses a dual-licence — Apache License 1.0 and a four-clause BSD licence. The library implements support for a number of ciphers and cryptographic functions. It was first released in 1998 and is widely used in Internet Web servers.

An Ubuntu 16.04.1 LTS guest virtual machine (VM) instance using KVM/QEMU is chosen to install Nginx.

```
$ cat /etc/lsb-release
DISTRIB_ID=Ubuntu
DISTRIB_RELEASE=16.04
DISTRIB_CODENAME=xenial
DISTRIB_DESCRIPTION="Ubuntu 16.04.1 LTS"
```

The default installation on the guest VM does not come with Python2, and hence you need to install this on the guest machine, manually, as shown below:

```
$ sudo apt-get update
$ sudo apt-get install python-minimal
```

The host system is a Parabola GNU/Linux-libre x86_64 system, and Ansible is installed on the host system using the distribution package manager. The version of Ansible used is 2.4.2.0 as indicated below:

```
$ ansible --version
ansible 2.4.2.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/shakthi/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /bin/ansible
  python version = 2.7.14 (default, Sep 20 2017, 01:25:59) [GCC 7.2.0]
```

You should add an entry to the */etc/hosts* file for the guest Ubuntu VM, as follows:

```
192.168.122.244 ubuntu
```

On the host system, let's create a project directory structure to store the Ansible playbooks, inventory and configuration files, as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /playbooks/admin/
       /files/
```

An 'inventory' file is created inside the *inventory/kvm* folder that contains the following:

```
ubuntu ansible_host=192.168.122.244 ansible_connection=ssh ansible_user=ubuntu
ansible_password=password
```

You should now be able to issue commands to the guest OS, using Ansible. For example:

```
$ ansible -i inventory/kvm/inventory ubuntu -m ping
ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Installing Nginx

The Nginx software package in Ubuntu can be installed on the guest machine. The APT package repository is first updated before installing the Nginx Web server. The Uncomplicated Firewall (UFW) is then used to enable both HTTP and HTTPS access on the guest OS. The Web server is then started, and the playbook waits for the server to listen on port 80. The Ansible playbook is provided below for reference:

```
---
- name: Install nginx
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [nginx]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install nginx
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - nginx

    - name: Allow Nginx Full
      ufw:
        rule: allow
        name: Nginx Full
        state: enabled

    - name: Allow Nginx Full
      ufw:
        rule: allow
        name: OpenSSH
        state: enabled

    - name: Start nginx
      service:
        name: nginx
```

```
        state: started

    - wait_for:
        port: 80
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx-ssl.
yml --tags nginx -K
```

The *-K* option prompts for the sudo password of the Ubuntu user. You can append multiple *-v* to the end of the playbook invocation to get a more verbose output.

If you open a browser on the host system with the URL *http://192.168.122.244*, you should see the default Nginx home page as shown in Figure 1.

Figure 1: Nginx home page

Generating SSL certificates

The required SSL certificates need to be created using Ansible. The OpenSSL and Python-openssl packages are installed after updating the APT software repository in Ubuntu. An OpenSSL private key is generated in the */etc/ssl/private/ansible.com.pem* file. The */etc/ssl/csr* directory is created before generating the OpenSSL certificate signing request (CSR) with the required certificate parameters in the */etc/ssl/csr/www.ansible.com.csr* file. The actual self-signed certificate is then generated as shown in the following playbook:

```
- name: Create SSL certificates
  hosts: ubuntu
  become: yes
  become_method: sudo
```

```
  gather_facts: true
  tags: [ssl]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install openssl
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - openssl
        - python-openssl

    - name: Generate an OpenSSL private key
      openssl_privatekey:
        path: /etc/ssl/private/ansible.com.pem

    - name: Create directory
      file:
        path: /etc/ssl/csr
        state: directory
        mode: 0755

    - name: Generate an OpenSSL Certificate Signing Request
      openssl_csr:
        path: /etc/ssl/csr/www.ansible.com.csr
        privatekey_path: /etc/ssl/private/ansible.com.pem
        country_name: IN
        organization_name: Ansible
        email_address: author@shakthimaan.com
        common_name: www.ansible.com

    - name: Generate a self signed certificate
      openssl_certificate:
        path: /etc/ssl/certs/nginx-selfsigned.crt
        privatekey_path: /etc/ssl/private/ansible.com.pem
        csr_path: /etc/ssl/csr/www.ansible.com.csr
        provider: selfsigned
```

The above playbook can be run as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx-ssl.
yml --tags ssl -K
```

Configuring Nginx for SSL

The final step is to configure Nginx to use SSL. A *self-signed.conf* file is created in the */etc/nginx/snippets* folder that contains the following:

```
ssl_certificate /etc/ssl/certs/nginx-selfsigned.crt;
ssl_certificate_key /etc/ssl/private/ansible.com.pem;
```

The SSL parameter configurations are stored in the */etc/nginx/snippets/ssl-params.conf* file as shown below:

```
# from https://cipherli.st/
# and https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html

ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
ssl_prefer_server_ciphers on;
ssl_ciphers “EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH”;
ssl_ecdh_curve secp384r1;
ssl_session_cache shared:SSL:10m;
ssl_session_tickets off;
ssl_stapling on;
ssl_stapling_verify on;
resolver 8.8.8.8 8.8.4.4 valid=300s;
resolver_timeout 5s;
# Disable preloading HSTS for now.  You can use the commented out header line
that includes
# the “preload” directive if you understand the implications.
#add_header Strict-Transport-Security “max-age=63072000; includeSubdomains;
preload”;
add_header Strict-Transport-Security “max-age=63072000; includeSubdomains”;
add_header X-Frame-Options DENY;
add_header X-Content-Type-Options nosniff;
```

The Nginx Web server configuration for this host (*google.com*, for example) is then created in the */etc/nginx/sites-enabled* folder with the following contents:

```
server {
```

```
        listen 80;
        root /var/www/html;
        index index.nginx-debian.html;

        server_name google.com www.google.com;
}
server {
        listen 443 ssl http2 default_server;
        include snippets/self-signed.conf;
        include snippets/ssl-params.conf;
}
```

The Ansible playbook for configuring Nginx with SSL is as follows:

```
- name: Setup nginx with SSL
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [https]

  tasks:
    - copy:
        src: ../../files/self-signed.conf
        dest: /etc/nginx/snippets/self-signed.conf
        owner: root
        group: root
        mode: 0644

    - copy:
        src: ../../files/ssl-params.conf
        dest: /etc/nginx/snippets/ssl-params.conf
        owner: root
        group: root
        mode: 0644

    - copy:
        src: ../../files/google.com
        dest: /etc/nginx/sites-enabled/google.com
        owner: root
        group: root
        mode: 0644
```

```
    - name: Restart nginx
      service:
        name: nginx
        state: restarted

    - wait_for:
        port: 443
```

The above playbook can be executed as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx-ssl.
yml --tags https -K
```

You can now open *https://192.168.122.244* in a browser on the host system, and view the self-signed certificate as shown in Figure 2.

After accepting the certificate, you will be able to see the default Nginx home page as shown in Figure 3.

You can also use the *curl* command to view the home page from the command line, as follows:

```
$ curl https://192.168.122.244 -k
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
```

```
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Validation

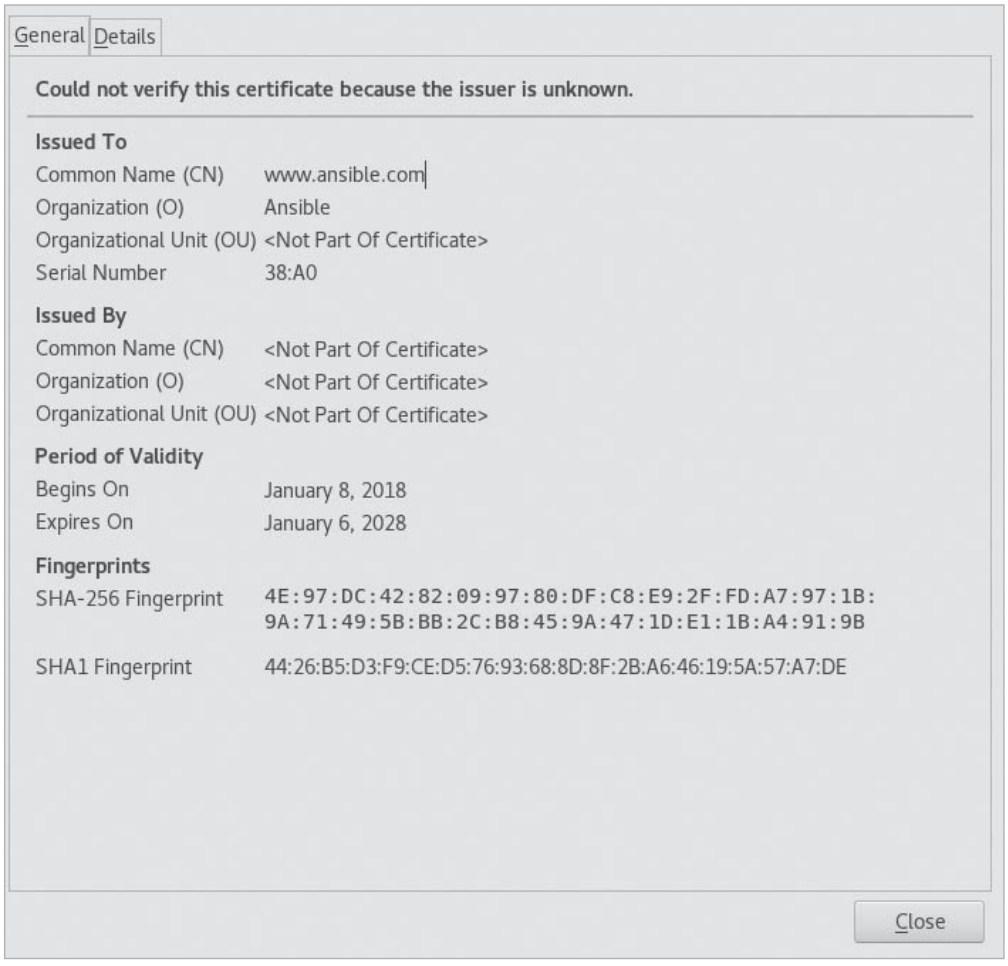


Figure 2: SSL certificate

Figure 3: Nginx HTTPS home page

You can run a number of validation checks on the SSL certificate, periodically, to ascertain that it still holds good and meets your requirements. A few examples of sanity checks that you can perform on the certificate are shown below for reference:

```
- name: Validate SSL certificate
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [validate]

  tasks:
    - name: Certificate matches with the private key
      openssl_certificate:
        path: /etc/ssl/certs/nginx-selfsigned.crt
        privatekey_path: /etc/ssl/private/ansible.com.pem
        provider: assertonly

    - name: Certificate can be used for digital signatures
      openssl_certificate:
        path: /etc/ssl/certs/nginx-selfsigned.crt
        provider: assertonly
        key_usage:
          - digitalSignature
        key_usage_strict: true

    - name: Certificate uses a recent signature algorithm (no SHA1, MD5 or DSA)
      openssl_certificate:
        path: /etc/ssl/certs/nginx-selfsigned.crt
        provider: assertonly
        signature_algorithms:
          - sha224WithRSAEncryption
          - sha256WithRSAEncryption
          - sha384WithRSAEncryption
          - sha512WithRSAEncryption
          - sha224WithECDSAEncryption
          - sha256WithECDSAEncryption
          - sha384WithECDSAEncryption
          - sha512WithECDSAEncryption

    - name: Certificate matches the domain
      openssl_certificate:
```

```
      path: /etc/ssl/certs/nginx-selfsigned.crt
      provider: assertonly
      subject_alt_name:
        - DNS:www.ansible.com
```

```
  - name: Certificate is valid for another month (30 days) from now
    openssl_certificate:
      path: /etc/ssl/certs/nginx-selfsigned.crt
      provider: assertonly
      valid_in: 2592000
```

You can invoke the above validation checks in the playbook using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx-ssl.
yml --tags validate -K
```

Uninstalling

An *uninstall* playbook is provided in the *playbooks/admin/uninstall-nginx.yml* file to stop the Nginx Web server, disable access to the port in the firewall, and to remove the software from the guest VM:

```
---
- name: Uninstall Nginx
 hosts: ubuntu
 become: yes
 become_method: sudo
 gather_facts: true
 tags: [server]
```

```
 tasks:
   - name: Stop the web server
     service:
       name: nginx
       state: stopped
```

```
   - name: Disable Nginx Full
     ufw:
       rule: deny
       name: Nginx Full
       state: enabled
```

```
   - name: Uninstall apache2
```

```
    package:
      name: "{{ item }}"
      state: absent
    with_items:
      - nginx
```

The above playbook can be run as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/admin/uninstall-nginx.
yml -K
```

You can verify the firewall status using the *ufw* command, as shown below:

```
$ sudo ufw status
Status: active

To                         Action      From
--                         ------      ----
Nginx Full                 DENY        Anywhere
OpenSSH                    ALLOW       Anywhere
Nginx Full (v6)            DENY        Anywhere (v6)
OpenSSH (v6)               ALLOW       Anywhere (v6)
```

Please refer to the Nginx documentation website (*https://nginx.org/en/docs/*) for more information.

Chapter 14

Ansible Deployment of the Aerospike NoSQL Database

In this 13th article in the DevOps series, we will install and set up an Aerospike NoSQL database cluster.

Aerospike is a free and open source, distributed NoSQL database written in C. It is primarily designed for high throughput and low latency applications. It is a key-value store optimised for Flash storage devices and runs on *nix platforms. It can also be used as an in-memory database residing entirely on RAM. Following the CAP (Consistency, High Availability, Partition Tolerance) theorem, at present, the database can be operated in AP mode. The database functions in three layers — the data storage layer, the clustering and distribution layer, and the client layer. As in many distributed databases, the server supports rolling upgrades and failover mechanisms in the cluster.

A number of client libraries for various programming language environments like C, Java, Go, Python, Node.js, etc, are available to create applications that can communicate with the Aerospike server. The basic data types that are supported are integer, string, bytes, double, list, map, GeoJSON and blobs (language serialised). Support for user defined functions (UDF) also exists, using the Lua programming language. The Aerospike tooling ecosystem has a number of connectors for the Spring framework, Hadoop, Kafka, etc. The Aerospike Management Console (AMC) is a Web-based application used to monitor the Aerospike cluster. The Aerospike server was first launched in 2012, and is released under the Affero General Public License.

GNU/Linux

A host system running Ubuntu 16.04.3 LTS (Xenial) with Vagrant and VirtualBox is used to launch three CentOS 6.9 virtual machines, on

which the Aerospike server will be installed and set up. The Vagrant file to launch the three nodes is shown below:

```
BOX_IMAGE  = “bento/centos-6.9”
NODE_COUNT = 3
Vagrant.configure(“2”) do |config|
 (1..NODE_COUNT).each do |i|
   # Create Aerospike server node
   config.vm.define “as#{i}” do |node|
     node.vm.box = BOX_IMAGE
     node.ssh.forward_agent = true
     # https://github.com/mitchellh/vagrant/issues/4967
     node.ssh.insert_key = false
     node.ssh.username = ‘vagrant’
     node.ssh.password = ‘vagrant’
     node.vm.network :private_network, ip: “10.0.0.#{i + 10}”
     node.vm.hostname = “as#{i}”
     node.vm.provider “virtualbox” do |vb|
       vb.name = “Aerospike-server-#{i}”
       vb.memory = “2048”
     end
   end
 end
end
```

The three instances are named ‘as1’, ‘as2’ and ‘as3’, and are assigned private IP addresses that can be accessed from the host system. The machines are brought up using the following command:

```
$ vagrant up
```

The version of Ansible used on the host system is 2.2.0.0. A project directory structure is created to store the Ansible playbooks and inventory, as follows:

```
ansible/inventory/vbox/
       /playbooks/configuration/
```

An ‘inventory’ file is created inside the *inventory/vbox* folder, which contains access information for each node:

```
[nodes]
```

```
as1 ansible_host=10.0.0.11 ansible_connection=ssh ansible_user=vagrant ansible_
ssh_private_key_file=~/.vagrant.d/insecure_private_key
as2 ansible_host=10.0.0.12 ansible_connection=ssh ansible_user=vagrant ansible_
ssh_private_key_file=~/.vagrant.d/insecure_private_key
as3 ansible_host=10.0.0.13 ansible_connection=ssh ansible_user=vagrant ansible_
ssh_private_key_file=~/.vagrant.d/insecure_private_key
```

We also create *'inventory/vbox/group_vars/all/all.yml'* with the following variables:

```
---
aerospike_edition: "community"
aerospike_version: "latest"

amc_version: 3.6.13
```

Installing Aerospike

The entire set of playbooks is available in the *playbooks/configuration/aerospike.yml* file. The first step is to install the Aerospike server on all the three nodes using Ansible, as shown below:

```
---
- name: Configure Aerospike cluster node
 hosts: nodes
 become: yes
 become_method: sudo
 gather_facts: true
 tags: [server]

 tasks:
   - name: Create downloads directory
     file: path="/home/{{ ansible_user }}/downloads" state=directory

   - name: Create installs directory
     file: path="/home/{{ ansible_user }}/installs" state=directory

   - name: Download Aerospike community server (RPM)
     get_url: url="http://aerospike.com/download/server/{{ aerospike_version
}}/artifact/el6" dest="/home/{{ ansible_user }}/downloads/"
     register: __community_server_file

   - set_fact:
```

```
      server_version: "{{ __community_server_file.dest | basename }}"

  - name: Extract Aerospike server
    command: tar xzvf "/home/{{ ansible_user }}/downloads/{{ server_version
}}" -C installs

  - name: Yum update
    yum: name=* update_cache=yes state=present

  - name: Install RPM dependencies
    become: yes
    become_method: sudo
    yum:
      name: "{{ item }}"
      state: latest
    with_items:
      - python
      - libselinux-python

  - name: Run asinstall
    shell: cd "/home/{{ ansible_user }}/installs/{{ server_version | regex_
replace('\.tgz', '') }}" && sudo ./asinstall

  - name: Start Aerospike server (RPM)
    shell: sudo service aerospike start

  - name: Wait for server to be up and running
    wait_for:
      port: 3000
      delay: 5
      state: started
```

The 'downloads' and 'installs' directories are first created. The Aerospike server package for CentOS 6 is then downloaded and extracted. The *YUM update* command is used to update the software repository, following which the dependencies and the Aerospike server are installed. The database server is then started, and the playbook waits for the server to listen on port 3000. The above playbook can be run as follows:

```
$ ansible-playbook -i inventory/vbox/inventory playbooks/configuration/
aerospike.yml --tags server
```

Setting up a multicast cluster

The Aerospike cluster can be set up either in multi-cast or mesh mode. The following playbook configures the cluster in multicast mode:

```
- name: Create multicast cluster
 hosts: nodes
 gather_facts: true
 sudo: yes
 tags: [multicast]

 vars:
   ip_address: "{{ ansible_eth1['ipv4']['address'] }}"
   line_address: "                address {{ ip_address }}"

 tasks:
   - name: Stop aerospike server
     command: service aerospike stop

   - name: Specify multicast configuration file parameters
     lineinfile:
       dest: /etc/aerospike/aerospike.conf
       insertafter: "{{ item.insertafter }}"
       line: "{{ item.line }}"
       state: present
     with_items:
       - { insertafter: 'proto-fd-max 15000', line: "        node-id-interface
eth1" }
       - { insertafter: 'port 9918', line: "{{ line_address }}"  }
       - { insertafter: 'fabric', line: "{{ line_address }} # Multicast:
Specify private IP address"  }

   - name: Update network service address
     replace:
       name: /etc/aerospike/aerospike.conf
       regexp: 'address any'
       replace: "address {{ ansible_eth1['ipv4']['address'] }}"

   - name: Start aerospike server
     command: service aerospike start
```

The above playbook can be executed using the following command:

```
$ ansible-playbook -i inventory/vbox/inventory playbooks/configuration/
aerospike.yml --tags multicast
```

You can verify the multi-cast cluster setup by logging into one of the virtual machines, and using the Aerospike Admin (asadm) tool as illustrated below:

```
[vagrant@as1 ~]$ asadm
Aerospike Interactive Shell, version 0.1.15

Found 3 nodes
Online:  10.0.0.11:3000, 10.0.0.13:3000, 10.0.0.12:3000

Admin> exit
[vagrant@as1 ~]$
```

Installing the Aerospike Management Console

The Aerospike Management Console (AMC) dashboard provides a graphical user interface (GUI) for the Aerospike server. Its installation playbook is as follows:

```
- name: Install Aerospike Management Console
 hosts: nodes
 become: yes
 become_method: sudo
 gather_facts: true
 tags: [amc]

 tasks:

   - name: Download AMC Community RPM package
     get_url: url="http://www.aerospike.com/download/amc/{{ amc_version }}/
artifact/el6" dest="/home/{{ ansible_user }}/downloads/"
     register: __community_amc_file

   - set_fact:
       amc_file: "{{ __community_amc_file.dest }}"

   - name: Install RPM dependencies
     yum:
       name: "{{ item }}"
       state: latest
```

```
    with_items:
      - gcc
      - python-devel

  - name: Install RPM amc
    command: rpm -ivh "{{ amc_file }}"
  - name: Start AMC server (RPM)
    shell: sudo service amc start
```

The invocation of the above playbook and a sample execution run is shown below:

```
$ ansible-playbook -i inventory/vbox/inventory playbooks/configuration/
aerospike.yml --tags amc
PLAY [Configure Aerospike cluster node] ***********************
TASK [setup] ******************************************************
ok: [as2]
ok: [as1]
ok: [as3]

PLAY [Create multicast cluster] ******************************

TASK [setup] ******************************************************
ok: [as1]
ok: [as2]
ok: [as3]

PLAY [Install Aerospike Management Console] *****************
TASK [setup] ******************************************************
ok: [as1]
ok: [as2]
ok: [as3]

TASK [Download AMC Community RPM package] *******************
changed: [as2]
changed: [as1]
changed: [as3]

TASK [set_fact] ***************************************************
ok: [as1]
ok: [as2]
ok: [as3]
```

```
TASK [Install RPM dependencies] ****************************
changed: [as1] => (item=[u'gcc', u'python-devel'])
changed: [as2] => (item=[u'gcc', u'python-devel'])
changed: [as3] => (item=[u'gcc', u'python-devel'])

TASK [Install RPM amc] *************************************
changed: [as1]
changed: [as2]
changed: [as3]

TASK [Start AMC server (RPM)] ******************************
changed: [as1]
changed: [as2]
changed: [as3]

PLAY RECAP *************************************************
as1          : ok=8    changed=4    unreachable=0    failed=0
as2          : ok=8    changed=4    unreachable=0    failed=0
as3          : ok=8    changed=4    unreachable=0    failed=0
```

You can now open *http://10.0.0.1:8081* in a browser on the host system, and you will be prompted with a modal window, as shown in Figure 1.

You can input 'localhost' for the 'Host Name', and the AMC dashboard opens as shown in Figure 2.

Uninstalling Aerospike

An uninstall playbook is written to stop the services, and uninstalls both the Aerospike Management Console (AMC) and the Aerospike server, as follows:

```
- name: Uninstall
 hosts: nodes
 become: yes
 become_method: sudo
 gather_facts: true
 tags: [uninstall]

 tasks:
   - name: Stop AMC server
     service:
       name: amc
       state: stopped
```

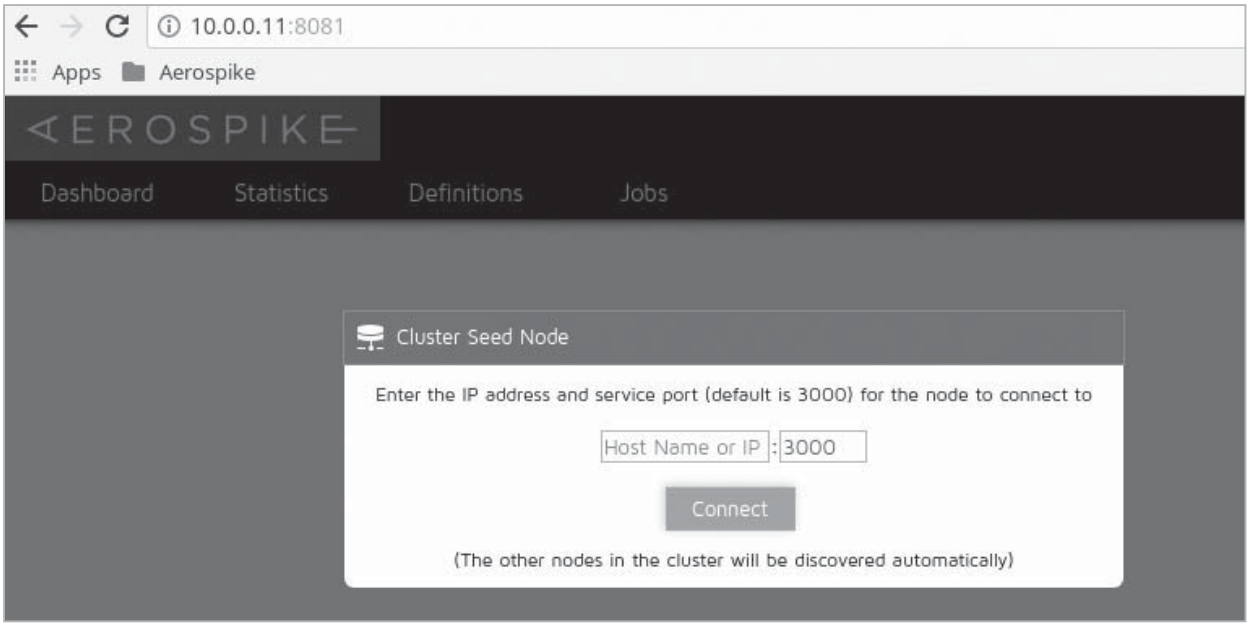


Figure 1: AMC modal window

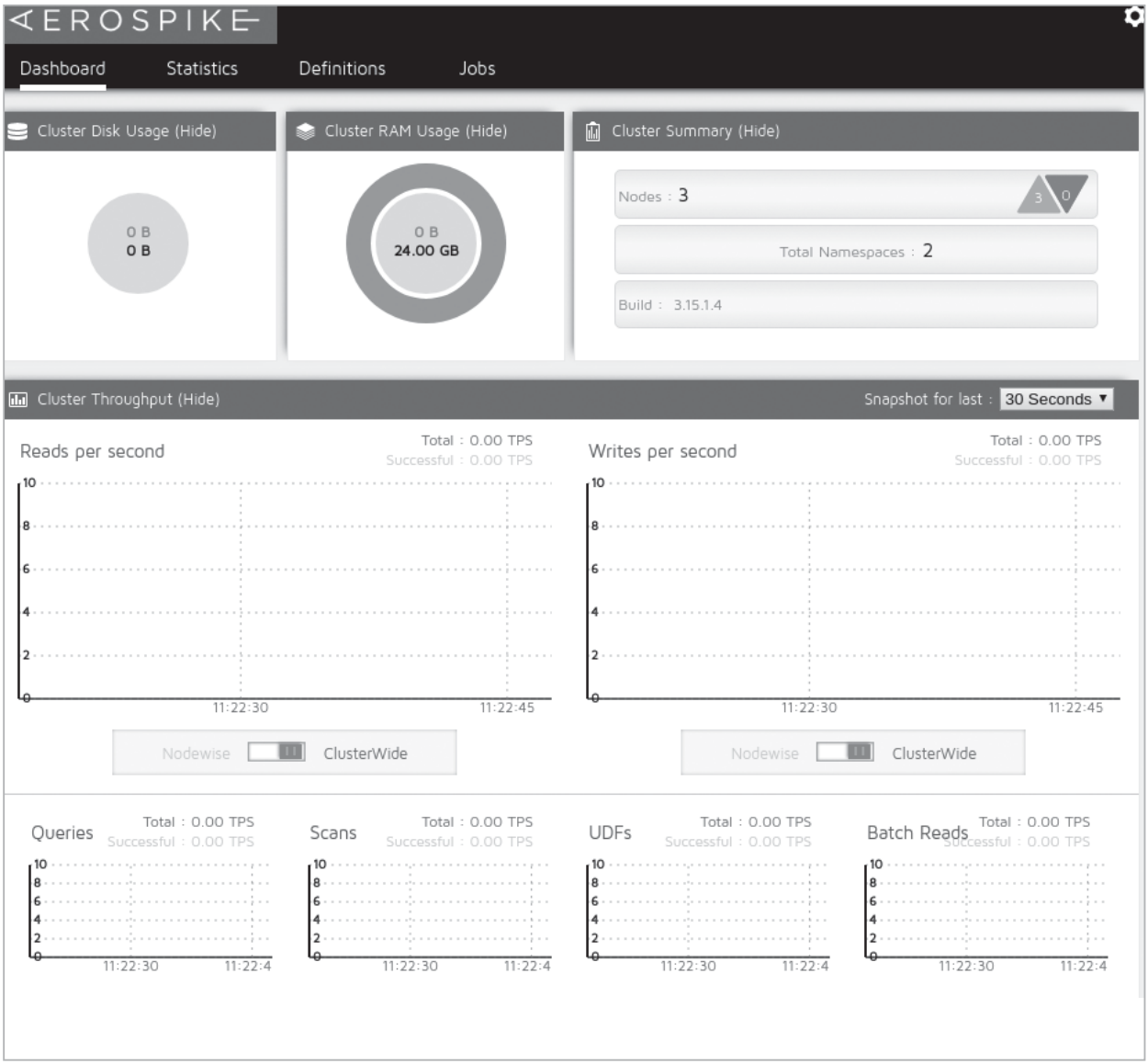


Figure 2: AMC dashboard

```
  - name: Uninstall RPM AMC
    command: yum remove aerospike-amc-* -y

  - name: Stop server
    command: service aerospike stop

  - name: Uninstall server
    shell: yum remove aerospike-* -y
```

The above playbook can be run using the following command:

```
$ ansible-playbook -i inventory/vbox/inventory playbooks/configuration/
aerospike.yml --tags uninstall
```

You can read the documentation at *https://www.aerospike.com/docs/* to learn more about Aerospike.

Chapter 15

Ansible Deployment of Elovation

The Elovation Ruby on Rails application was written by Christos Hrousis. It can be used to track the results of two-player games like chess, table tennis and foosball. This 14th article in this series tells us how to install the Elovation Ruby on Rails application.

The Elovation application uses the Elo rating system created by Arpad Elo, a Hungarian-born American physics professor. You can also use the Trueskill rating system for teams with multiple players and still provide rankings for individual players. Elovation requires at least Ruby on Rails 5.1 and uses the PostgreSQL database for its backend. It is free and open source software and has been released under the MIT licence.

GNU/Linux

An Ubuntu 16.04.1 LTS guest virtual machine (VM) instance using KVM/QEMU has been chosen to set up Elovation.

The host system is a Parabola GNU/Linux-libre x86_64 system, and Ansible is installed on the host system using the distribution package manager. The version of Ansible used is 2.4.3.0 as indicated below:

```
$ ansible --version
ansible 2.4.3.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/shakthi/.ansible/plugins/modules',
u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.14 (default, Jan  5 2018, 10:41:29) [GCC 7.2.1 20171224]
```

You should add an entry to the */etc/hosts* file for the guest 'Ubuntu' VM as follows:

```
192.168.122.244 ubuntu
```

On the host system, let's create a project directory structure to store the Ansible playbooks and inventory, as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
```

The Ubuntu 16.04.1 LTS server has Python 3 by default, and hence we can use the same with Ansible. The *inventory/kvm/inventory* file contains the following:

```
[elovation-host]
ubuntu ansible_host=192.168.122.244 ansible_connection=ssh ansible_user=ubuntu
ansible_password=ubuntu123

[elovation-host:vars]
ansible_python_interpreter=/usr/bin/python3
```

You should now be able to issue commands, using Ansible, to the guest OS. For example:

```
$ ansible -i inventory/kvm/inventory ubuntu -m ping

ubuntu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Dependencies

The apt-daily.service runs by default on a new installation of Ubuntu 16.04.1 LTS. If we need to install software, the APT lock held by this service needs to be removed. Hence, we need to first stop this service. The APT software package repository is then updated before installing the dependencies to set up Ruby.

```
---
- name: Install dependencies
  hosts: ubuntu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [apt]
```

```
  tasks:
    - name: Stop apt-daily.service
      shell: systemctl kill --kill-who=all apt-daily.service

    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install dependencies
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - libssl-dev
        - libreadline-dev
        - zlib1g-dev
        - build-essential
- git
```

The above playbook can be executed as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags apt -vv -K
```

The *-vv* represents the verbosity of the Ansible output. You can use up to four 'v's. The *-K* option prompts for the sudo password for the *ubuntu* user.

Rbenv

Let's use Rbenv to set up Ruby on this virtual instance. It allows the installation of multiple Ruby versions, and the version in production can be selected. The source repo of Rbenv is cloned first, and then *configure* and *make* are executed. The Rbenv PATH and initialisation are then updated in the *~/.bashrc* file.

```
- name: Build rbenv
  hosts: ubuntu
  tags: [rbenv]

  tasks:
    - name: Get rbenv
      git:
```

```
        repo: 'https://github.com/rbenv/rbenv.git'
        dest: "/home/{{ ansible_user }}/.rbenv"

    - name: Build rbenv
      shell: "cd /home/{{ ansible_user }}/.rbenv && src/configure && make -C
src"

    - name: Set rbenv PATH
      lineinfile:
        path: "/home/{{ ansible_user }}/.bashrc"
        state: present
        line: 'export PATH="$HOME/.rbenv/bin:$PATH"'

    - name: rbenv init
      lineinfile:
        path: "/home/{{ ansible_user }}/.bashrc"
        state: present
        line: 'eval "$(rbenv init -)"'
```

The playbook to set up Rbenv can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags rbenv -K
```

A sample execution output is shown below for reference:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags rbenv -K
SUDO password:

PLAY [Install dependencies] ****************************************************
*****************************************

TASK [Gathering Facts] *********************************************************
*****************************************
ok: [ubuntu]

PLAY [Build rbenv] *************************************************************
*****************************************

TASK [Gathering Facts] *********************************************************
*****************************************
ok: [ubuntu]
```

```
TASK [Get rbenv] ***************************************************************
*****************************************
ok: [ubuntu]

TASK [Build rbenv] *************************************************************
*****************************************
changed: [ubuntu]

TASK [Set rbenv PATH] **********************************************************
*****************************************
ok: [ubuntu]

TASK [rbenv init] **************************************************************
*****************************************
ok: [ubuntu]

PLAY [Build ruby] **************************************************************
*********************************************************************

TASK [Gathering Facts] *********************************************************
*****************************************
ok: [ubuntu]

PLAY [Postgresql] **************************************************************
*****************************************

TASK [Gathering Facts] *********************************************************
*****************************************
ok: [ubuntu]

PLAY [Elovation] ***************************************************************
*****************************************
TASK [Gathering Facts] *********************************************************
*****************************************
ok: [ubuntu]

PLAY RECAP *********************************************************************
*****************************************
ubuntu                     : ok=9    changed=1    unreachable=0    failed=0
```

Ruby

Ruby-build is a command line utility to install Ruby. Its repository needs to be cloned into the Rbenv plugins folder. Ruby 2.4.0 is then

installed using the *rbenv* command, and the same is set as the default global Ruby version in the following playbook:

```
- name: Build ruby
  hosts: ubuntu
  tags: [ruby]

  vars:
    rbenv_root: "/home/{{ ansible_user }}/.rbenv"

  tasks:
    - name: Create rbenv plugins
      shell: $SHELL -lc "mkdir -p {{ rbenv_root }}/plugins"

    - name: Get ruby-build
      git:
        repo: 'https://github.com/rbenv/ruby-build.git'
        dest: "{{ rbenv_root }}/plugins/ruby-build"

    - name: Install Ruby 2.4.0
      shell: $SHELL -lc "{{ rbenv_root }}/bin/rbenv install 2.4.0"

    - name: Set Ruby 2.4.0 as Global
      shell: $SHELL -lc "{{ rbenv_root }}/bin/rbenv global 2.4.0"
```

The above playbook can be invoked with the 'ruby' tags option as shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags ruby -vv -K
```

PostgreSQL

Elovation uses the PostgreSQL database, by default, as its backend data store. After updating the APT software package repository, we can install the PostgreSQL server and a few other dependencies. The PostgreSQL database server is started, and the Ansible playbook waits for the database to listen on Port 5432. A separate application user account is created in the database. Although we hard-coded the password in this example, in production, it is recommended that you use Vault to encrypt and decrypt the passwords with Ansible. The local authentication is changed from *peer* to *md5* in the PostgreSQL *pg_hba.conf* configuration file, and the database is restarted. The Ansible playbook again waits for the server to listen on Port 5432, as shown below:

```
- name: Postgresql
  hosts: ubuntu
  become: yes
  become_method: sudo
  tags: [postgresql]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install dependencies
      package:
        name: “{{ item }}”
        state: latest
      with_items:
        - python3-psycopg2
        - postgresql
        - postgresql-contrib
        - libpq-dev

    - name: Start database server
      systemd:
        name: postgresql
        state: started

    - wait_for:
        port: 5432

    - name: Create application user
      shell: sudo -u postgres createuser -s pguser

    - name: Set password for application user
      shell: sudo -u postgres psql -c “ALTER USER pguser WITH password
‘pguser123’”

    - name: Replace peer auth
      lineinfile:
        path: /etc/postgresql/9.5/main/pg_hba.conf
        regexp: ‘^local   all             all
peer’
        line: ‘local   all             all
```

```
md5'
```

```
    - name: Restart database server
      systemd:
        name: postgresql
        state: restarted
```

```
    - wait_for:
        port: 5432
```

The above playbook to install and configure the PostgreSQL database server can be run as shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags postgresql -vv -K
```

Elovation

The final step is to set up the Elovation Ruby on Rails application. The bundler software is used to track gems and the versions required by the Rails application. It is installed first and then the Elovation repository is cloned to the HOME folder. The *config/ database.yml* file is created and the database credentials are updated. The *bundle install* command is executed to fetch and install the required gems for the application. The database is created for the application and the migrations are executed to create the necessary tables. The entire playbook to set up Elovation is as follows:

```
- name: Elovation
  hosts: ubuntu
  tags: [elovation]
```

```
  vars:
    rbenv_root: "/home/{{ ansible_user }}/.rbenv"
```

```
  tasks:
    - name: Install bundler
      shell: "{{ rbenv_root }}/shims/gem install bundler"
```

```
    - name: Get elovation
      git:
        repo: 'https://github.com/elovation/elovation.git'
        dest: "/home/{{ ansible_user }}/elovation"
```

```
    - name: Create database.yml
```

```
    copy:
      src: "/home/{{ ansible_user }}/elovation/config/database.yml.example"
      dest: "/home/{{ ansible_user }}/elovation/config/database.yml"
      remote_src: yes
  - name: Add database credentials for development database
    lineinfile:
      path: "/home/{{ ansible_user }}/elovation/config/database.yml"
      insertafter: "  database: elovation_development"
      line: "  username: pguser\n  password: pguser123"
  - name: Add database credentials for test database
    lineinfile:
      path: "/home/{{ ansible_user }}/elovation/config/database.yml"
      insertafter: "  database: elovation_test"
      line: "  username: pguser\n  password: pguser123"

  - name: bundle install
    shell: "{{ rbenv_root }}/shims/bundle install"
    args:
      chdir: "/home/{{ ansible_user }}/elovation"

  - name: Create database
    shell: "RAILS_ENV='development' {{ rbenv_root }}/shims/bundle exec rake
db:create"
    args:
      chdir: "/home/{{ ansible_user }}/elovation"
  - name: Migrate database
    shell: "RAILS_ENV='development' {{ rbenv_root }}/shims/bundle exec rake
db:migrate"
    args:
      chdir: "/home/{{ ansible_user }}/elovation"
```

The above playbook can be executed as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/elovation.
yml --tags elovation -vv -K
```

You can then log in to the VM, and manually start the RAILS application using the following command:

```
$ cd elovation
$ RAILS_ENV="development" bundle exec rails server --binding=192.168.122.244
```

The application will listen on 192.168.122.244:3000, which you can open in a browser on the host system. You will see the home page, as

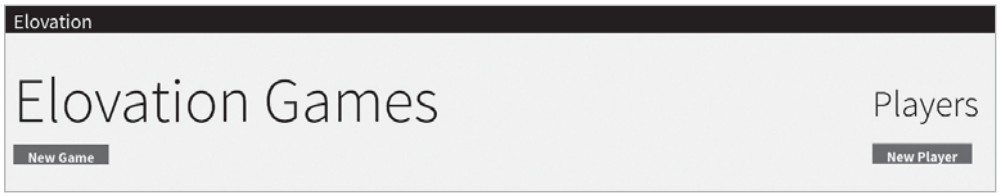


Figure 1: Home page

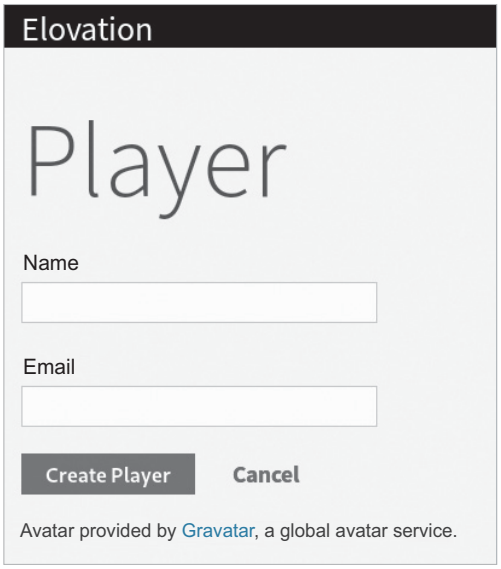


Figure 2: Adding a player

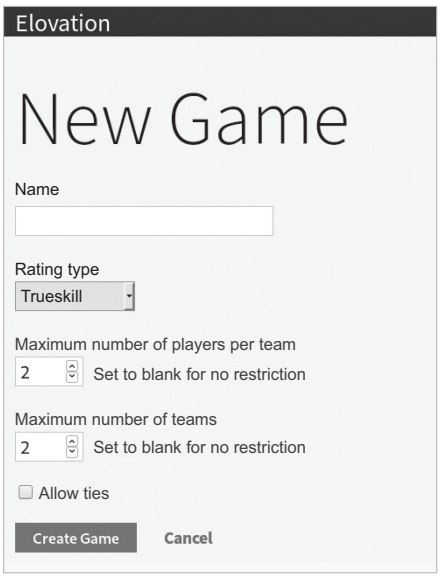


Figure 3: Creating a new game

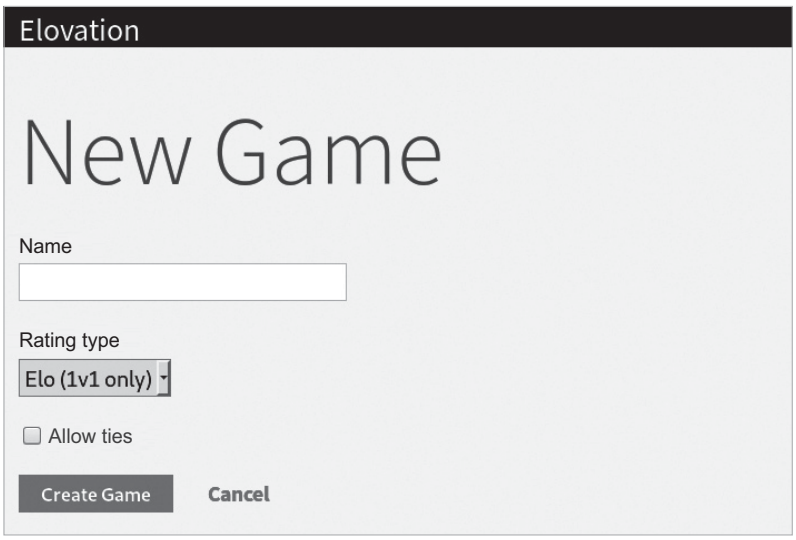


Figure 4: Creating a new game with Elo rating

shown in Figure 1.

You can add a player by providing the name and e-mail address, as shown in Figure 2.

You can start a new game using the Trueskill rating system, as shown in Figure 3.

You can also create a new game with the Elo rating system, as shown in Figure 4.

Chapter 16

Ansible Deployment of Nginx to Serve Static Files

This is the 15th article in the DevOps series, in which the author demonstrates how an Nginx server can be set up to serve static file content (HTML, CSS, JavaScript), and to use Goaccess for log analysis.

Nginx is a free and open source Web server written in C by Igor Sysoev. It is designed to handle thousands of client connections. It is also popularly used as a load balancer. It does not require much memory and can also be used as an HTTP cache server or a mail proxy server. It was released in 2004 and uses a BSD-like licence.

Goaccess is also free and open source software. It is a real-time Web log analyser written in C by Gerardo Orellana. You can run it remotely on a *nix terminal or access it through a browser. It requires only *ncurses* as a dependency when used from a terminal. It supports a number of Web log formats such as Apache, Nginx, Elastic load balancing, etc. You can also use a terminal dashboard with it and export reports to HTML. It was first released in 2010 and uses the MIT licence.

GNU/Linux

A Debian 9 (x86_64) guest virtual machine (VM) instance using KVM/ QEMU is chosen to install Nginx.

The host system is a Parabola GNU/ Linux-libre x86_64 system and Ansible is installed on it using the distribution package manager. The version of Ansible used is 2.5.0 as indicated below:

```
$ ansible --version
ansible 2.5.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/guest/.ansible/plugins/modules', u'/
usr/share/ansible/plugins/modules']
```

```
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.14 (default, Jan  5 2018, 10:41:29) [GCC 7.2.1 20171224]
```

You should add an entry to the */etc/hosts* file for the guest 'debian' VM as follows:

```
192.168.122.140 debian
```

On the host system, we will create a project directory structure to store the Ansible playbooks, the inventory and static Web files:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
debian ansible_host=192.168.122.140 ansible_connection=ssh ansible_user=debian
```

The *ansible/files/* directory has the following:

```
ansible/files/_site
             /example.domain
             /goaccessrc
```

The default Debian 9 installation does not install the sudo package. Start the new Debian VM, log in as the root user, and install the sudo package. You should also provide sudo access to the user account, which is 'debian' (in this example).

```
root@debian:~# apt-get install sudo
```

```
root@debian:~# adduser debian sudo
Adding user `debian' to group `sudo'...
Adding user debian to group sudo
Done.
```

You should now be able to issue commands from the host system to the guest OS, using Ansible. For example:

```
$ ansible -i inventory/kvm/inventory debian -m ping
```

```
debian | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

Nginx

The first step is to install the Nginx Web server. The software package repository is updated, and then the Nginx package is installed. The Nginx Web server is started and we wait for the server to listen on port 80. The playbook to install Nginx is as follows:

```
---
- name: Install nginx
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [nginx]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install nginx
      package:
        name: “{{ item }}”
        state: latest
      with_items:
        - nginx

    - name: Start nginx
      service:
        name: nginx
        state: started

    - wait_for:
        port: 80
```

The above playbook can be invoked using the command shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx.yml
--tags nginx -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four *v*'s for a more detailed output. The *-K* option prompts for the sudo password for the *debian* user account. You can now open a Web browser on the local host with the URL *http://192.168.122.140* and you should see the default Nginx home page as shown in Figure 1.

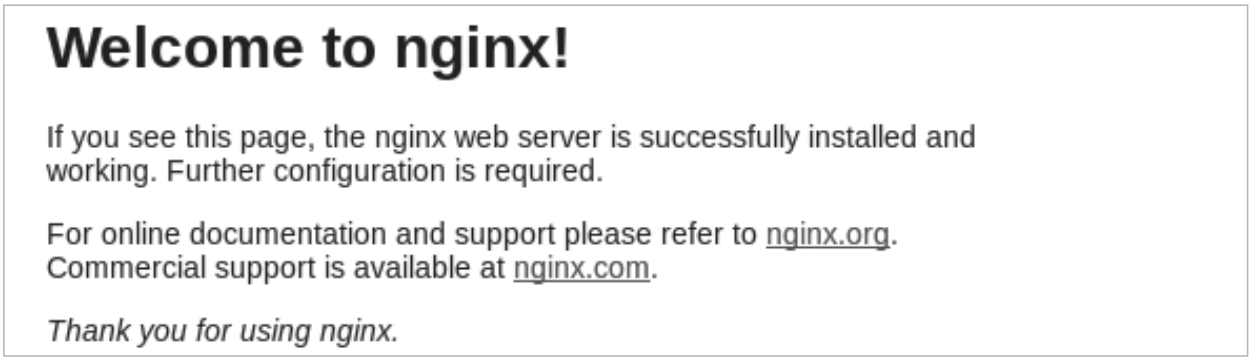


Figure 1: Nginx default home page

Serving static files

You can use any static site generator to create your HTML, CSS and JavaScript files. You will need to copy the generated files to the *files/_site* directory. The Ansible playbook to copy the static files to the Nginx Web server location is given below:

```
- name: Copy static files
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [static]

  tasks:
    - name: Stop nginx server
      service:
        name: nginx
        state: stopped

    - name: Remove existing directory
      shell: /bin/rm -rf /var/www/html/home/*

    - name: Home directory should exist
      file:
        path: /var/www/html/home
        state: directory
```

```
      mode: 0755

  - name: Copy latest files
    copy:
      src: ../../files/_site/
      dest: /var/www/html/home/
      directory_mode: yes

  - name: Copy example.domain nginx file
    copy:
      src: ../../files/example.domain
      dest: /etc/nginx/sites-available/

  - name: Create link file
    file:
      src: /etc/nginx/sites-available/example.domain
      dest: /etc/nginx/sites-enabled/example.domain
      owner: root
      group: root
      force: yes
      state: link

  - name: Start nginx
    service:
      name: nginx
      state: started

  - wait_for:
      port: 80
```

The Nginx Web server is first stopped, as we are changing the Web content. If you have multiple instances of the Web server running, you can run the playbook in a rolling upgrade fashion. In this way, if client requests occur during the upgrade, you will not have a downtime. In this playbook, we first remove the contents of the old directory, and then copy the new contents. Depending on the volume of content, this may or may not suit your requirements. You can also use the *synchronise* module in Ansible (*http://docs.ansible.com/ansible/latest/modules/synchronize_module.html*). This uses *rsync* to synchronise files between your local host and the remote server.

The *example.domain* is the Nginx configuration file for the website. As per the Nginx configuration, it is copied to the */etc/nginx/sites-available*

directory, and a symbolic link file to it is created in the */etc/nginx/sites-enabled* directory. The content of the *example.domain* is as follows:

```
server {
       listen 80;
       listen [::]:80;
       server_name example.domain www.example.domain;

       root /var/www/html;

       # Add index.php to the list if you are using PHP
       index index.html index.htm index.nginx-debian.html;

       location / {
              return 301 $scheme://www.example.domain/home$request_uri;
       }

       location /home {
              try_files $uri $uri/ =404;
       }
}
```

The Nginx Web server is then started to serve the newly copied content. The Ansible playbook for copying and serving the static files can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx.yml
--tags static -vv -K
```

You can now refresh the *http://192.168.122.140* page on your local host to see your static Web content.

Goaccess

The Goaccess interactive, real-time Web log analyser is available in the Debian package repository. After updating the software APT repository, Goaccess is installed. The configuration format to parse the Nginx logs is made available in the *goaccessrc* file. Its contents are as follows:

```
log-format %h %^[%d:%t %^]  "%r" %s %b "%R" "%u" %T %^
time-format %H:%M:%S
date-format %d/%b/%Y
```

The above code is copied to *~/.goaccessrc*. The Ansible playbook to install Goaccess is given below:

```
- name: Install goaccess
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [goaccess]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install goaccess
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - goaccess

    - name: Copy goaccessrc
      copy:
        src: ../../files/goaccessrc
        dest: "/home/{{ ansible_ssh_user }}/.goaccessrc"
```

The above playbook's invocation and sample output is shown below for reference:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/nginx.yml
--tags goaccess -vv -K
ansible-playbook 2.5.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/guest/.ansible/plugins/modules', u'/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.14 (default, Jan  5 2018, 10:41:29) [GCC 7.2.1 20171224]
Using /etc/ansible/ansible.cfg as config file
SUDO password:
PLAYBOOK: nginx.yml ************************************************************
```

```
*****************************************
3 plays in playbooks/configuration/nginx.yml

PLAY [Install nginx] ***********************************************************
*****************************************
TASK [Gathering Facts] *********************************************************
*****************************************
task path: /home/guest/ansible/playbooks/configuration/nginx.yml:2
ok: [debian]
META: ran handlers
META: ran handlers
META: ran handlers

PLAY [Copy static files] *******************************************************
*****************************************
TASK [Gathering Facts] *********************************************************
*****************************************task path: /home/guest/ansible/
playbooks/configuration/nginx.yml:29
ok: [debian]
META: ran handlers
META: ran handlers
META: ran handlers

PLAY [Install goaccess] ********************************************************
*****************************************

TASK [Gathering Facts] *********************************************************
*****************************************
task path: /home/guest/ansible/playbooks/configuration/nginx.yml:79
ok: [debian]
META: ran handlers

TASK [Update the software package repository] **********************************
*****************************************
task path: /home/guest/ansible/playbooks/configuration/nginx.yml:87
changed: [debian] => {"cache_update_time": 1523274608, "cache_updated": true,
"changed": true}

TASK [Install goaccess] ********************************************************
*****************************************
task path: /home/guest/ansible/playbooks/configuration/nginx.yml:91
ok: [debian] => (item=goaccess) => {"cache_update_time": 1523274608, "cache_
```

```
updated": false, "changed": false, "item": "goaccess"}

TASK [Copy goaccessrc] ********************************************************
*****************************************
task path: /home/guest/ansible/playbooks/configuration/nginx.yml:98
changed: [debian] => {"changed": true, "checksum":
"b8981bb97a7727d8fbd7d92cd1730b9cac19a2b0", "dest": "/home/debian/.goaccessrc",
"gid": 0, "group": "root", "md5sum": "f85cd220742cc6735e74327388744f3d",
"mode": "0644", "owner": "root", "size": 98, "src": "/home/debian/.ansible/tmp/
ansible-tmp-1523274615.53-165679868036633/source", "state": "file", "uid": 0}
META: ran handlers
META: ran handlers

PLAY RECAP ********************************************************************
*****************************************
debian                     : ok=6    changed=2    unreachable=0    failed=0
```

You can now log in to the guest VM and run Goaccess from the terminal for the Nginx *access.log* file as follows:

```
$ sudo goaccess -f /var/log/nginx/access.log -p ~/.goaccessrc -a
```

You will see the Goaccess dashboard as illustrated in Figure 2.

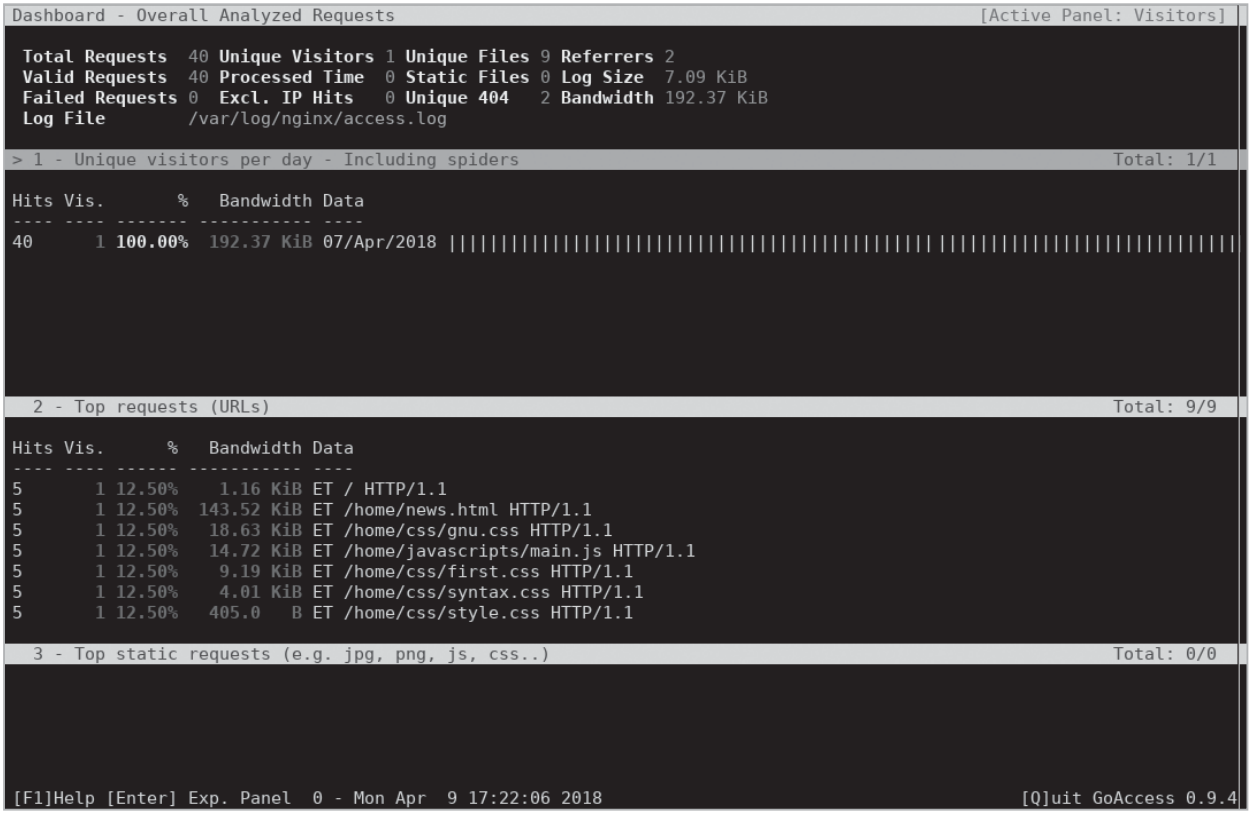


Figure 2: Goaccess dashboard

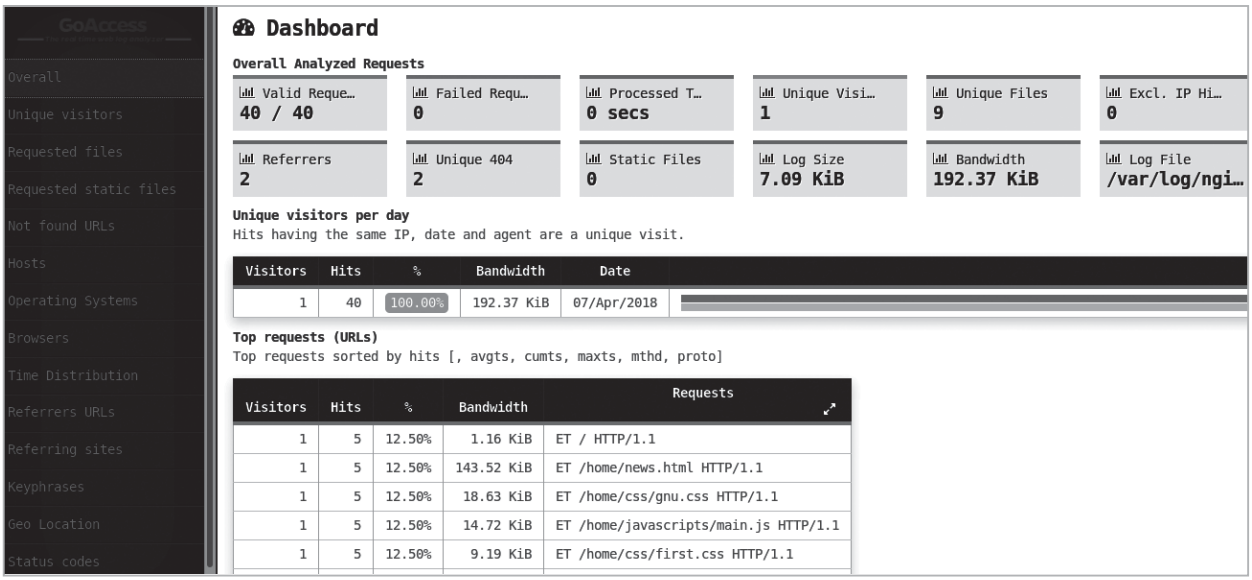


Figure 3: Goaccess HTML report

You can also export the dashboard to an HTML report that you can view in a browser:

```
$ sudo goaccess -f /var/log/nginx/access.log -p ~/.goaccessrc -a > report.html
```

A sample screenshot of the HTML report is shown in Figure 3.

You are encouraged to read the Goaccess manual page at *https://goaccess.io/man* to learn more about its usage.

Chapter 17

Using Ansible with the Security Technical Implementation Guide (STIG)

STIG is an acronym for Security Technical Implementation Guide, which is a cyber security protocol that sets the standards for the security of networks, computers, servers, etc. In this 16th article in the DevOps series, we will learn how to build Ansible playbooks to test and set up CentOS 6 as per STIG on RHEL6, version 1, release 19.

The Security Technical Implementation Guide (STIG) has been developed jointly by Red Hat, the National Security Agency (NSA) and the Defence Information Systems Agency (DISA) for the US Department of Defense (DoD). The security vulnerabilities are classified into three Category Codes (CAT for short), based on the severity.

CAT I type is an exploit that "…directly and immediately results in loss of confidentiality, availability or integrity."

CAT II type vulnerablity "…has a potential to result in the loss of confidentiality, availability or integrity."

The existence of a **CAT III type** vulnerability "…degrades measures to protect against loss of confidentiality, availability or integrity."

On October 16, 2009, the chief information officer of the Department of Defense (USA) released a memorandum with guidance on using free and open source software (FOSS). The memo can be obtained from *http://dodcio.defense.gov/Portals/0/Documents/FOSS/2009OSS.pdf.*

Setting things up

A CentOS 6.8 virtual machine (VM) running on KVM is used for the setup. Please ensure that the VM has access to the Internet. The Ansible version used on the host (Parabola GNU/Linux-libre x86_64) is 2.5.0.

```
$ ansible --version
ansible 2.5.0
  config file = /etc/ansible/ansible.cfg
```

```
  configured module search path = [u'/home/guest/.ansible/plugins/modules', u'/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 2.7.14 (default, Jan  5 2018, 10:41:29) [GCC 7.2.1 20171224]
```

The ansible/ folder contains the following files:

```
ansible/inventory/kvm/inventory
ansible/playbooks/configuration/stig.yml
ansible/playbooks/configuration/fix-stig.yml
```

The IP address of the guest CentOS 6.8 VM is added to the inventory file as shown below:

```
centos ansible_host=192.168.122.16 ansible_connection=ssh ansible_user=root
ansible_password=password
```

Also, add an entry for the *centos* guest in */etc/hosts* file as indicated below:

```
192.168.122.16 centos
```

The *libselinux-python* package needs to be installed on the CentOS guest VM as follows, in order to verify SELinux configuration using Ansible:

```
# yum update && yum install libselinux-python
```

We shall now go through a few CAT vulnerabilities and look at how to use Ansible to protect against them.

RHEL-06-000005

The rule version (STIG-ID) RHEL-06-000005 states that the audit system must send an email to the designated staff members when the storage volume reaches its capacity. The 'space_left_action' variable in */etc/audit/auditd.conf* should be set to an email address of an administrator. The following Ansible code snippet checks the same, and asserts if an '@' symbol for the email address exists.

```
---
- name: STIG
  hosts: centos
```

```
  become: true
  gather_facts: true
  tags: [STIG]

  tasks:
    - name: "RHEL-06-000005 | CAT II | The audit system must alert designated
staff members"
      shell: "grep ^space_left_action /etc/audit/auditd.conf"
      register: space_left_action_result

    - assert:
        that:
          - "'@' in space_left_action_result.stdout"
```

RHEL-06-000008

This rule states that the Red Hat GPG keys need to be installed in order to cryptographically verify that the packages are actually from Red Hat. The 'gpg-pubkey' software package should be installed in the system.

```
- name: "RHEL-06-000008 | CAT I | Vendor-provided cryptographic certificates
must be installed"

shell: "rpm -q gpg-pubkey"
  register: gpg_pubkey_result

- assert:
    that:
      - "gpg_pubkey_result.stdout != ''"
```

RHEL-06-000011

The system should always be up to date with the latest software. This can be checked from the output of 'yum check-update'. The remedy is to run 'yum update' to pull the latest software packages from the repositories, as follows:

```
- name: "RHEL-06-000011 | CAT II | System security patches and updates must be
installed"
  shell: "yum check-update"
  register: yum_check_update_result

- assert:
    that:
      - "yum_check_update_result.rc == 0"
```

RHEL-06-000013

This rule states that the downloaded packages need to be cryptographically verified before proceeding with the installation. The 'gpgcheck=1' verification should exist in the */etc/yum.conf* file.

```
- name: "RHEL-06-000013 | CAT II | System package management tool must be
cryptographically verified"
  shell: "grep gpgcheck /etc/yum.conf"
  register: gpgcheck_result

- assert:
    that:
      - "'=1' in gpgcheck_result.stdout"
```

RHEL-06-000016

The system should have a file integrity tool installed. Advanced Intrusion Detection Environment (AIDE) is a file and directory integrity checker, released under the GPL. It can be installed on CentOS, as follows:

```
- name: "RHEL-06-000016 | CAT II | A file integrity tool must be installed"
  shell: "rpm -q aide"
  register: aide_result

- assert:
    that:
      - "aide_result.stdout != ''"
```

RHEL-06-000018

The AIDE file integrity tool must be initialised before using it. An initial database needs to be created for its use. The database consists of regular expression rules obtained from the configuration files. A number of digest algorithms such as md5, sha1, sha256, sha512, crc32, etc, are used to check the integrity of files.

```
- name: "RHEL-06-000018 | CAT II | A file integrity baseline must be created"
  stat:
    path: /var/lib/aide/aide.db.gz
  register: aide_db_result

- assert:
    that:
      - "aide_db_result.stat.exists == true"
```

RHEL-06-000019

The *.rhosts* and */etc/hosts.equiv* files used by the *rsh* daemon can allow unauthorised access to the system. Hence, these should not exist in the file system. The Ansible Stat module is used to check the presence of these files.

```
- name: "RHEL-06-000019 | CAT I | There must be no /etc/hosts.equiv files"
  stat:
    path: /etc/hosts.equiv
  register: etc_hosts_result

- assert:
    that:
      - "etc_hosts_result.stat.exists == false"

- name: "RHEL-06-000019 | CAT I | There must be no .rhosts file"
  stat:
    path: ~/.rhosts
  register: rhosts_result

- assert:
    that:
      - "rhosts_result.stat.exists == false"
```

RHEL-06-000020

The SELinux policy needs to be set to 'Enforcing' in the */etc/selinux/config* file. This setting needs to take effect from the boot time, and it enforces limits on system services.

```
- name: "RHEL-06-000020 | CAT II | The system must use a Linux Security Module
to enforce limits"
  shell: "grep ^SELINUX= /etc/selinux/config"
  register: selinux_result

- assert:
    that:
      - "'enforcing' in selinux_result.stdout"
```

RHEL-06-000023

The SELinux targeted policy ensures that processes that are targeted run in a confined domain, and those that are not, run in an unconfined domain. The SELINUXTYPE variable needs to be set to 'targeted' in /

etc/selinux/config for this setting to take effect.

```
- name: "RHEL-06-000023 | CAT III | The system must use a Linux Security Module
to limit the privileges"

  shell: "grep ^SELINUXTYPE= /etc/selinux/config"
  register: selinux_type_result

- assert:
    that:
      - "'targeted' in selinux_type_result.stdout"
```

RHEL-06-000032

We need to ensure that there is only one root user in the system. The */etc/passwd* file can be checked to see that there is only one entry with UID 0.

```
- name: "RHEL-06-000032 | CAT II | The root account must be the only account
having UID 0"

  shell: "awk -F: '($3 == 0) {print}' /etc/passwd"
  register: root_account_result

- assert:
    that:
      - "root_account_result.stdout_lines | length == 1"
```

The entire playbook invocation and run to check the above STIG-IDs is shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/stig.yml

PLAY [STIG]*********************************************

TASK [Gathering Facts] *********************************
ok: [centos]

TASK [RHEL-06-000005 | CAT II | The audit system must alert designated staff
members] **********************************

changed: [centos]
```

```
TASK [assert] *********************************************ok: [centos] => {
    “changed”: false,
    “msg”: “All assertions passed”
}

TASK [RHEL-06-000008 | CAT I | Vendor-provided cryptographic certificates must
be installed]*****************************

changed: [centos]

TASK [assert] *********************************************ok: [centos] => {
    “changed”: false,
    “msg”: “All assertions passed”
}

TASK [RHEL-06-000011 | CAT II | System security patches and updates must be
installed] *********************************

changed: [centos]

TASK [assert] *********************************************ok: [centos] => {
    “changed”: false,
    “msg”: “All assertions passed”
}

TASK [RHEL-06-000013 | CAT II | System package management tool must be
cryptographically verified] *********************
changed: [centos]

TASK [assert] ***********************************************
ok: [centos] => {
    “changed”: false,
    “msg”: “All assertions passed”
}

TASK [RHEL-06-000016 | CAT II | A file integrity tool must be installed] *******
*****************************************

changed: [centos]

TASK [assert] ***********************************************
ok: [centos] => {
    “changed”: false,
```

```
    "msg": "All assertions passed"
}

TASK [RHEL-06-000018 | CAT II | A file integrity baseline must be created] *****
*****************************************ok: [centos]

TASK [assert]*************************************************ok: [centos] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [RHEL-06-000019 | CAT I | There must be no /etc/hosts.equiv files] ********
*************************************ok: [centos]

TASK [assert] ********************************************
ok: [centos] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [RHEL-06-000019 | CAT I | There must be no .rhosts file] ******************
****************************************
ok: [centos]

TASK [assert] ************************************************ok: [centos] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [RHEL-06-000020 | CAT II | The system must use a Linux Security Module to
enforce limits] ************************
changed: [centos]

TASK [assert] ********************************************
ok: [centos] => {
    "changed": false,
    "msg": "All assertions passed"
}

TASK [RHEL-06-000023 | CAT III | The system must use a Linux Security Module to
limit the privileges] ********************
changed: [centos]

TASK [assert] ********************************************ok: [centos] => {
```

```
    “changed”: false,
    “msg”: “All assertions passed”
}

TASK [RHEL-06-000032 | CAT II | The root account must be the only account
having UID 0] *********************************
changed: [centos]

TASK [assert] *************************************************
ok: [centos] => {
    “changed”: false,
    “msg”: “All assertions passed”
}

PLAY RECAP ****************************************************
centos                     : ok=23   changed=8    unreachable=0    failed=0
```

Fixing STIG

We will also need to create a *fix-stig.yml* Ansible playbook, to set up the system to meet the required STIG specification. The following playbook provides the necessary steps to set up the system for the above mentioned STIG-IDs.

```
# ansible-playbook -i inventory/kvm/inventory playbooks/configuration/fix-stig.
yml -vv
---
- name: STIG
  hosts: centos
  become: true
  gather_facts: true
  tags: [FIX]

  tasks:
    - name: “RHEL-06-000011 | CAT II | System security patches and updates must
be installed”
      # Also RHEL-06-000008 | CAT I | Vendor-provided cryptographic certificates
must be installed
      yum:
        name: ‘*’
        state: latest
    - name: “RHEL-06-000005 | CAT II | The audit system must alert designated
```

```
staff members"
      lineinfile:
        dest: "/etc/audit/auditd.conf"
        regexp: '^space_left_action'
        line: 'space_left_action = author@shakthimaan.com'

    - name: "RHEL-06-000016 | CAT II | A file integrity tool must be installed"
      yum:
        name: 'aide'
        state: latest

    - name: "RHEL-06-000018 | CAT II | A file integrity baseline must be
created"

      shell: "aide --init && cp /var/lib/aide/aide.db.new.gz /var/lib/aide/
aide.db.gz"

    - name: "RHEL-06-000019 | CAT I | There must be no /etc/hosts.equiv or
.rhosts file"
      file:
        dest: '{{ item }}'
        state: absent
      with_items:
        - { "/etc/hosts.equiv" }
        - { "~/.rhosts" }

    - name: "RHEL-06-000020 | CAT II | The system must use a Linux Security
Module to enforce limits"
      lineinfile:
        dest: "/etc/selinux/config"
        regexp: '^SELINUX='
        line: 'SELINUX=enforcing'

    - name: "RHEL-06-000023 | CAT III | The system must use a Linux Security
Module to limit the privileges"
      lineinfile:
        dest: "/etc/selinux/config"
        regexp: '^SELINUXTYPE='
        line: 'SELINUXTYPE=targeted'
```

You can obtain the entire STIG specification and documents from *https://iase.disa.mil/stigs/os/unix-linux/Pages/index.aspx.*

Chapter 18

Ansible Deployment of Consul

Consul supports multiple data centres, can be used as a key/value store, and can monitor cluster health. In this 17th article in the DevOps series, we discuss the Ansible deployment of Consul.

Consul is a tool that has been written by HashiCorp, and can be used for creating health checks for services and systems. It provides a simple HTTP API for using it as a key/value store. Consul is distributed, highly available, and is data centre aware. The recommended number of Consul nodes is three or five in a cluster, in order to handle failures. Every data centre can contain a Consul cluster. Consul is released under the Mozilla Public License 2.0.

Setting up Consul

We are going to set up and configure a Consul cluster with three Debian 9 (x86_64) guest virtual machines (VMs) using KVM/QEMU.

The host system is a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.5.3 as indicated below:

```
$ ansible --version
ansible 2.5.3
  config file = /etc/ansible/ansible.cfg
  configured module search path = [‘/home/guest/.ansible/plugins/modules’, ‘/usr/
share/ansible/plugins/modules’]
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

On the host system, we will create a project directory structure to store the Ansible playbooks, inventory and configuration files, as shown below:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
[consul1]
host1 ansible_host=192.168.122.140 ansible_connection=ssh ansible_user=debian
ansible_password=password

[consul2]
host2 ansible_host=192.168.122.208 ansible_connection=ssh ansible_user=debian
ansible_password=password

[consul3]
host3 ansible_host=192.168.122.59 ansible_connection=ssh ansible_user=debian
ansible_password=password

[bootstrap:children]
consul1

[server:children]
consul2
consul3
[all:children]
bootstrap
server
```

We have three Consul guest VMs running that are labelled as 'consul1', 'consul2' and 'consul3'. consul1 is defined as the bootstrap node. The other members (consul2 and consul3) of the Consul cluster belong to the server group. The 'all' group consists of all the three nodes.

The default Debian 9 installation does not have the *sudo* package installed. Log in as the root user, and install the *sudo* package on all the three VMs. The 'debian' user also requires *sudo* access:

```
root@debian:~# apt-get install sudo

root@debian:~# adduser debian sudo
Adding user `debian' to group `sudo'...
Adding user debian to group sudo
Done.
```

You can now test connectivity from Ansible to the individual Consul nodes as well as collectively, by using the following commands:

```
$ ansible -i inventory/kvm/inventory consul1 -m ping
consul1 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}

$ ansible -i inventory/kvm/inventory consul2 -m ping
consul2 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}

$ ansible -i inventory/kvm/inventory consul3 -m ping
consul3 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}

$ ansible -i inventory/kvm/inventory all -m ping
consul2 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
consul3 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
consul1 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

Consul

The first step is to install Consul on all the nodes. The software package repository is updated, and a few network tools are installed. The ‘consul’ binary is extracted from a Zip file obtained at HashiCorp’s website and copied to */usr/local/bin/consul*. The execution of the binary is verified. A ‘consul’ user account is created in the system, and the */var/consul* directory is created and owned by the ‘consul’

user. The playbook to install Consul is provided below for reference:

```
---
- name: Install Consul
  hosts: all
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [consul]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install dependencies
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - curl
        - net-tools
        - unzip

    - name: Install consul
      unarchive:
        src: https://releases.hashicorp.com/consul/1.1.0/consul_1.1.0_linux_
amd64.zip
        dest: /usr/local/bin
        remote_src: yes
    - name: Verify consul installation
      shell: "consul --version"
      register: consul_version

    - assert:
        that:
          - "'Consul' in consul_version.stdout"

    - name: Create consul user
      user:
        name: consul
```

```
    - name: Create Consul /var directory
      file:
        path: /var/consul
        state: directory
        owner: consul
        group: consul
        mode: 0755
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/consul.yml
--tags consul -vv -K
```

The '-vv' represents the verbosity in the Ansible output. You can use up to four 'v's for a more detailed output. The '-K' option prompts for the sudo password for the *debian* user account.

You can also now log in to any one of the nodes and check the version output by running Consul as indicated below:

```
$ consul --version
Consul v1.1.0
```

```
Protocol 2 spoken by default, understands 2 to 3 (agent will automatically use
protocol >2 when speaking to compatible agents)
```

Bootstrap node

The bootstrap Consul node is the first to be configured. The */etc/consul.d/bootstrap* directory is created and its configuration file (*bootstrap-config.json*) is copied to it. Its file contents are as follows:

```
{
       "bootstrap": true,
       "client_addr": "0.0.0.0",
       "server": true,
       "datacenter": "chennai",
       "data_dir": "/var/consul",
       "encrypt": "PfCkrq/lFPem7slKP5N2Cw==",
       "log_level": "INFO",
       "enable_syslog": true
}
```

The configuration specifies that this is a bootstrap node, and the server

should bind to any IP address. We specify a name to the data centre, and also the path to the data directory. An encryption key is specified to encrypt the traffic between the Consul nodes. Finally, the log level and use of syslog is specified in the configuration file.

We also create a *systemd* configuration file for starting the bootstrap Consul node as shown below:

```
[Unit]
Description=Consul service discovery agent
Requires=network-online.target
After=network-online.target

[Service]
Environment=GOMAXPROCS=2
Restart=on-failure
ExecStart=/usr/local/bin/consul agent -server -ui -data-dir=/tmp/consul
-node=host1 -config-dir=/etc/consul.d/bootstrap -enable-script-checks=true
ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGTERM

[Install]
WantedBy=multi-user.target
```

The '-ui' option is provided to enable the Web front-end for Consul. The bootstrap node's name is 'host1'. The entire playbook to set up and start the bootstrap node is as follows:

```
- name: Configure bootstrap Consul node
  hosts: bootstrap
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [bootstrap]

  tasks:
    - name: Create Consul bootstrap directory
      file:
        path: /etc/consul.d/bootstrap
        state: directory
        mode: 0755

    - name: Copy configuration file
```

```
      copy:
        src: ../../files/bootstrap-config.json
        dest: /etc/consul.d/bootstrap/config.json

    - name: Copy systemd bootstrap consul.service
      copy:
        src: ../../files/bootstrap-consul.service
        dest: /etc/systemd/system/consul.service

    - name: Start bootstrap Consul
      systemd:
        name: consul.service
        state: started
```

A sample execution of the above playbook is given below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/consul.yml
--tags bootstrap -K
SUDO password:

PLAY [Install Consul] *************************************

TASK [Gathering Facts] ************************************************************
************************ok: [host1]
ok: [host2]
ok: [host3]

PLAY [Configure bootstrap Consul node] ***********************

TASK [Gathering Facts] ************************************
ok: [host1]

TASK [Create Consul bootstrap directory] ********************
changed: [host1]

TASK [Copy configuration file] ******************************
changed: [host1]

TASK [Copy systemd bootstrap consul.service] ****************
changed: [host1]

TASK [Start bootstrap Consul] *******************************
```

```
changed: [host1]

PLAY [Configure other Consul nodes] ***************************

TASK [Gathering Facts] ************************************
ok: [host2]
ok: [host3]

PLAY RECAP ************************************************
host1    : ok=6    changed=4    unreachable=0    failed=0
host2    : ok=2    changed=0    unreachable=0    failed=0
host3    : ok=2    changed=0    unreachable=0    failed=0
```

Other Consul nodes

The last step is to configure the other Consul nodes in the cluster. A */etc/consul.d/server* directory is created and the node configuration file is copied to it. The contents of the configuration file are as follows:

```
{
       "bootstrap": false,
       "server": true,
       "datacenter": "chennai",
       "data_dir": "/var/consul",
       "encrypt": "PfCkrq/lFPem7slKP5N2Cw==",
       "log_level": "INFO",
       "enable_syslog": true,
       "start_join": [ {% for item in groups['all'] %} "{{ hostvars[item].
ansible_default_ipv4.address }}" {% if not loop.last %} , {% endif %} {% endfor
%} ]
}
```

The configuration specifies that this is not a bootstrap node but is part of the Consul cluster. The data centre, data directory and encryption keys are specified. The log level information is also supplied. Finally, the IP addresses of all the Consul nodes are mentioned to join the cluster.

A *systemd* configuration file is also created to start the Consul service on these nodes, as shown below:

```
[Unit]
Description=Consul service discovery agent
Requires=network-online.target
After=network-online.target
```

```
[Service]
Environment=GOMAXPROCS=2
Restart=on-failure
ExecStart=/usr/local/bin/consul agent -server -data-dir=/tmp/consul -node={{
inventory_hostname }} -config-dir=/etc/consul.d/server -enable-script-
checks=true
ExecReload=/bin/kill -HUP $MAINPID
KillSignal=SIGTERM

[Install]
WantedBy=multi-user.target
```

The *inventory_hostname* is used as the node name when starting the Consul service. The playbook for setting up the Consul nodes is given below:

```
- name: Configure other Consul nodes
  hosts: server
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [nodes]

  tasks:
    - name: Create Consul server directory
      file:
        path: /etc/consul.d/server
        state: directory
        mode: 0755

    - name: Copy configuration file
      template:
        src: "../../files/node-config.json.j2"
        dest: /etc/consul.d/server/config.json

    - name: Copy systemd node consul.service
      template:
        src: "../../files/node-consul.service.j2"
        dest: /etc/systemd/system/consul.service

    - name: Start Consul service
      systemd:
```

```
        name: consul.service
        state: started
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/consul.yml
--tags nodes -vv -K
```

Verification

You can now verify the nodes that are part of the Consul cluster using

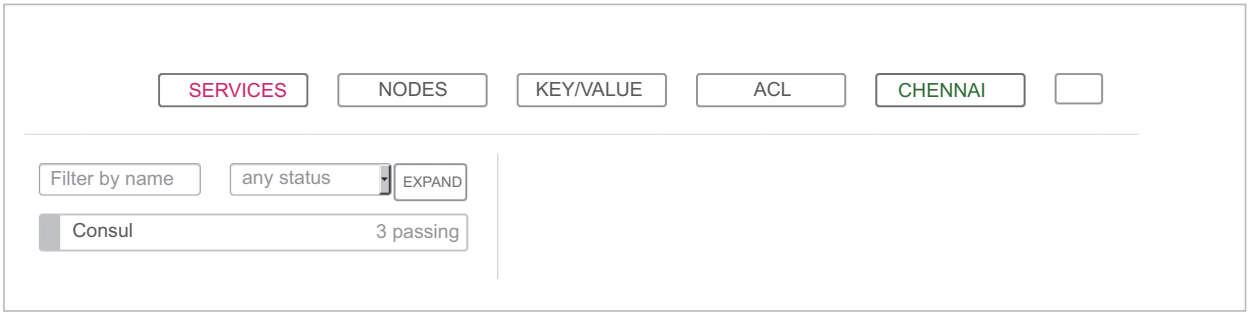


Figure 1: Consul Web UI

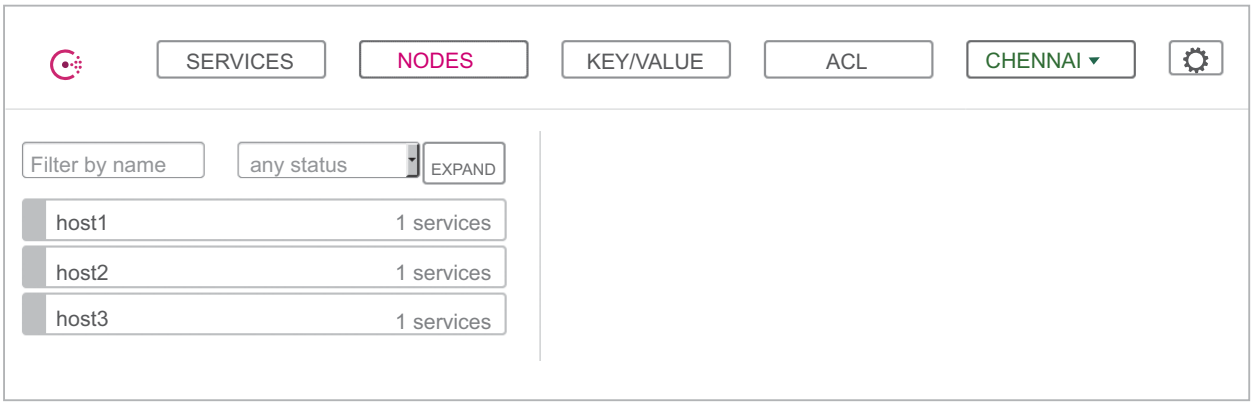


Figure 2: Consul nodes

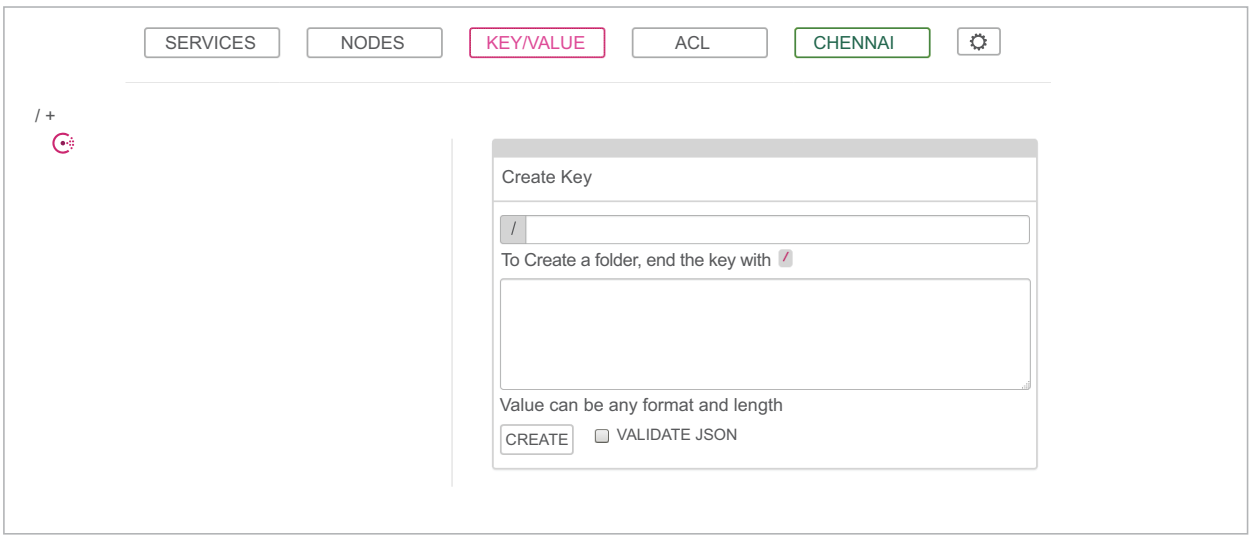


Figure 3: Consul key/value

the following commands from any host:

```
$ consul members
Node   Address  Status  Type  Build Protocol   DC     Segment
host1 192.168.122.140:8301 alive server 1.1.0 2 chennai <all>
host2 192.168.122.208:8301 alive server 1.1.0 2 chennai <all>
host3 192.168.122.59:8301  alive server 1.1.0 2 chennai <all>
```

The web UI listens on Port 8500 on host1, and you can make a Curl request for the same as shown below:

```
$ curl localhost:8500/v1/catalog/nodes
[
  {"ID":"f5de050b-df4d-5235-bb13-ffefe6c81032",
   "Node":"host1",
   "Address":"192.168.122.140",
   "Datacenter":"chennai",
   "TaggedAddresses":{"lan":"192.168.122.140","wan":"192.168.122.140"},
   "Meta":{"consul-network-segment":""},
   "CreateIndex":5,"ModifyIndex":6},
  {"ID":"96552ff0-7692-3553-d6dd-95077365699d",
   "Node":"host2",
   "Address":"192.168.122.208",
   "Datacenter":"chennai",
   "TaggedAddresses":{"lan":"192.168.122.208","wan":"192.168.122.208"},
   "Meta":{"consul-network-segment":""},
   "CreateIndex":10,"ModifyIndex":13},
  {"ID":"fa17e937-86db-597d-7951-ffcb402f439e",
    "Node":"host3",
    "Address":"192.168.122.59",
    "Datacenter":"chennai",
    "TaggedAddresses":{"lan":"192.168.122.59","wan":"192.168.122.59"},
    "Meta":{"consul-network-segment":""},
    "CreateIndex":12,"ModifyIndex":14}
```

You can open the URL *http://192.168.122.140:8500/ui* in a Web browser to see the default Web UI, as shown in Figure 1.

The nodes' link in the UI provides the status of the Consul hosts, as shown in Figure 2.

The 'key/value' link allows you to store and retrieve key values from Consul. Its UI is provided in Figure 3.

Do read the Consul documentation at *https://www.consul.io/docs/index.html* to learn more!

Chapter 19

Ansible Deployment of Monit

This is the 18th article in the DevOps series and it discusses the Ansible deployment of Monit, a free and open source utility for managing and monitoring processes, programs, files, directories and file systems on a *nix system.

Monit is a free and open source process supervision tool for *nix systems. It can also be used to monitor files and directories, and perform maintenance or repair tasks. The system status check can be done on the command line and viewed in a browser. It is written entirely in C and released under the AGPL 3.0 licence. In this 18th article in the DevOps series, we will learn to install and set up Monit for the system, as well as the SSH daemon and Nginx Web server monitoring.

Setting it up

A Debian 9 (x86_64) guest virtual machine (VM) using KVM/QEMU will be set up and monitored using Monit.

The host system is a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0, as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [‘/home/guest/.ansible/plugins/modules’, ‘/
usr/share/ansible/plugins/modules’]
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible playbook and inventory file are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
```

The *inventory/kvm/inventory* file contains the following code:

```
debian ansible_host=192.168.122.197 ansible_connection=ssh ansible_user=debian
ansible_password=password
```

The default Debian 9 installation does not have the *sudo* package installed. Log in to the VM and install the *sudo* package. The 'debian' user also requires *sudo* access:

```
root@debian:~# apt-get install sudo
```

```
root@debian:~# adduser debian sudo
Adding user `debian' to group `sudo'...
Adding user debian to group sudo
Done.
```

You should add an entry in */etc/hosts* file for the Debian VM as shown below:

```
192.168.122.197 debian
```

You can now test connectivity from Ansible to the Debian 9 VM using the following command:

```
$ ansible -i inventory/kvm/inventory debian -m ping
debian | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Installation

The Debian software package repository is first updated and then Monit is installed. The *net-tools* package is installed to provide the *netstat* command in the system. The Monit service is then started using *systemd*. The Ansible playbook for the above tasks is provided below, for reference:

```
---
```

```
- name: Install Monit
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [install]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes

    - name: Install monit
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - net-tools
        - monit

    - name: Start monit service
      systemd:
        name: monit.service
        state: started
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/monit.yml
--tags install -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four 'v's for a more detailed output. The *-K* option prompts for the *sudo* password for the Debian user account.

Web interface

Monit software provides a Web interface that listens on port 2812. The default configuration file for Monit is located at */etc/monit/monitrc*. The Web UI port needs to be enabled with basic login credentials. After making changes to the configuration file, the service needs to be restarted. The Ansible playbook to enable the Monit's Web interface is as follows:

```
- name: Configure UI
```

```
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [ui]

  tasks:
    - lineinfile:
        path: /etc/monit/monitrc
        regexp: 'httpd port 2812'
        line: 'set httpd port 2812 and'

    - lineinfile:
        path: /etc/monit/monitrc
        regexp: '#     allow admin:monit'
        line: '  allow admin:monit'

    - name: Restart monit service
      systemd:
        name: monit.service
        state: restarted

    - wait_for:
        port: 2812
```

The execution of the above playbook to enable the Web interface is shown below:

```
$  ansible-playbook -i inventory/kvm/inventory playbooks/configuration/monit.yml
--tags ui -K
SUDO password:

PLAY [Install Monit] ***************************************
TASK [Gathering Facts] *************************************
ok: [debian]

PLAY [Configure UI] ****************************************

TASK [Gathering Facts] *************************************
ok: [debian]

TASK [lineinfile] ******************************************
```

```
changed: [debian]

TASK [lineinfile] ******************************************
changed: [debian]

TASK [Restart monit service] *******************************
changed: [debian]

TASK [wait_for] ********************************************
ok: [debian]

PLAY [Configure ssh monitoring] ****************************

TASK [Gathering Facts] *************************************
ok: [debian]

PLAY [Configure ssh monitoring] ****************************

TASK [Gathering Facts] *************************************
ok: [debian]

PLAY RECAP *************************************************
debian                     : ok=8    changed=3    unreachable=0    failed=0
```

You can use the netstat command to verify that Monit is listening on port 2812 as shown below:

```
$ netstat -na | grep :2812
tcp        0      0 0.0.0.0:2812            0.0.0.0:*               LISTEN
tcp6       0      0 :::2812                 :::*                    LISTEN
```

The syntax validation of the Monit configuration file can be checked using the following command:

```
$ sudo monit -t
Control file syntax OK
```

The status of the Monit service can be verified as indicated below:

```
$ sudo monit status
Monit 5.20.0 uptime: 2m
```

```
System 'debian'
  status                       Running
  monitoring status            Monitored
  monitoring mode              active
  on reboot                    start
  load average                 [0.00] [0.02] [0.00]
  cpu                          0.4%us 0.3%sy 0.0%wa
  memory usage                 45.5 MB [4.6%]
  swap usage                   0 B [0.0%]
  uptime                       56m
  boot time                    Mon, 09 Jul 2018 15:03:06
  data collected               Mon, 09 Jul 2018 15:57:42
```

A summary report of the Monit service can also be printed in the console output as follows:

```
$ sudo monit summary
Monit 5.20.0 uptime: 4m
┌─────────────────────────────────┬──────────────────────────────┬──────────────
┐
│ Service Name                    │ Status                       │ Type         │
├─────────────────────────────────┼──────────────────────────────┼──────────────
┤
│ debian                          │ Running                      │ System       
│
└─────────────────────────────────┴──────────────────────────────┴──────────────
┘
```

The status of the Monit service can also be checked from the command line using *systemctl*, as shown below:

```
$ sudo systemctl status monit
[sudo] password for debian:

● monit.service - LSB: service and resource monitoring daemon
   Loaded: loaded (/etc/init.d/monit; generated; vendor preset: enabled)
   Active: active (running) since Mon 2018-07-09 15:55:42 IST; 1min 21s ago
     Docs: man:systemd-sysv-generator(8)
  Process: 2293 ExecStop=/etc/init.d/monit stop (code=exited, status=0/SUCCESS)
  Process: 2298 ExecStart=/etc/init.d/monit start (code=exited, status=0/
SUCCESS)
    Tasks: 2 (limit: 4915)
```

```
  CGroup: /system.slice/monit.service
          └─2305 /usr/bin/monit -c /etc/monit/monitrc
```

```
Jul 09 15:55:42 debian systemd[1]: Stopped LSB: service and resource monitoring
daemon.
Jul 09 15:55:42 debian systemd[1]: Starting LSB: service and resource
monitoring daemon...
```

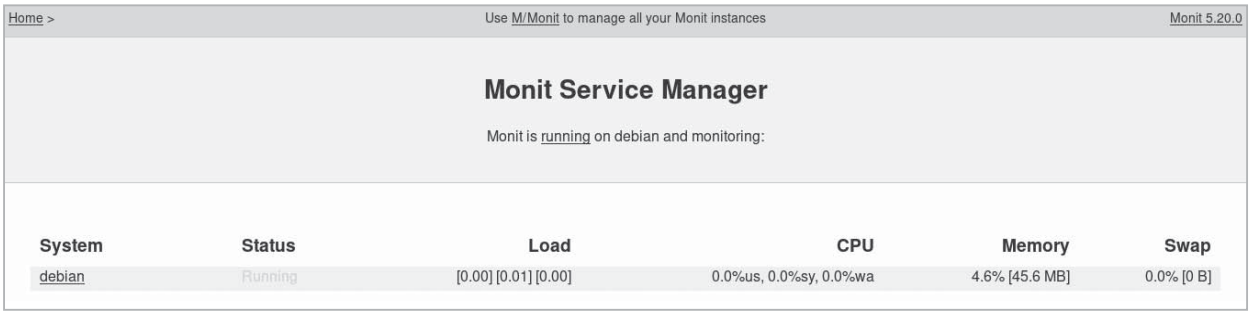


Figure 1: Monit Web UI

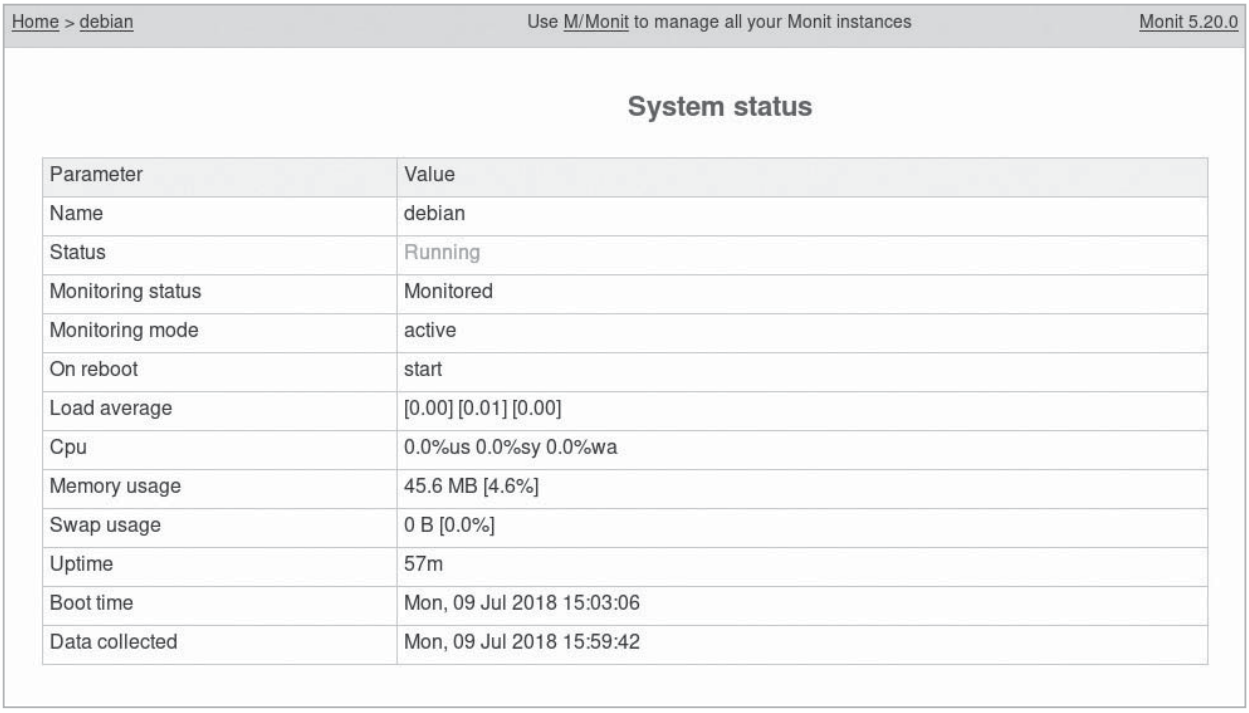


Figure 2: Monit Web UI status

```
Jul 09 15:55:42 debian monit[2298]: Starting daemon monitor: monit.
Jul 09 15:55:42 debian systemd[1]: Started LSB: service and resource monitoring
daemon.
```

You can now open the URL *http://192.168.122.197:2812* in a browser on the host system to see the default Monit home page, as shown in Figure 1.

When you click on 'debian' under the *System* column, a more detailed status output is made available, as shown in Figure 2.

SSH

We can now set up Monit to monitor the SSH daemon running inside the VM. A check block for SSH needs to be added to the Monit configuration file and the service needs to be restarted. The Ansible playbook with the required SSH monitoring configuration is given below:

```
- name: Configure monitoring for nginx
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [ssh]

  tasks:
    - name: Add ssh monitoring
      blockinfile:
        path: /etc/monit/monitrc
        marker_begin: “ssh BEGIN”
        marker_end: “ssh END”
        block: |
          check process sshd with pidfile /var/run/sshd.pid
            group system
            group sshd
            start program = “/etc/init.d/ssh start”
            stop  program = “/etc/init.d/ssh stop”
            if failed host localhost port 22 with proto ssh then restart
            if 5 restarts with 5 cycles then timeout
            depend on sshd_bin
            depend on sshd_rc
            depend on sshd_rsa_key
            depend on sshd_dsa_key

          check file sshd_bin with path /usr/sbin/sshd
            group sshd
            include /etc/monit/templates/rootbin

          check file sshd_rsa_key with path /etc/ssh/ssh_host_rsa_key
            group sshd
            include /etc/monit/templates/rootstrict

          check file sshd_dsa_key with path /etc/ssh/ssh_host_ecdsa_key
            group sshd
```

```
          include /etc/monit/templates/rootstrict

        check file sshd_rc with path /etc/ssh/sshd_config
          group sshd
          include /etc/monit/templates/rootrc

  - name: Restart monit service
    systemd:
      name: monit.service
      state: restarted
```

The above playbook can be invoked as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/monit.yml
--tags ssh -vv -K
```

The Monit status command in the command line now produces a more detailed output on monitoring the SSH daemon as shown below:

```
$ sudo monit status
Monit 5.20.0 uptime: 0m

Process 'sshd'
  status                            Running
  monitoring status                 Monitored
  monitoring mode                   active
  on reboot                         start
  pid                               381
  parent pid                        1
  uid                               0
  effective uid                     0
  gid                               0
  uptime                            1h 6m
  threads                           1
  children                          5
  cpu                               0.0%
  cpu total                         0.0%
  memory                            0.6% [6.2 MB]
  memory total                      3.3% [33.2 MB]
  port response time                7.847 ms to localhost:22 type TCP/IP  protocol
SSH
  data collected                    Mon, 09 Jul 2018 16:09:09
```

```
File 'sshd_bin'
  status                      Accessible
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  permission                  755
  uid                         0
  gid                         0
  size                        772.5 kB
  timestamp                   Thu, 05 Apr 2018 13:18:00
  checksum                    0c4b6da99164fb9a9291966b1c5b9902 (MD5)
  data collected              Mon, 09 Jul 2018 16:09:09

File 'sshd_rsa_key'
  status                      Accessible
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  permission                  600
  uid                         0
  gid                         0
  size                        1.6 kB
  timestamp                   Thu, 05 Apr 2018 13:18:18
  checksum                    de068898089f42cd3ada992e18af4a23 (MD5)
  data collected              Mon, 09 Jul 2018 16:09:09

File 'sshd_dsa_key'
  status                      Accessible
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  permission                  600
  uid                         0
  gid                         0
  size                        227 B
  timestamp                   Thu, 05 Apr 2018 13:18:18
  checksum                    abbf249f7ca23dd6f8186072dc0173a3 (MD5)
  data collected              Mon, 09 Jul 2018 16:09:09

File 'sshd_rc'
  status                      Accessible
  monitoring status           Monitored
```

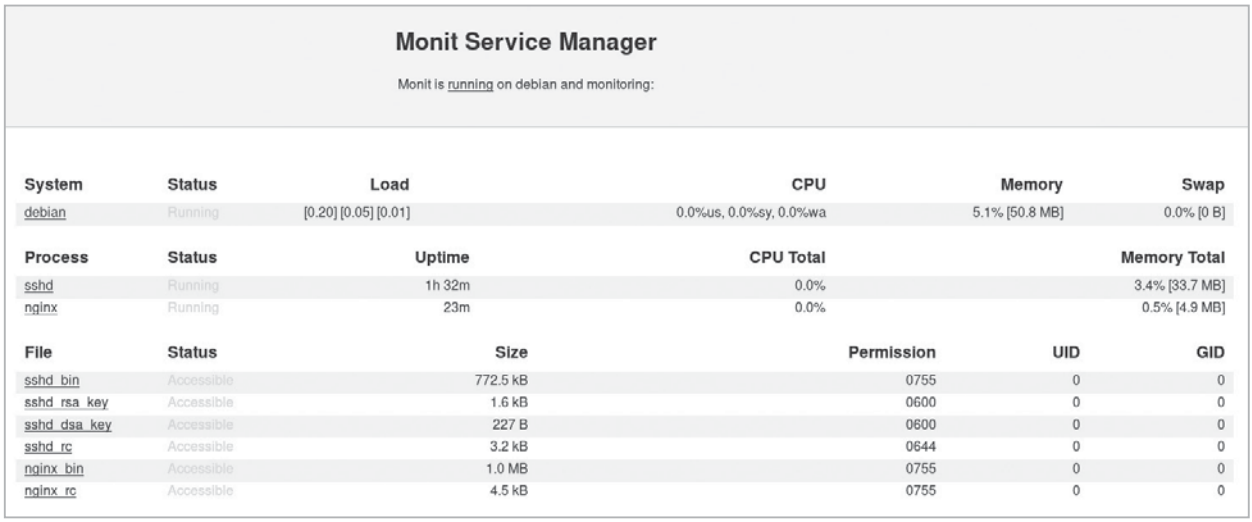
```
  monitoring mode             active
  on reboot                   start
  permission                  644
  uid                         0
  gid                         0
  size                        3.2 kB
  timestamp                   Thu, 05 Apr 2018 13:18:17
  checksum                    bbad7ed242a834e831c7066901cee49e (MD5)
  data collected              Mon, 09 Jul 2018 16:09:09

System 'debian'
  status                      Running
  monitoring status           Monitored
  monitoring mode             active
  on reboot                   start
  load average                [0.00] [0.00] [0.00]
  cpu                         0.0%us 0.0%sy 0.0%wa
  memory usage                47.6 MB [4.8%]
  swap usage                  0 B [0.0%]
  uptime                      1h 6m
  boot time                   Mon, 09 Jul 2018 15:03:06
  data collected              Mon, 09 Jul 2018 16:09:09
```

Nginx

We can also set up Nginx on the guest VM and monitor it using Monit. The software package repository is first updated and Nginx is then installed. The service is started and we wait for it to listen on port 80. The Nginx monitoring check is then added to the Monit configuration file and the Monit service is restarted. The Ansible playbook for the above tasks is provided below, for reference:

```
- name: Configure ssh monitoring
  hosts: debian
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [nginx]

  tasks:
    - name: Update the software package repository
      apt:
        update_cache: yes
    - name: Install nginx
```

```
    package:
      name: "{{ item }}"
      state: latest
    with_items:
      - nginx

  - name: Start nginx
    service:
      name: nginx
      state: started

  - wait_for:
      port: 80

  - name: Add nginx monitoring
    blockinfile:
      path: /etc/monit/monitrc
      marker_begin: "nginx BEGIN"
      marker_end: "nginx END"
      block: |
        check process nginx with pidfile /var/run/nginx.pid
          group www
          group nginx
          start program = "/etc/init.d/nginx start"
          stop program = "/etc/init.d/nginx stop"
          if 5 restarts with 5 cycles then timeout
          depend nginx_bin
          depend nginx_rc
```



Monit Service Manager

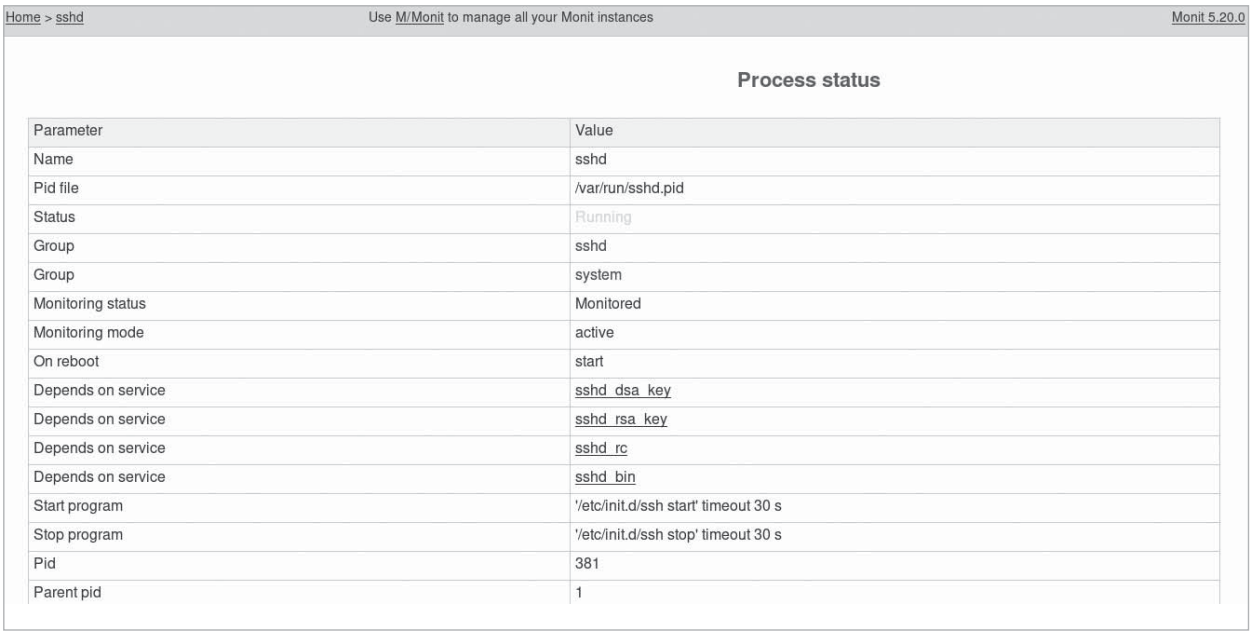
Monit is running on debian and monitoring:

System	Status	Load	CPU	Memory	Swap
debian	Running	[0.20] [0.05] [0.01]	0.0%us, 0.0%sy, 0.0%wa	5.1% [50.8 MB]	0.0% [0 B]

Process	Status	Uptime	CPU Total	Memory Total
sshd	Running	1h 32m	0.0%	3.4% [33.7 MB]
nginx	Running	23m	0.0%	0.5% [4.9 MB]

File	Status	Size	Permission	UID	GID
sshd_bin	Accessible	772.5 kB	0755	0	0
sshd_rsa_key	Accessible	1.6 kB	0600	0	0
sshd_dsa_key	Accessible	227 B	0600	0	0
sshd_rc	Accessible	3.2 kB	0644	0	0
nginx_bin	Accessible	1.0 MB	0755	0	0
nginx_rc	Accessible	4.5 kB	0755	0	0

Figure 3: Monit Web UI with SSH and Nginx

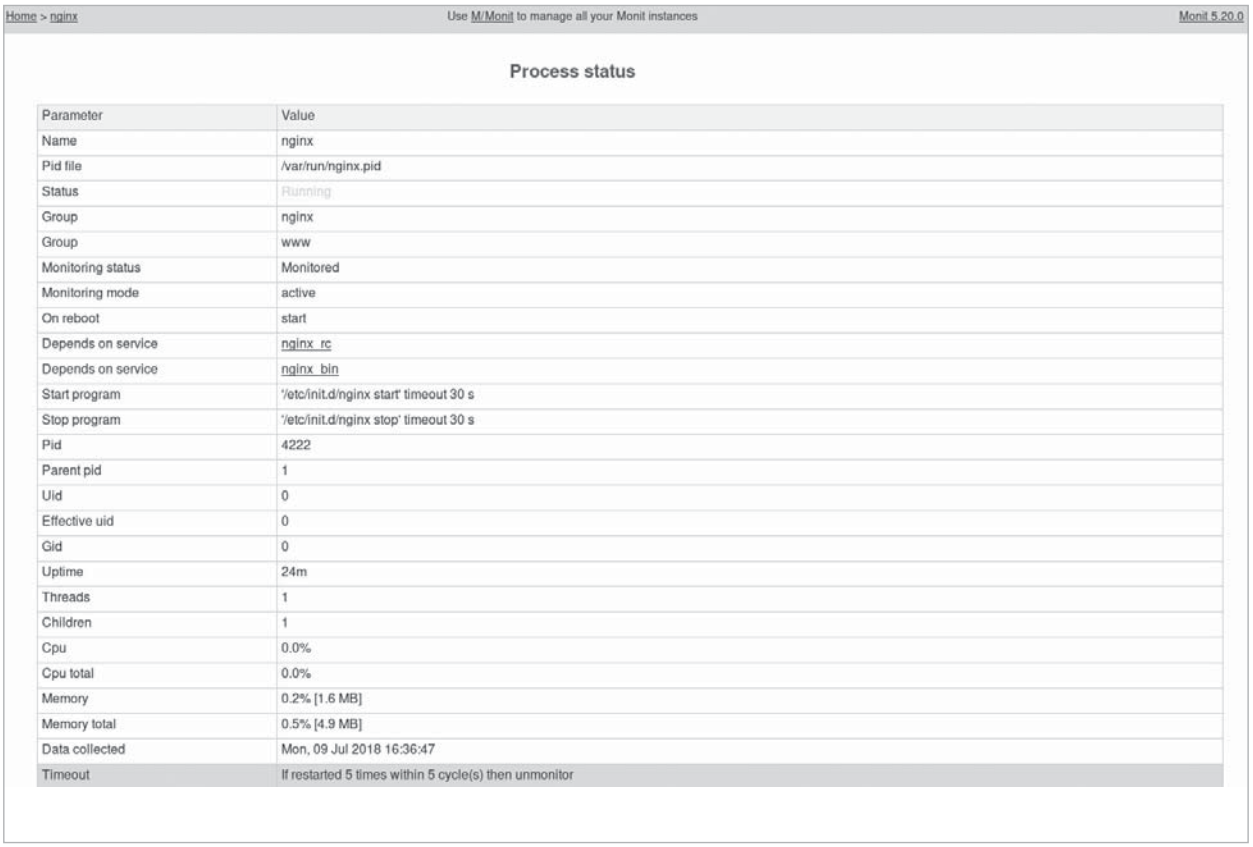


Home > sshd — Use M/Monit to manage all your Monit instances — Monit 5.20.0

Process status

Parameter	Value
Name	sshd
Pid file	/var/run/sshd.pid
Status	Running
Group	sshd
Group	system
Monitoring status	Monitored
Monitoring mode	active
On reboot	start
Depends on service	sshd_dsa_key
Depends on service	sshd_rsa_key
Depends on service	sshd_rc
Depends on service	sshd_bin
Start program	'/etc/init.d/ssh start' timeout 30 s
Stop program	'/etc/init.d/ssh stop' timeout 30 s
Pid	381
Parent pid	1

Figure 4: Monit with the SSH process



Home > nginx — Use M/Monit to manage all your Monit instances — Monit 5.20.0

Process status

Parameter	Value
Name	nginx
Pid file	/var/run/nginx.pid
Status	Running
Group	nginx
Group	www
Monitoring status	Monitored
Monitoring mode	active
On reboot	start
Depends on service	nginx_rc
Depends on service	nginx_bin
Start program	'/etc/init.d/nginx start' timeout 30 s
Stop program	'/etc/init.d/nginx stop' timeout 30 s
Pid	4222
Parent pid	1
Uid	0
Effective uid	0
Gid	0
Uptime	24m
Threads	1
Children	1
Cpu	0.0%
Cpu total	0.0%
Memory	0.2% [1.6 MB]
Memory total	0.5% [4.9 MB]
Data collected	Mon, 09 Jul 2018 16:36:47
Timeout	If restarted 5 times within 5 cycle(s) then unmonitor

Figure 5: Monit Nginx

```
        check file nginx_bin with path /usr/sbin/nginx
          group nginx
          include /etc/monit/templates/rootbin
        check file nginx_rc with path /etc/init.d/nginx
          group nginx
```

```
            include /etc/monit/templates/rootbin
```

```
    - name: Restart monit service
      systemd:
        name: monit.service
        state: restarted
The above playbook can be invoked as follows:
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/monit.yml
--tags nginx -vv -K
```

The Monit home page now contains the status of the system, the SSH daemon and the Nginx Web server, as shown in Figure 3.

Clicking on the 'sshd' or 'nginx' link on the Monit home page provides a more detailed status page, as shown in Figures 4 and 5, respectively.

You are encouraged to read the Monit manual at *https://mmonit.com/monit/documentation/monit.html* to learn more about its options and usage.

Chapter 20

Ansible Deployment of Sensu and Uchiwa

While Sensu is a free and open source monitoring and telemetry tool, Uchiwa provides the free and open source dashboard for Sensu. This article covers the Ansible deployment of Sensu and Uchiwa.

Sensu Core is a free and open source monitoring and telemetry solution. You can use it to monitor services, an application's health, servers and important KPIs. It is primarily a monitoring event pipeline which allows you to filter or amend incoming events, and send alerts and notifications. It uses JSON for all its configuration files and integrates well with automation tools. Uchiwa is a free and open source dashboard for Sensu, written in the Go programming language. It also requires Node.js for front-end assets, and JavaScript. Both Sensu Core and Uchiwa are released under the MIT licence.

Setting them up

A CentOS 7 (x86_64) guest virtual machine (VM) using KVM/QEMU will be used to set up Sensu.

The host system is a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0 as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/guest/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory, playbook and configuration files are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
sensu ansible_host=192.168.122.43 ansible_connection=ssh ansible_user=centos
ansible_password=centos123
```

A 'centos' user is created in the guest VM and sudo access is provided for this user using the 'visudo' command. SELinux needs to allow access to port 3000 for the Uchiwa dashboard. You should also add an entry in */etc/hosts* file for the CentOS VM as shown below:

```
192.168.122.43 sensu
```

You can now test connectivity from Ansible to the CentOS 7 VM using the following command:

```
$ ansible -i inventory/kvm/inventory sensu -m ping
sensu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Installing the repositories

The first step is to install the repositories required for Sensu. The Extra Packages for Enterprise Linux (EPEL) repository is added. The *sensu.repo* also needs to be created in the guest VM, whose file contents are shown below:

```
[sensu]
name=sensu
baseurl=https://sensu.global.ssl.fastly.net/yum/$releasever/$basearch/
gpgcheck=0
enabled=1
```

The Ansible playbook for setting up the pre-requisite repositories is given below:

```
---

- name: Repository setup
  hosts: sensu
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [repo]

  tasks:
    - name: Install epel-release
      yum:
        name: epel-release
        state: present

    - name: Create sensu.repo
      copy:
        src: ../../files/sensu.repo
        dest: /etc/yum.repos.d/sensu.repo
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/sensu.yml
--tags repo -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four v's for a more detailed output. The *-K* option prompts for the sudo password for the centos user account.

Redis

The Redis in-memory database is used as a data store and for transport. You can install it using the YUM tool. In the following example, the protected-mode in Redis configuration is disabled as we are in development mode and the server is started. We wait for the Redis server to run on port 6379. The Ansible playbook for installing, configuring and starting Redis is as follows:

```
- name: Install Redis
  hosts: sensu
  become: yes
  become_method: sudo
  gather_facts: true
```

```
  tags: [redis]

  tasks:
    - name: Install Redis
      yum:
        name: redis
        state: present

    - lineinfile:
        path: /etc/redis.conf
        regexp: '^protected-mode yes'
        line: 'protected-mode no'

    - name: Start Redis
      systemd:
        name: redis
        state: started

    - wait_for:
        port: 6379
```

The above playbook can be executed as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/sensu.yml
--tags redis -vv -K
```

Sensu

You can now proceed to install Sensu and the Uchiwa dashboard. The *jq* tool is used to process JSON data in the command line. The Sensu *config.json* file contents specify the Redis transport and API access information as shown below:

```
{
 "transport": {
   "name": "redis"
 },
 "api": {
   "host": "127.0.0.1",
   "port": 4567
 }
}
```

The *client.json* file sets the environment to 'development' and the

subscription name to 'linux' as shown below:

```
{
 "client": {
   "environment": "development",
   "subscriptions": [
     "linux"
   ]
 }
}
```

The Uchiwa dashboard configuration file is given by *uchiwa.json*, which includes information on Sensu as well as the host and port where Sensu should run. The *uchiwa.json* file contents are as follows:

```
{
 "sensu": [
   {
     "name": "sensu",
     "host": "127.0.0.1",
     "port": 4567,
     "timeout": 10
   }
 ],
 "uchiwa": {
   "host": "0.0.0.0",
   "port": 3000,
   "refresh": 10
 }
}
```

The above configuration files are copied to the */etc/sensu* directory in their respective locations. The firewall rule to allow port 3000 for the Uchiwa dashboard is then enabled. The *sensu-{api, client, server}* and the Uchiwa dashboard services are then started. The Ansible playbook for the above tasks is provided below for reference:

```
- name: Install sensu, uchiwa
  hosts: sensu
  become: yes
  become_method: sudo
  gather_facts: true
```

```
  tags: [sensu]

  tasks:
    - name: Install sensu and packages
      yum:
        name: "{{ item }}"
        state: present
      with_items:
        - sensu
        - uchiwa
        - jq
    - name: Create config.json
      copy:
        src: ../../files/config.json
        dest: /etc/sensu/config.json
    - name: Create client.json
      copy:
        src: ../../files/client.json
        dest: /etc/sensu/conf.d/client.json
    - name: Create uchiwa.json
      copy:
        src: ../../files/uchiwa.json
        dest: /etc/sensu/uchiwa.json

    - file:
        path: /etc/sensu
        owner: sensu
        group: sensu
        recurse: yes

    - firewalld:
        port: 3000/tcp
        state: enabled

    - name: Start services
      systemd:
        name: "{{ item }}"
        state: started
      with_items:
        - sensu-server
        - sensu-api
        - sensu-client
```

```
- uchiwa
```

The execution output for installing Sensu and Uchiwa is shown below:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/sensu.yml
--tags sensu -K
SUDO password:

PLAY [Repository setup] ************************************

TASK [Gathering Facts] *************************************
ok: [sensu]

PLAY [Install Redis] ***************************************

TASK [Gathering Facts] *************************************ok: [sensu]

PLAY [Install sensu, uchiwa] *******************************

TASK [Gathering Facts] *************************************
ok: [sensu]

TASK [Install sensu and packages] **************************
changed: [sensu] => (item=[‘sensu’, ‘uchiwa’, ‘jq’])

TASK [Create config.json] ***********************************
changed: [sensu]

TASK [Create client.json] ***********************************
changed: [sensu]

TASK [Create uchiwa.json] ***********************************
changed: [sensu]

TASK [file] *************************************************
changed: [sensu]

TASK [firewalld] ********************************************
changed: [sensu]

TASK [Start services] ***************************************
changed: [sensu] => (item=sensu-server)
```

```
changed: [sensu] => (item=sensu-api)
changed: [sensu] => (item=sensu-client)
changed: [sensu] => (item=uchiwa)

PLAY [Enable checks] ************************************

TASK [Gathering Facts] ***********************************
ok: [sensu]

PLAY RECAP ***********************************************
sensu                      : ok=11   changed=7    unreachable=0    failed=0
```

You can now verify that Sensu is running fine by querying the API for clients using the *Curl* command and parsing the output using *jq*, as follows:

```
$ curl -s http://127.0.0.1:4567/clients | jq .
[
  {
    “name”: “localhost.localdomain”,
    “address”: “192.168.122.164”,
    “environment”: “development”,
    “subscriptions”: [
      “linux”,
      “client:localhost.localdomain”
    ],
    “version”: “1.4.3”,
    “timestamp”: 1533722644
  }

]
```

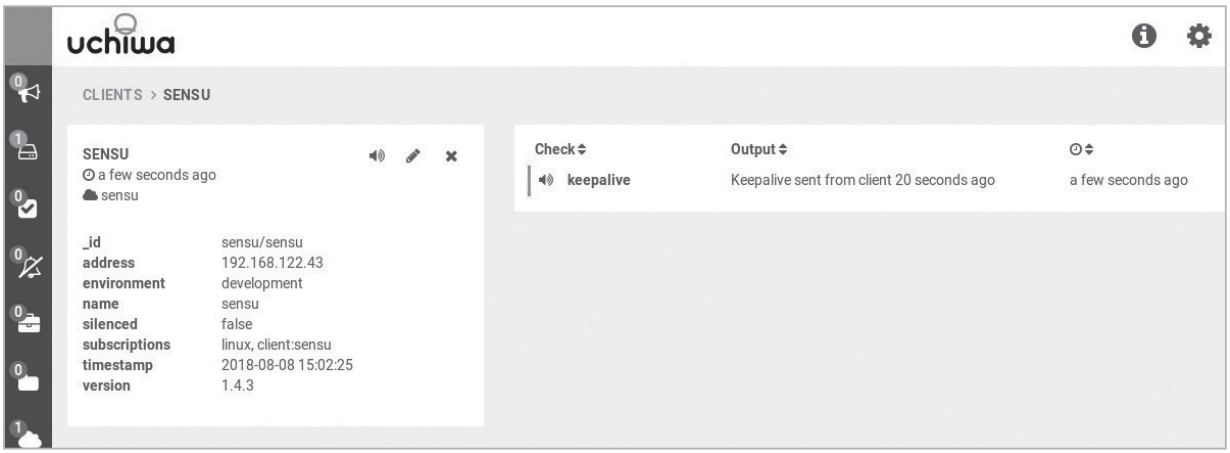


Figure 1: Sensu client Web UI

Figure 2: Sensu data centre Web UI

The Uchiwa dashboard is available at *http://192.168.122.43:3000* in the host system, and you can view the Sensu client Web interface as shown in Figure 1.

The Sensu data center view is shown in Figure 2.

Checks

The monitoring checks for CPU, disk and memory can be set up on the guest VM and viewed in the Uchiwa dashboard. The *sensu-install* command is used to install the Ruby script checks that will be run periodically. The check configurations for CPU, disk and memory are copied to the */etc/sensu/conf.d* directory and provided below for reference:

```
Listing 6.1: check_cpu_linux.json
{
 “checks”: {
   “check-cpu-linux”: {
     “command”: “/opt/sensu/embedded/bin/check-cpu.rb -w 80 -c 90 “,
     “interval”: 60,
     “subscribers”: [
       “linux”
     ]
   }
 }
}
```

```
Listing 6.2: check_disk_usage_linux.json
{
 “checks”: {
   “check-disk-usage-linux”: {
```

```
    "command": "/opt/sensu/embedded/bin/check-disk-usage.rb -w 80 -c 90",
    "interval": 60,
    "subscribers": [
      "linux"
    ]
  }
 }
}
```

Listing 6.3: check_memory_linux.json

```
{
 "checks": {
  "check_memory_linux": {
    "command": "/opt/sensu/embedded/bin/check-memory-percent.rb -w 90 -c 95",
    "interval": 60,
    "subscribers": [
      "linux"
    ]
  }
 }
}
```

Finally, the Sensu services are restarted. The Ansible playbook to install the checks is given below, for reference:

```
- name: Enable checks
  hosts: sensu
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [checks]

  tasks:
    - name: Install checks
      command: "sensu-install -p {{ item }}"
      args:
        chdir: /opt/sensu/embedded/bin
      with_items:
        - cpu-checks
        - disk-checks
        - memory-checks

    - name: Create check json files
```

```
    copy:
      src: "../../files/{{ item }}.json"
      dest: "/etc/sensu/conf.d/{{ item }}.json"
    with_items:
      - check_cpu_linux
      - check_disk_usage_linux
      - check_memory_linux
  - name: Restart services
    systemd:
      name: "{{ item }}"
      state: restarted
    with_items:
      - sensu-server
      - sensu-api
      - sensu-client
- uchiwa
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/sensu.yml
```

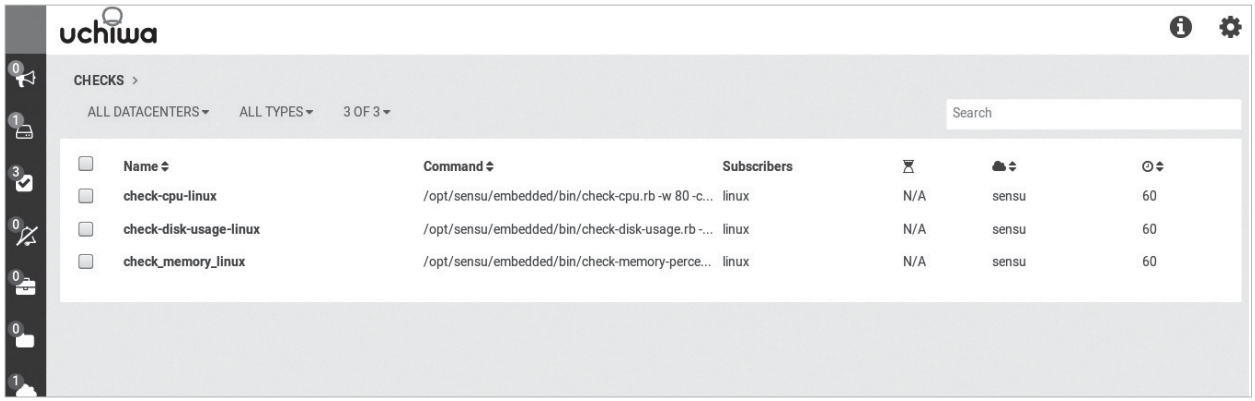


Figure 3: List of Sensu checks

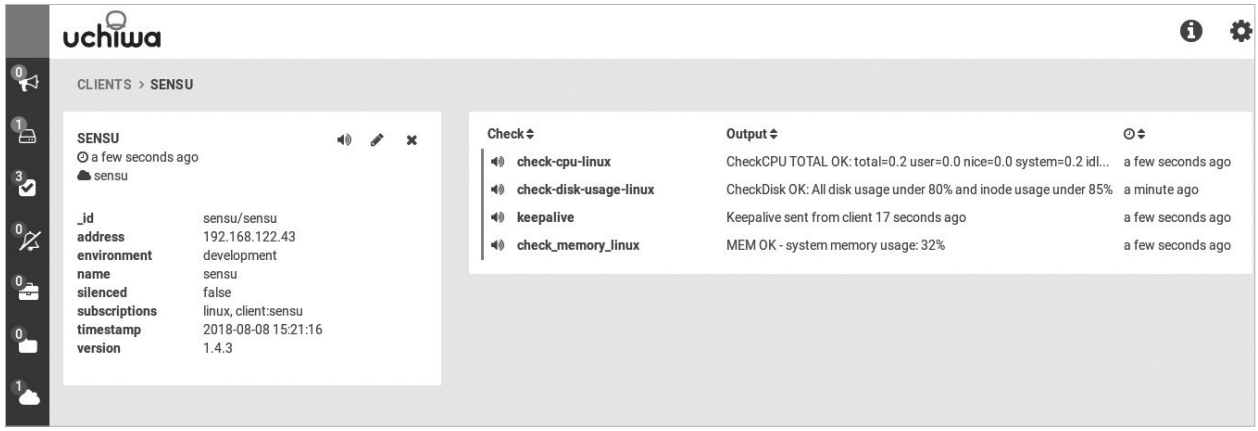


Figure 4: Output of Sensu checks

```
--tags checks -vv -K
```

The Sensu dashboard will now have the installed checks as shown in Figure 3.

The results of the check output are also available in the dashboard as shown in Figure 4.

You are encouraged to read the Sensu Core documentation available at *https://docs.sensu.io/sensu-core/1.4/* to learn more about the framework and its usage.

Chapter 21

Ansible Deployment of Bugzilla

Bugzilla is Free/Libre and Open Source (FLOSS) bug tracker software developed by the Mozilla project. In this 20th article in the DevOps series, we will learn how to set up and configure Bugzilla.

Bugzilla is Web based and written using the Perl programming language. Released on August 26, 1998, it is used by a number of other FLOSS projects such as the Linux kernel, GNOME, Red Hat, Apache and KDE. It requires a database, Web server and at least Perl 5 to run. You can also configure it to use an SMTP mail server to send emails. It has been released under the Mozilla Public License.

Setup

A CentOS 7 (x86_64) guest virtual machine (VM) using KVM/QEMU will be used to install and configure Bugzilla. The host system is a Parabola GNU/Linux-libre x86_64 system, and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0 as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [‘/home/guest/.ansible/plugins/modules’, ‘/
usr/share/ansible/plugins/modules’]
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory, playbook and configuration files are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
bugzilla ansible_host=192.168.122.87 ansible_connection=ssh ansible_user=centos
ansible_password=password
```

A 'centos' user is created in the guest VM, and sudo access is provided for this user using the *visudo* command. SELinux needs to allow access for port 80. You should also add an entry in the */etc/hosts* file for the CentOS VM, as shown below:

```
192.168.122.87 bugzilla
```

You can now test connectivity from Ansible to the CentOS 7 VM using the following command:

```
$ ansible -i inventory/kvm/inventory bugzilla -m ping
sensu | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Repository

The installation of Bugzilla requires additional repositories — *deltarpm* and *epel-release*. These need to be installed first, and then 'yum update' is required to update the cache. The Ansible playbook for installing the repositories is given below:

```
---
- name: Repository setup
  hosts: bugzilla
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [repo]

  tasks:
    - name: Install additional repositories
      yum:
        name: "{{ item }}"
        state: present
      with_items:
        - "deltarpm"
```

```
      - "epel-release"

  - name: Yum update
    yum: name=* update_cache=yes state=present
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/bugzilla.
yml --tags repo -vv -K
```

The '-vv' represents the verbosity in the Ansible output. You can use up to four v's for a more detailed output. The '-K' option prompts for the sudo password for the 'centos' user account.

HTTPD

Bugzilla does require a Web server and the Apache HTTP (httpd) server package, and its dependencies need to be installed. The firewall needs to allow port 80. After installing the packages, the httpd service is started, and we wait for the server to listen on port 80. The Ansible playbook for the above tasks is as follows:

```
- name: Install httpd
  hosts: bugzilla
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [httpd]

  tasks:

  - name: Install httpd and dependency packages
    yum:
      name: "{{ item }}"
      state: present
    with_items:
      - "httpd"
      - "httpd-devel"
      - "mod_ssl"
      - "mod_perl"
      - "mod_perl-devel"

  - firewalld:
```

```
      port: 80/tcp
      state: enabled

  - name: Start httpd
    service:
      name: httpd
      state: started

  - wait_for:
      port: 80
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/bugzilla.
yml --tags httpd -vv -K
```

You can now open the following URL *http://192.168.122.87* on your host system, and you should see the default httpd home page, as shown in Figure 1.

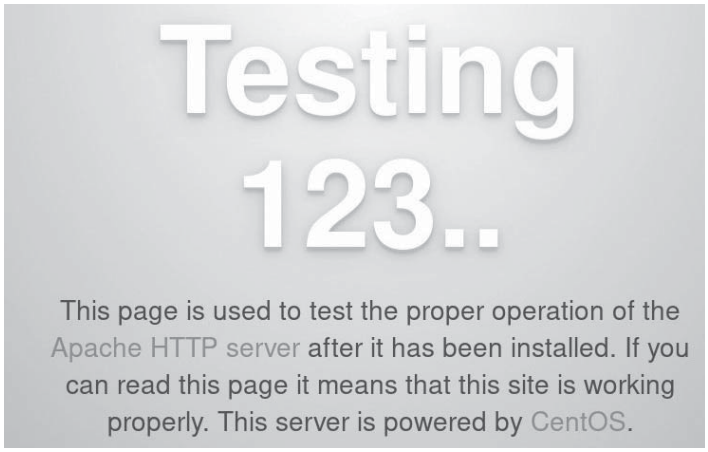


Figure 1: httpd home page

MariaDB

Bugzilla requires a database and we shall use MariaDB, the community-developed fork of the MySQL database. MariaDB and a few dependency packages need to be installed. The */etc/my.cnf* configuration file needs to be updated with '*max_allowed_packet=4M*' as required by Bugzilla. The database server is then started, and we wait for the server to listen on port 3306. A 'bugs' user account is created in MariaDB for use with Bugzilla along with a password. A 'bugs' database is also created. The Ansible playbook for setting up MariaDB is as follows:

```
- name: Install and configure MariaDB
  hosts: bugzilla
```

```
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [mariadb]

  tasks:
    - name: Install MariaDB and dependency packages
      yum:
        name: "{{ item }}"
        state: present
      with_items:
        - "mariadb-server"
        - "mariadb"
        - "mariadb-devel"
        - "MySQL-python"
    - name: Allow maximum allowed packet
      lineinfile:
        path: /etc/my.cnf
        line: 'max_allowed_packet=4M'
    - name: Start mariadb
      service:
        name: mariadb
        state: started

    - wait_for:
        port: 3306

    - name: Create bugs database user
      mysql_user:
        name: bugs
        password: bugs123
        priv: '*.*:ALL,GRANT'
        state: present

    - name: Create a database
      mysql_db:
        name: bugs
        state: present
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/bugzilla.
yml --tags mariadb -vv -K
```

Bugzilla

Perl is required for Bugzilla, and we need to install the same and a few other Comprehensive Perl Archive Network (CPAN) dependencies. The Bugzilla-5.0 tarball is downloaded and copied to */var/www/html/bugzilla*. The sources come with a *checksetup.pl* script that you can invoke to see if the minimum installation requirements are met. The first time you invoke the script, it will exit with errors stating that the Perl modules are missing. You can then use the *install-module.pl* script to install the Perl dependencies. You should then rerun the *checksetup.pl* script to check against the installed Perl modules. The */var/www/html/bugzilla/localconfig* file needs to be updated with the database password. A *bugzilla.conf* for the Web server is created at */etc/httpd/conf.d/bugzilla*, for which the file contents are shown below:

```
<VirtualHost *:80>
    DocumentRoot /var/www/html/bugzilla/
</VirtualHost>
<Directory /var/www/html/bugzilla>
    AddHandler cgi-script .cgi
    Options +Indexes +ExecCGI
    DirectoryIndex index.cgi
    AllowOverride Limit FileInfo Indexes
</Directory>
```

The Ansible playbook to set up Bugzilla is as follows:

```
- name: Install and configure Bugzilla
  hosts: bugzilla
  become: yes
  become_method: sudo
  gather_facts: true
  tags: [bugzilla]

  tasks:
    - name: Install Bugzilla dependency packages
      yum:
        name: "{{ item }}"
        state: present
        exclude: "perl-homedir"
      with_items:
        - "gcc"
        - "gcc-c++"
```

```
        - "graphviz"
        - "graphviz-devel"
        - "patchutils"
        - "gd"
        - "gd-devel"
        - "wget"
        - "perl*"
        - "perl-CPAN"

    - name: Download Bugzilla
      unarchive:
        src: https://ftp.mozilla.org/pub/mozilla.org/webtools/bugzilla-5.0.tar.
gz
        dest: /var/www/html
        remote_src: yes

    - name: Rename bugzilla-5.0 to bugzilla
      command: mv /var/www/html/bugzilla-5.0 /var/www/html/bugzilla

    - name: Install Perl modules
      command: ./checksetup.pl
      args:
        chdir: /var/www/html/bugzilla
      ignore_errors: true

    - name: Install Perl modules
      command: /usr/bin/perl install-module.pl --all
      args:

        chdir: /var/www/html/bugzilla
    - name: Install Perl modules
      command: ./checksetup.pl
      args:
        chdir: /var/www/html/bugzilla
      ignore_errors: true

    - name: Update localconfig
      lineinfile:
        path: /var/www/html/bugzilla/localconfig
        regexp: '^$db_pass'
        line: "$db_pass='bugs123';"

    - name: Update htaccess
```

```
    lineinfile:
      path: /var/www/html/bugzilla/.htaccess
      regexp: '^Options -'
      line: "# Options - Indexes"
```

```
  - name: Create /etc/httpd/conf.d/bugzilla.conf
    copy:
      src: ../../files/bugzilla.conf
      dest: /etc/httpd/conf.d/bugzilla.conf
      mode: 0644
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/bugzilla.
yml --tags bugzilla -vv -K
```

Final configuration

A final configuration step is to run *checksetup.pl* manually, as it prompts for user credentials. You can change to the */var/www/html/bugzilla* directory, and execute the *sudo ./checksetup.pl* script, which will again check the installed Perl modules, create tables in the database, and prompt for the Administrator user account name and password. This has to be entered manually. A sample execution output is shown below for reference:

```
[centos@localhost bugzilla]$ sudo ./checksetup.pl
 * This is Bugzilla 5.0 on perl 5.16.3
 * Running on Linux 3.10.0-862.el7.x86_64 #1 SMP Fri Apr 20 16:44:24 UTC 2018

Checking perl modules...
Checking for              CGI.pm (v3.51)     ok: found v4.40
Checking for          Digest-SHA (any)       ok: found v5.85
Checking for            TimeDate (v2.23)     ok: found v2.24
Checking for            DateTime (v0.75)     ok: found v1.50
Checking for    DateTime-TimeZone (v1.64)     ok: found v2.19
Checking for           DBI (v1.614)    ok: found v1.627
Checking for     Template-Toolkit (v2.24)     ok: found v2.27
Checking for    Email-Sender (v1.300011) ok: found v1.300031
Checking for    Email-MIME (v1.904)    ok: found v1.946
Checking for             URI (v1.55)     ok: found v1.74
Checking for    List-MoreUtils (v0.32)     ok: found v0.428
Checking for    Math-Random-ISAAC (v1.0.1)   ok: found v1.004
Checking for     File-Slurp (v9999.13) ok: found v9999.19
```

```
Checking for              JSON-XS (v2.01)     ok: found v3.04

Checking available perl DBD modules...
Checking for               DBD-Pg (v2.7.0)    not found
Checking for      DBD-mysql (v4.001)    ok: found v4.023
Checking for           DBD-SQLite (v1.29)     ok: found v1.58
Checking for           DBD-Oracle (v1.19)     not found

The following Perl modules are optional:
Checking for                   GD (v1.20)     ok: found v2.69
Checking for            Chart (v2.4.1)    ok: found v2.4.10
Checking for         Template-GD (any)       ok: found v1.56
Checking for          GDTextUtil (any)       ok: found v0.86
Checking for             GDGraph (any)       ok: found v1.54
Checking for         MIME-tools (v5.406)    ok: found v5.509
Checking for          libwww-perl (any)       ok: found v6.35
Checking for             XML-Twig (any)       ok: found v3.52
Checking for        PatchReader (v0.9.6)    ok: found v0.9.6
Checking for            perl-ldap (any)       ok: found v0.65
Checking for          Authen-SASL (any)       ok: found v2.16
Checking for         Net-SMTP-SSL (v1.01)     ok: found v1.04
Checking for           RadiusPerl (any)       ok: found v0.27
Checking for            SOAP-Lite (v0.712)    ok: found v1.27
Checking for        XMLRPC-Lite (v0.712)    ok: found v0.717
Checking for             JSON-RPC (any)       ok: found v1.06
Checking for           Test-Taint (v1.06)     ok: found v1.06
Checking for          HTML-Parser (v3.67)     ok: found v3.72
Checking for        HTML-Scrubber (any)       ok: found v0.17
Checking for                Encode (v2.21)     ok: found v2.98
Checking for        Encode-Detect (any)       ok: found v1.01
Checking for        Email-Reply (any)       ok: found v1.204

Checking for HTML-FormatText-WithLinks(v0.13) ok: found v0.15
Checking for          TheSchwartz (v1.07)     ok: found v1.12
Checking for       Daemon-Generic (any)       ok: found v0.85
Checking for         mod_perl (v1.999022) ok: found v2.000010
Checking for     Apache-SizeLimit (v0.96)     ok: found v0.97
Checking for       File-MimeInfo (any)       ok: found v0.29
Checking for        IO-stringy (any)       ok: found v2.111
Checking for      Cache-Memcached (any)       ok: found v1.30
Checking for  File-Copy-Recursive (any)       ok: found v0.44
Checking for              mod_env (any)       ok
```

```
Checking for          mod_expires (any)       ok
Checking for          mod_headers (any)       ok
Checking for          mod_rewrite (any)       ok
Checking for          mod_version (any)       ok

Reading ./localconfig...
Checking for        DBD-mysql (v4.001)    ok: found v4.023
Checking for                MySQL (v5.0.15)   ok: found v5.5.60-MariaDB

WARNING: You need to set the max_allowed_packet parameter in your MySQL
configuration to at least 3276750. Currently it is set to 1048576.
You can set this parameter in the [mysqld] section of your MySQL

configuration file.

Adding new table bz_schema...
Initializing bz_schema...
Creating tables...
Converting attach_data maximum size to 100G...
Setting up choices for standard drop-down fields:
   priority bug_status rep_platform resolution bug_severity op_sys
Creating ./data directory...
Creating ./data/assets directory...
Creating ./data/attachments directory...
Creating ./data/db directory...
Creating ./data/extensions directory...
Creating ./data/mining directory...
Creating ./data/webdot directory...
Creating ./graphs directory...
Creating ./skins/custom directory...
Creating ./data/extensions/additional...
Creating ./data/mailer.testfile...
Creating ./Bugzilla/.htaccess...
Creating ./data/.htaccess...
Creating ./data/assets/.htaccess...
Creating ./data/attachments/.htaccess...
Creating ./data/webdot/.htaccess...
Creating ./graphs/.htaccess...
Creating ./lib/.htaccess...
Creating ./template/.htaccess...
Creating contrib/.htaccess...
Creating t/.htaccess...
```

```
Creating xt/.htaccess...
Precompiling templates...done.
Fixing file permissions...
Initializing “Dependency Tree Changes” email_setting ...
Initializing “Product/Component Changes” email_setting ...
Marking closed bug statuses as such...
Creating default classification ‘Unclassified’...
Setting up foreign keys...
Setting up the default status workflow...
Creating default groups...
Setting up user preferences...

Looks like we don’t have an administrator set up yet. Either this is
your first time using Bugzilla, or your administrator’s privileges
might have accidentally been deleted.

Enter the e-mail address of the administrator: mail@shakthimaan.com
Enter the real name of the administrator: Shakthi Kannan
Enter a password for the administrator account:
Please retype the password to verify:
mail@shakthimaan.com is now set up as an administrator.
Creating initial dummy product ‘TestProduct’...
```

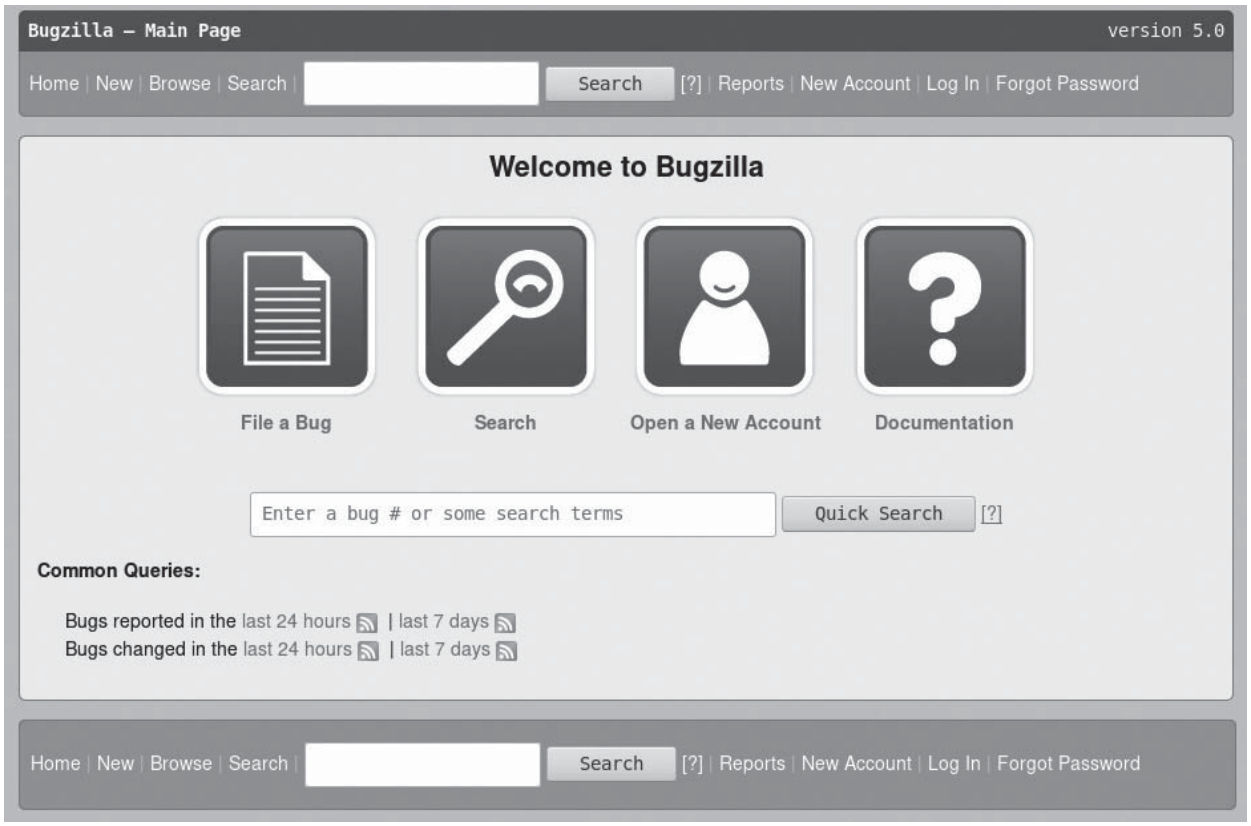


Figure 2: Bugzilla home page

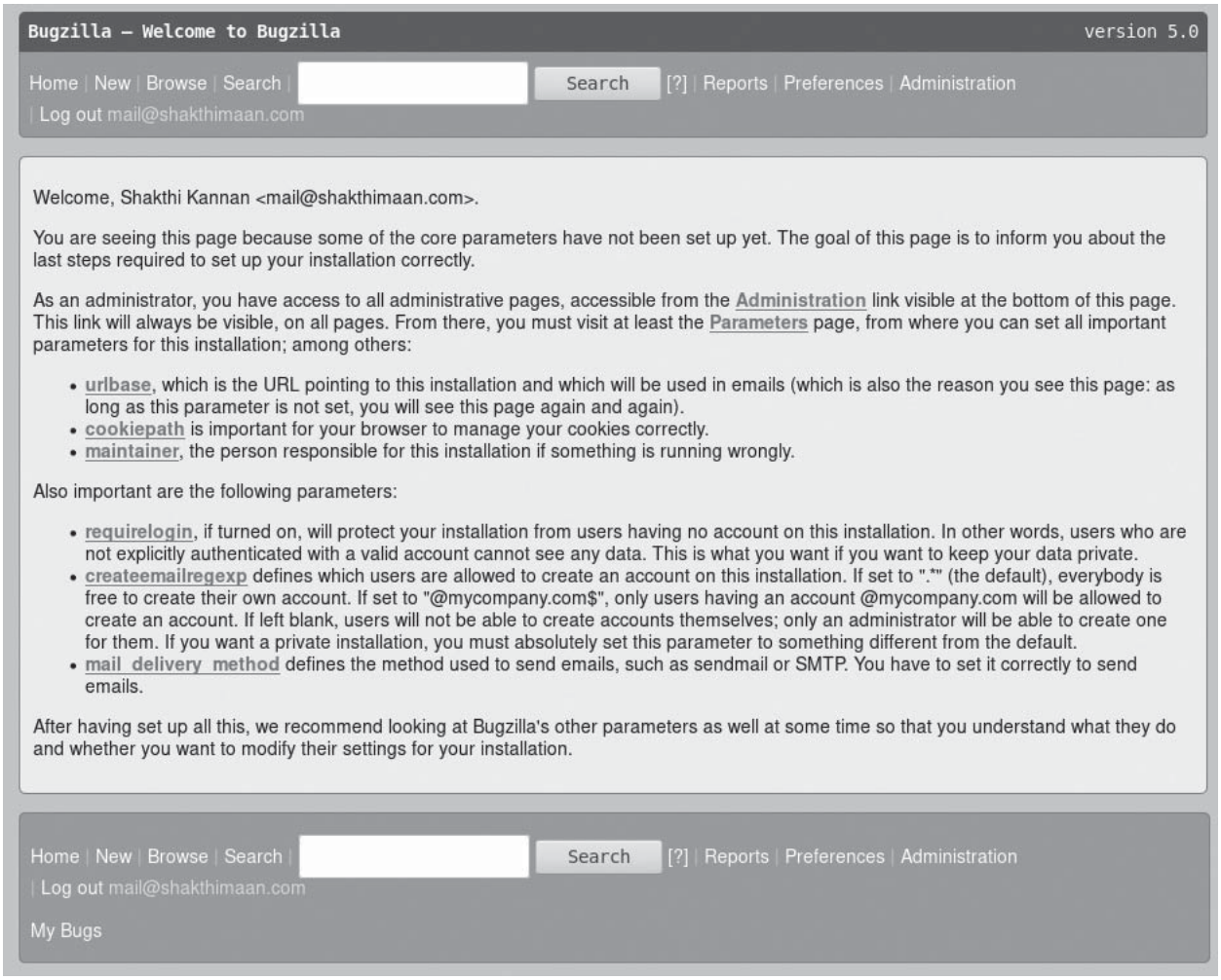


Figure 3: Bugzilla logged in default page

```
Now that you have installed Bugzilla, you should visit the 'Parameters'
page (linked in the footer of the Administrator account) to ensure it
is set up as you wish - this includes setting the 'urlbase' option to
the correct URL.
checksetup.pl complete.
```

You can now restart the httpd server using the following command:

```
$ sudo systemctl restart httpd
```

You can open the URL *http://192.168.122.87* on your host system to see the Bugzilla home page, as shown in Figure 2.

After logging in with the created user credentials, you should see the default page, as shown in Figure 3.

The Bugzilla documentation is available in different formats for your reference at *https://www.bugzilla.org/docs/*.

Chapter 22

Hardening of Parabola

Every computer offers security measures to isolate it from outside attacks. In this 21st article in the DevOps series, we will learn how to harden, automate and verify a Parabola GNU/Linux-libre system using Ansible.

Parabola GNU/Linux-libre is a free software GNU/Linux distribution based on packages from Arch Linux, but without the binary blobs. It respects the freedom of users and is endorsed by the Free Software Foundation. It is extremely lightweight and simple in design. The distribution follows a rolling release, but also provides installation media in the ISO format. The Pacman package manager is used, and the software packages are available for i686, x86_64 and armv7h architectures. The Linux-libre kernel is used instead of the generic Linux kernel. The official IRC channel is #*parabola* on *irc.freenode.net*, and the website of the project is *https://www.parabola.nu/*.

Setup

A Parabola GNU/Linux-libre (x86_64) 2018.06.02 ISO is used to set up a guest virtual machine (VM) with KVM/QEMU. The host system is also a Parabola GNU/Linux-libre x86_64 system, and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0, as is indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/guest/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible

  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory, playbook and configuration files are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
parabola ansible_host=192.168.122.128 ansible_connection=ssh ansible_
user=parabola ansible_password=password
```

The s*udo* package needs to be installed in the guest Parabola VM. A *parabola* user is created in the guest VM, and *sudo* access is provided for this user with the *visudo* command. You also need to ensure that the ssh daemon is started in the guest VM using the following command:

```
$ sudo systemctl start sshd
```

You should add an entry in the */etc/hosts* file on the host system for the Parabola guest VM as indicated below:

```
192.168.122.128 parabola
```

You can now test connectivity from Ansible to the Parabola guest VM using the following Ansible commands:

```
$ ansible -i inventory/kvm/inventory parabola -m ping
parabola | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

Passwords

The *pwgen* utility is useful to generate passwords. It has a number of options that you can use to generate strong passwords with a combination of numerals, symbols and characters. The Ansible playbook to install *pwgen* is given below for reference:

```
- name: Harden Parabola
  hosts: parabola
  become: yes
  become_method: sudo
```

```
  gather_facts: yes
  tags: [security]

  tasks:
    - name: Install pwgen
      package:
        name: pwgen
        state: latest
```

/etc file permissions

/etc/shadow and */etc/passwd* should have restricted file permissions. The */etc/shadow* file should be owned by the *root* user and should not be accessible by any other user. The */etc/passwd* file should similarly be owned by the *root* user and have restricted permissions. The following part of the Ansible playbook checks the required permissions of these two files:

```
- name: File permissions for /etc/shadow
  stat:
    path: /etc/shadow
  register: shadow

- assert:
    that:
      - "shadow.stat.pw_name == 'root'"
      - "shadow.stat.readable == true"
      - "shadow.stat.rgrp == false"
      - "shadow.stat.roth == false"
      - "shadow.stat.writeable == true"
      - "shadow.stat.wgrp == false"
      - "shadow.stat.woth == false"
      - "shadow.stat.xgrp == false"
      - "shadow.stat.xoth == false"
      - "shadow.stat.xusr == false"

- name: File permissions for /etc/passwd
  stat:
    path: /etc/passwd
  register: passwd


- assert:
```

```
    that:
     - "passwd.stat.pw_name == 'root'"
     - "passwd.stat.readable == true"
     - "passwd.stat.rgrp == true"
     - "passwd.stat.roth == true"
     - "passwd.stat.writeable == true"
     - "passwd.stat.wgrp == false"
     - "passwd.stat.woth == false"
     - "passwd.stat.xgrp == false"
     - "passwd.stat.xoth == false"
     - "passwd.stat.xusr == false"
```

Enforcing strong passwords

The password policy can be defined in the */etc/pam.d/passwd* file. For example, the following constraints are enforced when creating new passwords using the Ansible playbook given below:

- In case of error, prompt twice for password.
- The minimum length of the password should be ten characters.
- At least six characters should be different from the old password.
- There should be at least one digit.
- There should be at least one upper case character.
- There should be at least one other character.
- There should be at least one lower case character.

```
- name: Update /etc/pam.d/passwd
  lineinfile:
    path: /etc/pam.d/passwd
    insertbefore: '^password.*required.*pam_unix.so'
    line: 'password        required        pam_cracklib.so retry=2 minlen=10
difok=6 dcredit=-1 ucredit=-1 ocredit=-1 lcredit=-1'
```

Disk encryption

It is recommended that your disk be encrypted, and you use a strong passphrase that can be provided as input when booting the system. The Parabola wiki has useful documentation on disk encryption at *https://wiki.parabola.nu/Disk_encryption*. You can verify that the disk is encrypted using the following Ansible code snippet:

```
- name: Check disk is encrypted
  shell: lsblk /dev/sda | grep / | grep crypt
  register: encrypted
```

```
- assert:
    that:
      - “encrypted.rc == 0”
```

Mount options

The file systems that are mounted should have restrictive options, and use only what is required. The following commands check the *nosuid, nodev* and *noexec* options for the */tmp* and */dev/shm* directories:

```
- name: Check /tmp
  shell: mount | grep /tmp | grep -E ‘nosuid.*nodev’
  register: tmp

- assert:
    that:
      - “tmp.rc == 0”

- name: Check /dev/shm
  shell: mount | grep /dev/shm | grep -E ‘nosuid.*nodev.*noexec’
  register: tmp

- assert:
    that:
- “tmp.rc == 0”
```

File access permissions

Access to */boot* and */etc/iptables* should be restricted to only users with privileged access. The following playbook snippet updates the 0700 permission for both the */boot* and */etc/iptables* directories.

```
- name: Change file mode for /boot, /etc/iptables
  file:
    path: “{{ item }}”
    mode: 0700
  with_items:
    - /boot
    -/etc/iptables
```

Lock out user

You can lock a user account after a certain number of failed logins by updating the */etc/pam.d/system-login* file as shown below:

```
- name: Update /etc/pam.d/system-login
  lineinfile:
    path: /etc/pam.d/system-login
    regexp: '^auth.*required.*pam_tally.so'
    line: 'auth       required   pam_tally.so         deny=2 unlock_time=600
onerr=succeed file=/var/log/faillog'
```

The account can only be unlocked after a certain time (*unlock_time)* or by the *root* user.

Limit the processes

In order to prevent Denial of Service (DoS) attacks, it is a good practice to limit the number of processes that a user can run. This policy can be updated in the */etc/security/limits.conf* file as shown below:

```
- name: Update /etc/security/limits.conf
  lineinfile:
    path: /etc/security/limits.conf
    insertbefore: '# End of file'
    line: "{{ item }}"
  with_items:
    - '* soft nproc 100'
    - '* hard nproc 200'
```

Restrict root login

You can restrict root login by allowing only users who belong to the *wheel* group to log in to *root* using *su*. The same needs to be updated in the */etc/pam.d/su* and */etc/pam.d/su-l* files as shown below:

```
- name: Update /etc/pam.d/su
  lineinfile:
    path: /etc/pam.d/su
    regexp: '^#auth    required  pam_wheel.so use_uid'
    line: 'auth     required  pam_wheel.so use_uid'
```

```
- name: Update /etc/pam.d/su-l
  lineinfile:
    path: /etc/pam.d/su-l
    regexp: '^#auth    required  pam_wheel.so use_uid'
    line: 'auth     required  pam_wheel.so use_uid'
```

Restrict access to kernel log

The output of *dmesg* can provide useful information to attackers and

hence one needs to restrict access to it. The *ansible/files/* folder has both *50-dmesg-restrict.conf* and *50-kptr-restrict.conf* files, which need to be copied to *etc/sysctl.d* for the above functionality.

```
$ cat files/50-kptr-restrict.conf
kernel.kptr_restrict = 1
```

```
$ cat files/50-dmesg-restrict.conf
kernel.dmesg_restrict = 1
```

```
- name: Restrict access to kernel logs
  copy:
    src: ../../files/50-dmesg-restrict.conf
    dest: /etc/sysctl.d/50-dmesg-restrict.conf
    owner: root
    group: root
    mode: 0644
```

```
- name: Restrict access to kernel pointers in procfs
  copy:
```

```
    src: ../../files/50-kptr-restrict.conf
    dest: /etc/sysctl.d/50-kptr-restrict.conf
    owner: root
    group: root
    mode: 0644
```

Disable root login

The root SSH login should be disabled, by default. A remote user should only be able to log in to the system, and sudo if they have been given permissions. The same can be accomplished by setting *PermitRootlogin no* in */etc/ssh/sshd_config* as shown below:

```
- name: Disable root login
  lineinfile:
    path: /etc/ssh/sshd_config
    regexp: '^#PermitRootLogin'
    line: 'PermitRootLogin no'
```

The console logins for root can also be disabled by commenting them out in /etc/securetty file.

```
- name: Disable root console logins
  replace:
    path: /etc/securetty
    regexp: '^(?!#)(.*)'
    replace: '# \1'
```

Execution

The entire playbook is provided below for reference:

```
---
- name: Harden Parabola
  hosts: parabola
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [security]

  tasks:
    - name: Install pwgen
      package:
        name: pwgen
        state: latest

    - name: File permissions for /etc/shadow
      stat:
        path: /etc/shadow
      register: shadow

    - assert:
        that:
          - "shadow.stat.pw_name == 'root'"
          - "shadow.stat.readable == true"
          - "shadow.stat.rgrp == false"
          - "shadow.stat.roth == false"
          - "shadow.stat.writeable == true"
          - "shadow.stat.wgrp == false"
          - "shadow.stat.woth == false"
          - "shadow.stat.xgrp == false"
          - "shadow.stat.xoth == false"
          - "shadow.stat.xusr == false"

    - name: File permissions for /etc/passwd
      stat:
```

```
        path: /etc/passwd
      register: passwd

    - assert:
        that:
          - "passwd.stat.pw_name == 'root'"
          - "passwd.stat.readable == true"
          - "passwd.stat.rgrp == true"
          - "passwd.stat.roth == true"
          - "passwd.stat.writeable == true"
          - "passwd.stat.wgrp == false"
          - "passwd.stat.woth == false"
          - "passwd.stat.xgrp == false"
          - "passwd.stat.xoth == false"
          - "passwd.stat.xusr == false"
    - name: Update /etc/pam.d/passwd
      lineinfile:
        path: /etc/pam.d/passwd
        insertbefore: '^password.*required.*pam_unix.so'
        line: 'password        required        pam_cracklib.so retry=2
minlen=10 difok=6 dcredit=-1 ucredit=-1 ocredit=-1 lcredit=-1'

    - name: Check disk is encrypted
      shell: lsblk /dev/sda | grep / | grep crypt
      register: encrypted

    - assert:
        that:
          - "encrypted.rc == 0"

    - name: Check /tmp
      shell: mount | grep /tmp | grep -E 'nosuid.*nodev'
      register: tmp

    - assert:
        that:
          - "tmp.rc == 0"

    - name: Check /dev/shm
      shell: mount | grep /dev/shm | grep -E 'nosuid.*nodev.*noexec'
      register: tmp

    - assert:
```

```
        that:
          - "tmp.rc == 0"


    - name: Change file mode for /boot, /etc/iptables
      file:
        path: "{{ item }}"
        mode: 0700
      with_items:
        - /boot
        - /etc/iptables

    - name: Update /etc/pam.d/system-login
      lineinfile:
        path: /etc/pam.d/system-login
        regexp: '^auth.*required.*pam_tally.so'
        line: 'auth       required   pam_tally.so         deny=2 unlock_
time=600 onerr=succeed file=/var/log/faillog'

    - name: Update /etc/security/limits.conf
      lineinfile:
        path: /etc/security/limits.conf
        insertbefore: '# End of file'
        line: "{{ item }}"
      with_items:
        - '* soft nproc 100'
        - '* hard nproc 200'

    - name: Update /etc/pam.d/su
      lineinfile:
        path: /etc/pam.d/su
        regexp: '^#auth    required  pam_wheel.so use_uid'
        line: 'auth    required  pam_wheel.so use_uid'

    - name: Update /etc/pam.d/su-l
      lineinfile:
        path: /etc/pam.d/su-l
        regexp: '^#auth required  pam_wheel.so use_uid'
        line: 'auth    required  pam_wheel.so use_uid'

    - name: Restrict access to kernel logs
      copy:
```

```
        src: ../../files/50-dmesg-restrict.conf
        dest: /etc/sysctl.d/50-dmesg-restrict.conf
        owner: root
        group: root
        mode: 0644

    - name: Restrict access to kernel pointers in procfs
      copy:

        src: ../../files/50-kptr-restrict.conf
        dest: /etc/sysctl.d/50-kptr-restrict.conf
        owner: root
        group: root
        mode: 0644

    - name: Disable root login
      lineinfile:
        path: /etc/ssh/sshd_config
        regexp: '^#PermitRootLogin'
        line: 'PermitRootLogin no'

    - name: Disable root console logins
      replace:
        path: /etc/securetty
        regexp: '^(?!#)(.*)'
        replace: '# \1'
```

You can invoke the above playbook using:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/parabola.
yml -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four *v*'s for a more detailed output. The *-K* option prompts for the *sudo password* for the guest Parabola user account.

You are encouraged to read the security guide and best practices for Parabola GNU/Linux-libre available at *https://wiki.parabola.nu/Security* for more information.

Chapter 23

Using Ansible to Build GNU Guile

This is the 22nd article in the DevOps series and here we will use Ansible to automate the process of building GNU Guile from its source code. GNU 'Guile' is an acronym for GNU Ubiquitous Intelligent Language for Extensions, which is an implementation of the Scheme programming language and an extension language for the GNU Project.

GNU Guile was designed by Aubrey Jaffer, Tom Lord and Miles Bader and was first made available in 1993. You can write desktop, Web and command line applications in GNU Guile. The latest stable version, as of July 2018, is v 2.2.4. It is released under the GPL and is available for both IA-32 and x86-64 platforms.

Setup

A Parabola GNU/Linux-libre (x86_64) 2018.06.02 ISO is used to set up a guest virtual machine (VM) using KVM/QEMU. The host system is also a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0 as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/guest/.ansible/plugins/modules', '/
usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory and playbook files are created on the host system, as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
```

The *inventory/kvm/inventory* file contains the following:

```
guile ansible_host=192.168.122.109 ansible_connection=ssh ansible_user=parabola
ansible_password=password
```

The 'sudo' package needs to be installed in the guest Parabola VM. A Parabola user is created in the guest VM and sudo access is provided for this user using the 'visudo' command. Python and Openssh software are also required to be installed on the guest VM. The ssh daemon is started in the guest VM using the following command:

```
$ sudo systemctl start sshd.service
```

You should add an entry in the */etc/hosts* file on the host system for the Parabola guest VM, as indicated below:

```
192.168.122.109 guile
```

You can now test the connectivity from Ansible to the Parabola guest VM using the following Ansible command:

```
$  ansible -i inventory/kvm/inventory guile -m ping
guile | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Software dependencies

GNU Guile requires a number of software dependencies, such as *gmp, libunistring, libffi* and *readline*. The essential build tools—*autoconf, automake, libtool, flex, gcc, gc* and *make* — are also needed for building GNU Guile. The Ansible playbook to install the above software is given below:

```
---
- name: Repository setup
  hosts: guile
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [packages]

  tasks:
```

```
    - name: Install dependencies
      package:
        name: "{{ item }}"
        state: latest
      with_items:
        - "git"
        - "gmp"
        - "libunistring"
        - "libffi"
        - "readline"
        - "autoconf"
        - "libtool"
        - "automake"
        - "flex"
        - "pkgconf"
        - "gcc"
        - "gc"
- "make"
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags packages -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four 'v' s for a more detailed output. The *-K* option prompts for the sudo password for the guest Parabola user account.

GNU Guile

The master branch from the GNU Guile Git repository (*git.sv.gnu.org/guile.git*) will be used as the source for the build process. The source code is cloned to the */home/parabola/guile* directory. The basic GNU Autotools procedure of running *autogen.sh*, *configure* and *make* in the source code will compile GNU Guile. The build will take a few hours to complete, even on the latest hardware. The Ansible playbook to compile GNU Guile is as follows:

```
- name: Build guile
  hosts: guile
  tags: [guile]

  vars:
    guile_home: "/home/parabola/guile"
```

```
  tasks:
    - name: Clone Guile from git
      git:
        repo: 'git://git.sv.gnu.org/guile.git'
        dest: "{{ guile_home }}"

    - name: Run autogen.sh
      command: ./autogen.sh
      args:
        chdir: "{{ guile_home }}"

    - name: Run configure
      command: ./configure
      args:
        chdir: "{{ guile_home }}"

    - name: Run make
      command: make
      args:
        chdir: "{{ guile_home }}"
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags guile -vv -K
```

Installation

The built executables, libraries and header files can be installed on the guest VM. By default, the *make install* command will install the artifacts to */usr/local*. The Ansible playbook for installation is given below for reference:

```
- name: Install guile
  hosts: guile
  become: yes
  become_method: sudo
  tags: [install]

  vars:
    guile_home: "/home/parabola/guile"

  tasks:
    - name: Install Guile
```

```
      command: make install
      args:
        chdir: "{{ guile_home }}"
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags install -vv -K
```

Tests

The GNU Guile source code comes with a comprehensive test suite that you can invoke using *make check* to run tests against the built binary. The following Ansible playbook will run the GNU Guile tests:

```
- name: Test guile
  hosts: guile
  tags: [test]

  vars:
    guile_home: "/home/parabola/guile"

  tasks:
    - name: Run Guile tests
      command: make check
      args:
        chdir: "{{ guile_home }}"
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags test -vv -K
```

Documentation

TeX is required to generate PDFs from the built-in documentation sources in the GNU Guile code repository. The following Ansible playbook will generate the PDF documentation:

```
- name: Generate documentation
  hosts: guile
  become: yes
  become_method: sudo
  gather_facts: yes
```

```
  tags: [docs]

  vars:
    guile_home: “/home/parabola/guile”

  tasks:
    - name: Install texlive-core
      package:
        name: texlive-core
        state: latest

    - name: Generate documentation
      command: make pdf
      args:
        chdir: “{{ guile_home }}”
```

A sample execution output of invoking the above playbook is given below:

```
$  ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags docs -K
SUDO password:

PLAY [Repository setup] ********************************
TASK [Gathering Facts] *********************************
ok: [guile]
PLAY [Build guile] *************************************
TASK [Gathering Facts] *********************************
ok: [guile]
PLAY [Install guile] ***********************************
TASK [Gathering Facts] *********************************
ok: [guile]
PLAY [Test guile] **************************************
TASK [Gathering Facts] *********************************
ok: [guile]
PLAY [Generate documentation] **************************
TASK [Gathering Facts] *********************************
ok: [guile]
TASK [Install texlive-core] ****************************
changed: [guile]
TASK [Generate documentation] **************************
changed: [guile]
```

```
PLAY [Run benchmarks] ************************************
TASK [Gathering Facts] ************************************
ok: [guile]
PLAY RECAP ***********************************************
guile                      : ok=8    changed=2    unreachable=0    failed=0
```

The following PDFs are built and available in the GNU Guile sources directory:

```
$ find . -name *.pdf
./doc/ref/scheme.pdf
./doc/ref/gds.pdf
./doc/ref/guile.pdf
./doc/ref/hierarchy.pdf
./doc/r5rs/r5rs.pdf
```

Benchmarking

A *benchmark-guile* script is available to run benchmarks against the built binaries. The following Ansible playbook can be used to run benchmarking:

```
- name: Run benchmarks
  hosts: guile
  gather_facts: yes
  tags: [benchmarks]

  vars:
    guile_home: "/home/parabola/guile"

  tasks:
    - name: Run benchmarks
      command: ./benchmark-guile
      args:
        chdir: "{{ guile_home }}"
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/guile.yml
--tags benchmarks -vv -K
```

You can learn more about GNU Guile from its home page at *https://www.gnu.org/software/guile/*.

Chapter 24

Configuring a PostgreSQL Master-Slave Setup Using Ansible

PostgreSQL is a free and open source, ACID-compliant, transactional database written in the C programming language. It supports updatable views, triggers, stored procedures and foreign keys, and manages concurrency using multi-version concurrency control (MVCC). In this 23rd article in the DevOps series, we will learn how to install and configure a PostgreSQL master-slave replication setup.

PostgreSQL has a number of interfaces for various programming languages and libraries that are available to interact with the database. The primary command-line tool to interact with the database is *psql*. GUI administrative tools are also available. It was first released in 1996 under the PostgreSQL licence.

CentOS 7 (x86_64) is used as the base OS for the PostgreSQL database server. A couple of guest CentOS VMs are launched using KVM. One instance is used as the database master, while the other is used as a replication slave. The *centos* users in both the VMs are given sudo access using the *visudo* command. SELinux is disabled for the exercise.

The host system is a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0 as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [‘/home/guest/.ansible/plugins/modules’, ‘/
usr/share/ansible/plugins/modules’]
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
  executable location = /usr/bin/ansible
```

```
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory and playbook file are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
```

The *inventory/kvm/inventory* file contains the following code:

```
[pgmaster]
host1 ansible_host=192.168.122.174 ansible_connection=ssh ansible_user=centos
ansible_password=centos123

[pgslave]
host2 ansible_host=192.168.122.113 ansible_connection=ssh ansible_user=centos
ansible_password=centos123

[all:children]
pgmaster
pgslave
```

The *host1* and *host2* entries are added in the */etc/hosts* file on the host system as shown below:

```
192.168.122.174 host1
192.168.122.113 host2
```

You can test connectivity from Ansible to the CentOS guest VMs using the following Ansible commands:

```
$ ansible -i inventory/kvm/inventory pgmaster -m ping
host1 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}

$ ansible -i inventory/kvm/inventory pgslave -m ping
host2 | SUCCESS => {
    “changed”: false,
    “ping”: “pong”
}
```

The Common setup

The PostgreSQL server needs to be installed on both the master and slave instances, and the database needs to be initialised on both. The Postgres user password is changed. Although the password is listed as a variable in the playbook, it can be encrypted and stored using Ansible Vault when used in production. The *firewalld* daemon is started and port 5432 for the database is allowed through the firewall. The playbook to install and configure the base PostgreSQL server is given below:

```
---
- name: Common pre-requisites
  hosts: all
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [common]

  vars:
    db_password: “postgres123”

  tasks:
    - name: Install pgdg-centos96 RPM
      package:
        name: https://yum.postgresql.org/9.6/redhat/rhel-7-x86_64/pgdg-
centos96-9.6-3.noarch.rpm
        state: present

    - name: Install PostgreSQL RPM
      package:
        name: “{{ item }}”
        state: latest
      with_items:
        - “postgresql96-server”
        - “postgresql96-contrib”
        - “nano”

    - name: Initialize database
      shell: sudo ./postgresql96-setup initdb
      args:
        chdir: /usr/pgsql-9.6/bin/

    - name: Start PostgreSQL server
```

```
      systemd:
        name: postgresql-9.6
        enabled: yes
        state: started

    - name: Wait for server to start
      wait_for:
        port: 5432

    - name: Change postgres password
      shell: sudo -u postgres psql -c "ALTER USER postgres WITH password '{{
db_password }}'"

    - name: Start and enable firewalld
      systemd:
        name: firewalld
        enabled: yes
        state: started

    - name: Allow postgresql through firewall
      firewalld:
        service: postgresql
        permanent: yes
        state: enabled

    - name: Reload firewalld
      shell: firewall-cmd --reload

    - name: List reloaded firewall
      shell: firewall-cmd --list-all
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/
postgresql.yml --tags common -vv --K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four '*v*' s for a more detailed output. The *-K* option prompts for the *sudo* password for the guest CentOS user account.

The PostgreSQL master

The PostgreSQL configuration files need to be updated on the master node with various settings required for replication. The database server is

restarted and a replication user is also created. The *replica_password* can be encrypted and stored using Ansible Vault, when used in production. The playbook to set up the PostgreSQL master instance is as follows:

```
- name: Setup pgmaster
  hosts: pgmaster
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [pgmaster]

  vars:
    replica_password: 'replica123'

  tasks:
    - name: Update /var/lib/pgsql/9.6/data/postgresql.conf
      lineinfile:
        path: /var/lib/pgsql/9.6/data/postgresql.conf
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
      with_items:
        - { regexp: "#listen_addresses = 'localhost'", line: "listen_addresses
= '{{ ansible_default_ipv4.address }}'" }
        - { regexp: '#wal_level = minimal', line: 'wal_level = hot_standby' }
        - { regexp: '#synchronous_commit = on', line: 'synchronous_commit =
local' }
        - { regexp: '#archive_mode = off', line: 'archive_mode = on' }
        - { regexp: "#archive_command = ''", line: "archive_command = 'cp %p /
var/lib/pgsql/9.6/archive/%f'" }
        - { regexp: '#max_wal_senders = 0', line: 'max_wal_senders = 2' }
        - { regexp: '#wal_keep_segments = 0', line: 'wal_keep_segments = 2' }
        - { regexp: "#synchronous_standby_names = ''", line: "synchronous_
standby_names = 'slave01'" }

    - name: Create archive directory
      file:
        path: /var/lib/pgsql/9.6/archive
        mode: 0700
        owner: postgres
        group: postgres
        state: directory

    - name: Update pg_hba.conf
```

```
      blockinfile:
        path: /var/lib/pgsql/9.6/data/pg_hba.conf
        insertafter: '#host    replication     postgres        ::1/128
ident'
        block: |
          # Localhost
          host    replication     replica          127.0.0.1/32            md5

          # PostgreSQL Master IP address
          host    replication     replica
{{ ansible_default_ipv4.address}}/32            md5

          # PostgreSQL Slave IP address
          host    replication     replica
{{ hostvars['host2'].ansible_default_ipv4.address }}/32            md5

    - name: Restart PostgreSQL server
      systemd:
        name: postgresql-9.6
        enabled: yes
        state: restarted

    - name: Create replication user
      shell: sudo -u postgres psql -c "CREATE USER replica REPLICATION LOGIN
ENCRYPTED PASSWORD '{{ replica_password }}'"
```

The above playbook invocation and sample output is shown below, for reference:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/
postgresql.yml --tags pgmaster -K
SUDO password:

PLAY [Common pre-requisites] ******************************

TASK [Gathering Facts] *************************************
ok: [host2]
ok: [host1]

PLAY [Setup pgmaster] **************************************

TASK [Gathering Facts] *************************************
```

```
ok: [host1]

TASK [Update /var/lib/pgsql/9.6/data/postgresql.conf] *******

changed: [host1] => (item={'regexp': "#listen_addresses = 'localhost'", 'line':
"listen_addresses = '192.168.122.174'"})
changed: [host1] => (item={'regexp': '#wal_level = minimal', 'line': 'wal_level
= hot_standby'})
changed: [host1] => (item={'regexp': '#synchronous_commit = on', 'line':
'synchronous_commit = local'})
changed: [host1] => (item={'regexp': '#archive_mode = off', 'line': 'archive_
mode = on'})
changed: [host1] => (item={'regexp': "#archive_command = ''", 'line': "archive_
command = 'cp %p /var/lib/pgsql/9.6/archive/%f'"})
changed: [host1] => (item={'regexp': '#max_wal_senders = 0', 'line': 'max_wal_
senders = 2'})
changed: [host1] => (item={'regexp': '#wal_keep_segments = 0', 'line': 'wal_
keep_segments = 2'})
changed: [host1] => (item={'regexp': "#synchronous_standby_names = ''", 'line':
"synchronous_standby_names = 'slave01'"})

TASK [Create archive directory] ****************************
changed: [host1]

TASK [Update pg_hba.conf] **********************************
changed: [host1]

TASK [Restart PostgreSQL server] ***************************
changed: [host1]

TASK [Create replication user] *****************************

 [WARNING]: Consider using 'become', 'become_method'  'become_user' rather than
running sudo

changed: [host1]

PLAY [Setup pgslave] ***************************************

TASK [Gathering Facts] *************************************
ok: [host2]
```

```
PLAY RECAP *********************************************************

host1                      : ok=7    changed=5    unreachable=0    failed=0
host2                      : ok=2    changed=0    unreachable=0    failed=0
```

The PostgreSQL slave

The last step is to configure the PostgreSQL slave instance to receive data from the master instance. The PostgreSQL server is initially stopped, the *var/lib/pgsql/9.6/data* directory is backed up, and a new *data/* directory is created. The initial data is fetched from the master using *pg_basebackup*. The replica user password can be encrypted and stored using Ansible Vault. The *postgresql.conf* file is updated, a *recovery.conf* file is created, and the PostgreSQL server on the slave instance is started. The Ansible playbook to configure the replication slave is given below:

```
- name: Setup pgslave
  hosts: pgslave
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [pgslave]

  vars:
    replica_password: 'replica123'

  tasks:
    - name: Stop PostgreSQL server
      systemd:
        name: postgresql-9.6
        state: stopped

    - name: Move data backup
      shell: mv data data-backup
      args:
        chdir: /var/lib/pgsql/9.6/

    - name: Create data directory
      file:
        path: /var/lib/pgsql/9.6/data
        mode: 0700
        owner: postgres
        group: postgres
```

```
        state: directory

    - name: Backup initial data from master
      shell: su - postgres -c "PGPASSWORD={{ replica_password }} pg_basebackup
-w -h {{ hostvars['host1'].ansible_default_ipv4.address }} -U replica -D /var/
lib/pgsql/9.6/data -P --xlog"

    - name: Update /var/lib/pgsql/9.6/data/postgresql.conf
      lineinfile:
        path: /var/lib/pgsql/9.6/data/postgresql.conf
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
      with_items:
        - { regexp: "#listen_addresses = 'localhost'", line: "listen_addresses
= '{{ ansible_default_ipv4.address }}'" }
        - { regexp: "#hot_standby = off", line: "hot_standby = on" }

    - name: Create recovery.conf
      blockinfile:
        path: /var/lib/pgsql/9.6/data/recovery.conf
        block: |
          standby_mode = 'on'
          primary_conninfo = 'host={{ hostvars['host1'].ansible_default_ipv4.
address }} port=5432 user=replica password={{ replica_password }} application_
name=slave01'
          trigger_file = '/tmp/postgresql.trigger.5432'
        mode: 0600
        owner: postgres
        group: postgres
        state: present
        create: yes

    - name: Start PostgreSQL server
      systemd:
        name: postgresql-9.6
        state: started
```

The above playbook can be executed using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/
postgresql.yml --tags pgslave -K
```

Testing

You can log in to the master instance, become a Postgres user, and run the following *psql* commands to see the synchronisation state of the setup:

```
# su - postgres
-bash-4.2$ psql -c “select application_name, state, sync_priority, sync_state
from pg_stat_replication;”
 application_name |   state   | sync_priority | sync_state
------------------+-----------+---------------+------------
 slave01          | streaming |             1 | sync
(1 row)

-bash-4.2$ psql -x -c “select * from pg_stat_replication;”
-[ RECORD 1 ]----+---------------------------------
pid              | 4054
usesysid         | 16384
usename          | replica
application_name | slave01
client_addr      | 192.168.122.113
client_hostname  |
client_port      | 53720
backend_start    | 2018-12-12 21:32:07.382766+05:30
backend_xmin     |
state            | streaming
sent_location    | 0/3000060
write_location   | 0/3000060
flush_location   | 0/3000060
replay_location  | 0/3000060
sync_priority    | 1
sync_state       | sync
```

You can create tables and insert records on the master instance. An example is given below:

```
-bash-4.2$ psql
psql (9.6.11)
Type “help” for help.

postgres=# CREATE TABLE student (name VARCHAR(100));
CREATE TABLE
postgres=# INSERT INTO student VALUES (‘Adith’);
INSERT 0 1
```

```
postgres=# INSERT INTO student VALUES (‘Shakthi’);
INSERT 0 1
postgres=#
```

You can now verify that the records exist on the slave instance using the following set of commands:

```
[root@host1 ~]# su - postgres
Last login: Wed Dec 12 21:32:02 IST 2018
-bash-4.2$ psql
psql (9.6.11)
Type “help” for help.

postgres=# select * from student;
  name
---------
 Adith
 Shakthi
(2 rows)
```

If you try to insert records on the slave instance, the database will throw a ‘read-only transaction’ error as shown below:

```
postgres=# INSERT INTO student VALUES (‘Foo’);
ERROR:  cannot execute INSERT in a read-only transaction
postgres=#
```

You are encouraged to read Chapter 26 of the book ‘High Availability, Load Balancing Replication’ from the PostgreSQL documentation page at *https://www.postgresql.org/docs/current/high-availability.html*.

Chapter 25

Using Ansible to Set Up HAProxy as a Load Balancer for Nginx

In this 24th article in the DevOps series, we will learn how to set up HAProxy as a load balancer for multiple Nginx Web servers using Ansible.

HAProxy is free, open source, highly available, load balancer software written by Willy Tarreau in 2000. It is implemented in the C programming language. It is known for its high performance and is extremely reliable and secure. It supports both Layer 4 (TCP) and Layer 7 (HTTP) based application load balancing, and is released under the GPLv2 licence. Nginx is a Web server created by Igor Sysoev, and is also written in the C programming language. It can be used as a reverse proxy, mail proxy and as an HTTP cache. It was first released in 2004 and uses the 2-clause BSD licence.

Setup

The HAProxy and Nginx installations use CentOS 7 (x86_64) as their base operating system. A single instance is launched using KVM for running HAProxy. A couple of CentOS VMs are provisioned to install and configure Nginx. The *centos* users in all the VMs are given sudo access using the *visudo* command. SELinux is disabled for the exercise.

The host system is a Parabola GNU/Linux-libre x86_64 system and Ansible is installed using the distribution package manager. The version of Ansible used is 2.6.0 as indicated below:

```
$ ansible --version
ansible 2.6.0
  config file = /etc/ansible/ansible.cfg
  configured module search path = [‘/home/guest/.ansible/plugins/modules’, ‘/
usr/share/ansible/plugins/modules’]
  ansible python module location = /usr/lib/python3.6/site-packages/ansible
```

```
  executable location = /usr/bin/ansible
  python version = 3.6.5 (default, May 11 2018, 04:00:52) [GCC 8.1.0]
```

The Ansible inventory, files and playbook directories are created on the host system as follows:

```
ansible/inventory/kvm/
       /playbooks/configuration/
       /files/
```

The *inventory/kvm/inventory* file contains the following:

```
[front]
haproxy ansible_host=192.168.122.113 ansible_connection=ssh ansible_user=centos
ansible_password=password

[web1]
nginx1 ansible_host=192.168.122.248 ansible_connection=ssh ansible_user=centos
ansible_password=password

[web2]
nginx2 ansible_host=192.168.122.147 ansible_connection=ssh ansible_user=centos
ansible_password=password

[web:children]
web1
web2
```

The 'front' group contains the HAProxy instance information. The couple of Nginx Web servers are grouped together under a 'web' group, and can also be accessed individually. You can test connectivity from Ansible to the CentOS guest VMs using the following Ansible commands:

```
$ ansible -i inventory/kvm/inventory haproxy -m ping
haproxy | SUCCESS => {
    "changed": false,
    "ping": "pong"
}

$ ansible -i inventory/kvm/inventory nginx1 -m ping
nginx1 | SUCCESS => {
    "changed": false,
```

```
    "ping": "pong"
}
$ ansible -i inventory/kvm/inventory nginx2 -m ping
nginx2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}

$ ansible -i inventory/kvm/inventory web -m ping
nginx1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
nginx2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

Nginx

We will first set up the Nginx Web servers. The EPEL release RPM is installed and the HAProxy server IP address is added to */etc/hosts* file on the instances. The YUM package manager is used to install Nginx, and then Port 80 is allowed through the firewall. We then start the Nginx Web server and wait for the server to listen on port 80. The Ansible playbook to install and set up Nginx is as follows:

```
---
- name: Setup Nginx web server
  hosts: web
  become: yes
  become_method: sudo
  gather_facts: yes
  tags: [web]

  tasks:
    - name: Install EPEL Release
      package:
        name: epel-release
        state: present

    - name: Add HAProxy server to /etc/hosts
      lineinfile:
```

```
        path: /etc/hosts
        line: "{{ hostvars['haproxy'].ansible_host }} haproxy"
        state: present

    - name: Install Nginx
      package:
        name: nginx
        state: present

    - name: Allow port 80
      shell: iptables -I INPUT -p tcp --dport 80 -m state --state
NEW,ESTABLISHED -j ACCEPT

    - name: Start Nginx server
      systemd:
        name: nginx
        enabled: yes
        state: started

    - name: Wait for server to start
      wait_for:
        port: 80
```

The above playbook can be invoked using the following command:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/frontend.
yml --tags web -vv -K
```

The *-vv* represents the verbosity in the Ansible output. You can use up to four '*v*' s for a more detailed output. The *-K* option prompts for the sudo password for the guest CentOS user account.

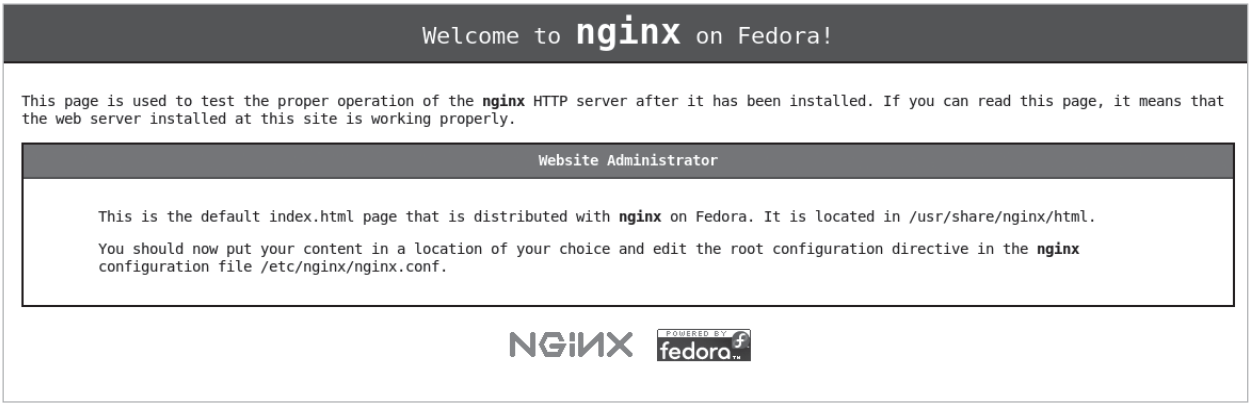


Figure 1: Default Nginx home page

You can now open the Nginx Web server URLs (*http://192.168.122.248* and *http://192.168.122.147*) in a browser to see the default Nginx home page as shown in Figure 1.

HAProxy

The YUM package repository needs to be updated before proceeding to install HAProxy. The Nginx server IP addresses and hostnames are added to the */etc/hosts* file on the instances. The default */etc/haproxy/haproxy.cfg* directory is backed up and a new *haproxy.cfg* file is created, whose file contents are shown below:

```
global
    log                 127.0.0.1 local2

    chroot           /var/lib/haproxy
    pidfile          /var/run/haproxy.pid
    maxconn      4000
    user              haproxy
    group       haproxy
    daemon
     stats socket /var/lib/haproxy/stats

defaults
    mode                    http
    log                     global
    option                  httplog
    option                  dontlognull
    option http-server-close
    option forwardfor       except 127.0.0.0/8
    option                  redispatch
    retries                 3
    timeout http-request    10s
    timeout queue           1m
    timeout connect         10s
    timeout client          1m
    timeout server          1m
    timeout http-keep-alive 10s
    timeout check           10s
    maxconn                 3000

listen haproxy-monitoring *:8080
    mode http
    option forwardfor
```

```
    option httpclose
    stats enable
    stats show-legends
    stats refresh 5s
    stats uri /stats
    stats realm Haproxy\ Statistics
    stats auth admin:password
    stats admin if TRUE
    default_backend app-main

frontend main
    bind *:80
    option http-server-close
    option forwardfor
    default_backend app-main

backend app-main
    balance roundrobin
    option httpchk HEAD / HTTP/1.1\r\nHost:\ localhost
{% for item in groups['web'] %}
    server {{ hostvars[item]['inventory_hostname'] }} {{ hostvars[item]
['ansible_default_ipv4']['address'] }}:80 check
{% endfor %}
```

The rsyslog software will be used to collect the HAProxy logs. The */etc/rsyslog.conf* configuration is updated to load the UDP Syslog Input Module (imudp) and to run a UDP server on Port 514. A */etc/rsyslog.d/haproxy.conf* configuration file is created, the contents of which are as follows:

```
local2.=info     /var/log/haproxy-access.log
local2.notice    /var/log/haproxy-info.log
```

The firewall is updated to allow Port 8080, and the rsyslog server is restarted. The final step is to start the HAProxy server and wait for it to listen on Port 8080. The complete Ansible playbook to install and configure HAProxy is given below:

```
- name: Setup HAProxy
  hosts: front
  become: yes
  become_method: sudo
```

```
  gather_facts: yes
  tags: [haproxy]

  tasks:
    - name: Yum update
      yum: name=* update_cache=yes state=present

    - name: Install HAProxy
      package:
        name: haproxy
        state: present

    - name: Add Nginx servers to /etc/hosts
      lineinfile:
        path: /etc/hosts
        line: "{{ hostvars[item]['ansible_default_ipv4']['address'] }} {{
hostvars[item]['inventory_hostname'] }}"
        state: present
      with_items: "{{ groups['web'] }}"

    - name: Backup default haproxy.cfg
      command: mv haproxy.cfg haproxy.cfg.orig
      args:
        chdir: /etc/haproxy

    - name: Create new haproxy.cfg
      template:
        src: ../../files/haproxy.cfg.j2
        dest: /etc/haproxy/haproxy.cfg
        mode: 0644

    - name: Update /etc/rsyslog.conf
      lineinfile:
        path: /etc/rsyslog.conf
        regexp: "{{ item.regexp }}"
        line: "{{ item.line }}"
      with_items:
        - { regexp: '#\$ModLoad imudp', line: '$ModLoad imudp' }
        - { regexp: '#\$UDPServerRun 514', line: '$UDPServerRun 514' }

    - name: Create /etc/rsyslog.d/haproxy.conf
      copy:
```

```
        src: ../../files/haproxy.conf
        dest: /etc/rsyslog.d/haproxy.conf
        mode: 0644

    - name: Allow port 8080
      shell: iptables -I INPUT -p tcp --dport 8080 -m state --state
NEW,ESTABLISHED -j ACCEPT

    - name: Restart rsyslog
      systemd:
        name: rsyslog
        state: restarted

    - name: Start HAProxy server
      systemd:
        name: haproxy
        enabled: yes
        state: started

    - name: Wait for server to start
      wait_for:
        port: 8080
```

A sample execution of the above playbook is as follows:

```
$ ansible-playbook -i inventory/kvm/inventory playbooks/configuration/frontend.
yml --tags haproxy -K
SUDO password:

PLAY [Setup Nginx web server] ****************************
TASK [Gathering Facts] ***********************************
ok: [nginx1]
ok: [nginx2]
PLAY [Setup HAProxy] *************************************
TASK [Gathering Facts] ***********************************
ok: [haproxy]
TASK [Yum update] ****************************************
ok: [haproxy]
TASK [Install HAProxy] ***********************************
changed: [haproxy]
TASK [Add Nginx servers to /etc/hosts] *********************
changed: [haproxy] => (item=nginx2)
```

```
changed: [haproxy] => (item=nginx1)
TASK [Backup default haproxy.cfg] ***************************
changed: [haproxy]
TASK [Create new haproxy.cfg] *******************************
changed: [haproxy]
TASK [Update /etc/rsyslog.conf] *****************************
changed: [haproxy] => (item={'regexp': '#\\$ModLoad imudp', 'line': '$ModLoad
imudp'})
changed: [haproxy] => (item={'regexp': '#\\$UDPServerRun 514', 'line':
'$UDPServerRun 514'})

TASK [Create /etc/rsyslog.d/haproxy.conf] ********************
changed: [haproxy]
TASK [Allow port 8080] **************************************
changed: [haproxy]
TASK [Restart rsyslog] **************************************
```

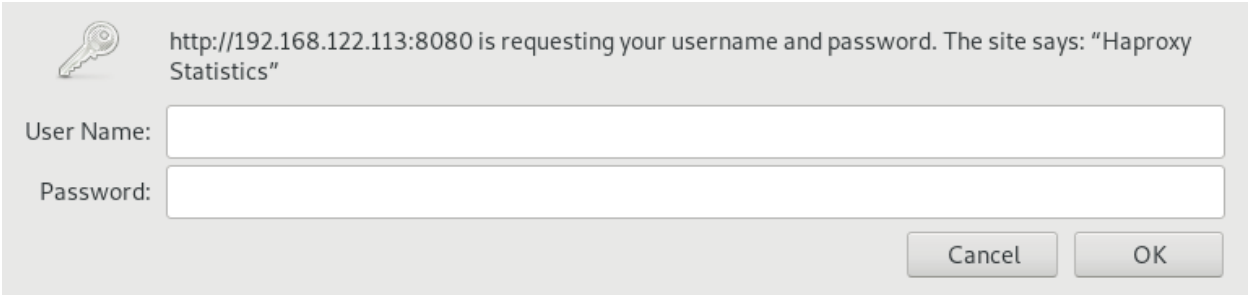


Figure 2: HAProxy Web login

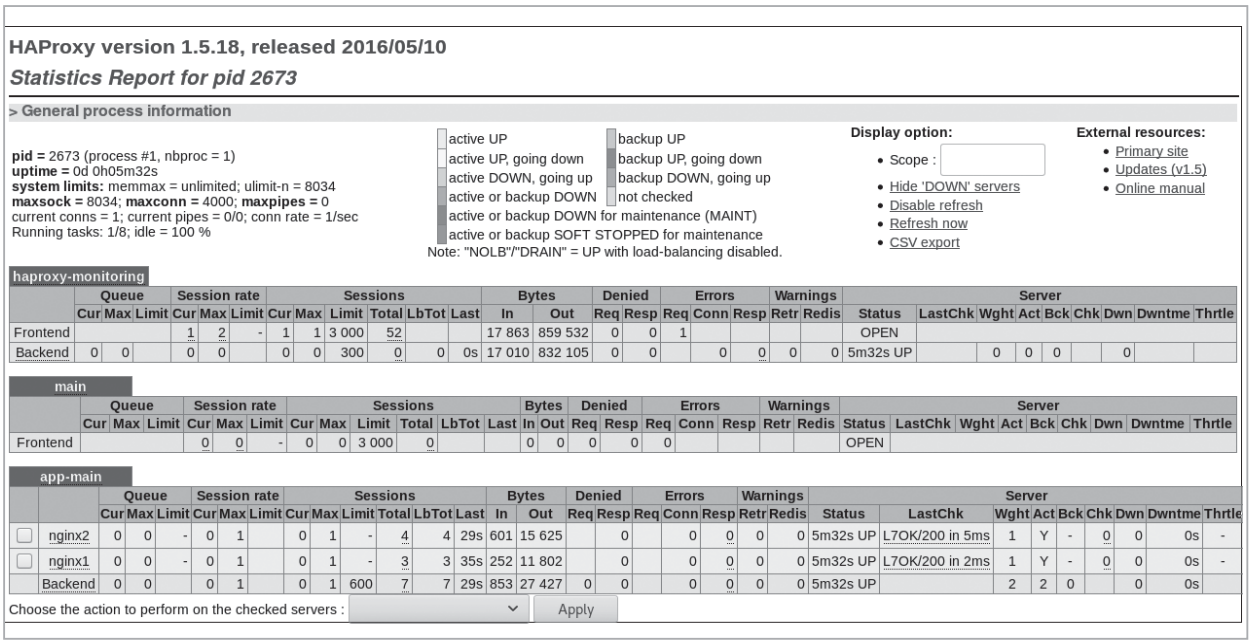


Figure 3: HAProxy stats

```
changed: [haproxy]
TASK [Start HAProxy server] ********************************
changed: [haproxy]

TASK [Wait for server to start] ****************************
ok: [haproxy]

PLAY RECAP *************************************************
haproxy     : ok=12   changed=9    unreachable=0    failed=0
nginx1      : ok=1    changed=0    unreachable=0    failed=0
nginx2      : ok=1    changed=0    unreachable=0    failed=0
```

Testing

You can open the HAProxy Web page using *http://192.168.122.113:8080/stats* and you will be prompted to log in as shown in Figure 2.

You can use the credentials (*admin:password*) as specified in */etc/haproxy/haproxy.cfg* to log in, and you will see the HAProxy stats page as shown in Figure 3.

You can make multiple requests to the HAProxy front-end server in the browser or using *curl http://192.168.122.113:8080* from the command line. You will observe that the requests are being sent to both the Nginx Web servers in a round-robin fashion, which can be seen in the app-main section of the HAProxy stats page.

About the Authors

Shakthi Kannan
The author is a free software enthusiast and blogs at shakthimaan.com.

Vinayak Vaid
The author works as an automation engineer at Infosys Limited, Pune. He has worked on different testing technologies and automation tools like QTP, Selenium and Coded UI. He can be contacted at *vinayakvaid91@gmail.com.*