Table of Contents

[1.0 Introduction 3](#_Toc536465687)

[1.1 Purpose 3](#_Toc536465688)

[2.0 Chef Components 3](#_Toc536465689)

[2.1 Workstation 3](#_Toc536465690)

[2.2 Node 3](#_Toc536465691)

[2.3 Chef Server 4](#_Toc536465692)

[2.4 Resources 4](#_Toc536465693)

[2.5 Cookbook 5](#_Toc536465694)

[2.6 Attributes 5](#_Toc536465695)

[2.7 File 5](#_Toc536465696)

[2.8 Policy 6](#_Toc536465697)

[3.0 Installing ChefDK 6](#_Toc536465698)

[3.1 Configuring ChefDK 6](#_Toc536465699)

[3.1.1 Creating the chef repo 6](#_Toc536465700)

[3.1.2 Create the .chef directory 6](#_Toc536465701)

[3.1.3 Starter Kit 7](#_Toc536465702)

[3.2 Configure the Chef Repository – with Web UI 7](#_Toc536465703)

[3.3 Configure the Chef Repository – without Web UI 8](#_Toc536465704)

[3.3.1 Create an Organization on the Chef Server 8](#_Toc536465705)

[3.3.2 Create the User 9](#_Toc536465706)

[3.3.3 Create the config.rb file 9](#_Toc536465707)

[3.3.4 Get SSL Certificate 9](#_Toc536465708)

[3.3.5 Verify Installation 9](#_Toc536465709)

[4.0 Installing Chef Server 10](#_Toc536465710)

[4.1 Installation Process 10](#_Toc536465711)

[5.0 Installing Chef Automate 10](#_Toc536465712)

[5.1 Chef Automate Server Installation and Configuration 11](#_Toc536465713)

[5.2 Configuring node data collection 12](#_Toc536465714)

[5.2.1 Configure a Data Collector token in Chef Automate 12](#_Toc536465715)

[5.2.2 Configure your Chef server to point to Chef Automate 12](#_Toc536465716)

[6.0 Bootstrapping a node 13](#_Toc536465717)

[6.1 Knife bootstrap 13](#_Toc536465718)

[6.2 Prerequisites 14](#_Toc536465719)

[7.0 Berkshelf 14](#_Toc536465720)

[7.1 Using Cookbooks from Supermarket 15](#_Toc536465721)

[8.0 Linux Patching though Chef 16](#_Toc536465722)

[8.1 Prerequisites 16](#_Toc536465723)

[8.1.1 Each node must be bootstrapped. 16](#_Toc536465724)

[8.1.2 Each of the server must be categorized based on environments. 17](#_Toc536465725)

[8.2 Getting the List of Nodes to be patched from Linux Team 17](#_Toc536465726)

[8.3 Pulling the list of available patches. 17](#_Toc536465727)

[8.4.1 Update the audit.txt 17](#_Toc536465728)

[8.4.2 Execute the cookbook 18](#_Toc536465729)

[8.4.3 Generate the reports. 18](#_Toc536465730)

[8.4 Applying the patches 19](#_Toc536465731)

# Introduction

## Purpose

Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.

The module of the Kubernetes:-

* Create a Kubernetes Cluster
* Deploy an Application
* Explore the application
* Expose your application publicly
* Scale up your application
* Update your application

The abstractions in Kubernetes allow you to deploy containerized application to a cluster without tying them specifically to individual machine.

To make use of this model application need to be packaged in a way that decouples them from individual hosts: they need to be containerized. Containerized application are more flexible and available than past deployment models.

Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.

A Kubernetes cluster mainly comprises of the two types of resources. \

* Master
  + Master is responsible for managing the cluster, such as scheduling application, maintaining application’s desired state, scaling application
* Nodes
  + A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster
  + Each node has Kubelet, which is an agent for managing the node and communicating with the Kubernetes Master.
  + Each node should also have tool for handling container operation such as **Docker**.

When you deploy the application in Kubernetes

* You instruct the master to start the application containers
* The Master schedules the containers to run on the cluster’s nodes
* The Node communicates with the master using the **KubernetesAPI**, which is exposed by master.

Once you have your Kubernetes cluster up and running you can deploy the containerized application on top of it. In order to do so, we create a **Deployment** Configuration.

The Deployment instructs Kubernetes how to create and update instances of your application.  Once you've created a Deployment, the Kubernetes master schedules mentioned application instances onto individual Nodes in the cluster.

Once the application instances are created, a Kubernetes Deployment Controller continuously monitors those instances.

If the Node hosting an instance goes down or is deleted, the Deployment controller replaces the instance with an instance on another Node in the cluster.

You can create and manage a Deployment by using the Kubernetes command line interface, **Kubectl**.

Kubectl uses the Kubernetes API to interact with the cluster.

When you created a Deployment, Kubernetes created a **POD** to host your application instance. A POD is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker) and some shared resources for these containers.

A Pod models an application-specific "logical host" and can contain different application containers which are relatively tightly coupled.

The containers in a Pod share an **IP Address** and **port space**, are always co-located and co-scheduled, and run in a shared context on the same Node.

A Pod always runs on a **Node**. A Node is a worker machine in Kubernetes and may be either a virtual or a physical machine.

You use the command line interface **Kubectl**, to interact with the KubernetesAPI and get information about the application deployed and their environment. The most common operation that can be done by the **Kubectl** are.

* **kubectl get** :- list resources
* **kubectl describe**:- show detailed information about resources
* **kubectl logs** :- print logs information
* **kubectl exec**:- executes command in a POD

A **service** in Kubernetes is an abstraction which defines a logical set of POD and a policy by which to access them. Service enable a loose coupling between the dependant POD. A service is defined using YAML. The set of POD targeted by the Service are identified by ***LabelSelector***. Services allow application to receive traffic.

## Workstation

One (or more) workstations are configured to allow users to author, test, and maintain cookbooks. Cookbooks are uploaded to the Chef server from the workstation. Some cookbooks are custom to the organization and others are based on community cookbooks available from the Chef Supermarket.

Chef workstation includes important common-line tools

* Chef
  + Use the chef command-line tool to work with items in a chef-repo.
* Knife
  + Use the knife command-line tool to interact with nodes or work with objects on the Chef server

## Node

A node is any machine—physical, virtual, cloud, network device, etc.—that is under management by Chef.

A chef-client is installed on every node that is under management by Chef. The chef-client performs all of the configuration tasks that are specified by the run-list and will pull down any required configuration data from the Chef server as it is needed during the chef-client run.

Key components of node that are under the management by Chef include.

* Chef –client
  + When a chef-client is run, it will perform all of the steps that are required to bring the node into the expected state. Like
    - Registering and authenticating the node with the Chef Server
    - Building the node object
    - Synchronizing cookbook
    - Compiling the resources

RSA public key-pairs are used to authenticate the chef-client with the Chef server every time a chef-client needs access to data that is stored on the Chef server.

* Ohai

## Chef Server

The Chef server acts as a hub of information. Cookbooks and policy settings are uploaded to the Chef server by users from workstations.

The chef-client accesses the Chef server from the node on which it’s installed to get configuration data, performs searches of historical chef-client run data, and then pulls down the necessary configuration data. After the chef-client run is finished, the chef-client uploads updated run data to the Chef server.

## Resources

A resource is a statement of configuration policy that

* Describes the desired state for a configuration item
* Declares the steps needed to bring that item to the desired state
* Specifies a resource type—such as package, template, or service
* Lists additional details (also known as resource properties), as necessary
* Are grouped into recipes, which describe working configurations

A resource is a ruby block with 4 major components; a type, a name, one (or more) properties with values, one (or more actions).

type 'name' do

attribute 'value'

action :type\_of\_action

end

A resource used to install a tar.gz package.

package 'tar' do

version '1.16.1'

action: install

end

The complete list of resources can are at: <https://docs.chef.io/resource_reference.html>

## Cookbook

Use the knife command-line tool to interact with nodes or work with objects on the Chef server.  A cookbook defines a scenario and contains everything that is required to support that scenario.

* Recipes that specify the resources to use and the order in which they are to be applied
* Attribute values
* File distributions
* Templates
* Extensions to Chef, such as custom resources and libraries

Some of the components of the Chef Cookbook are as follows

* Attributes
* Files
* Libraries
* Metadata
* Recipes

Chef command to generate the cookbook

*S chef generate cookbook FOO*

## Attributes

An attribute is a specific detail about a node. Attributes are defined by these

* The state of the node itself
* Cookbooks (in attribute files and/or recipes)
* Roles
* Environments

To generate an attribute, use the following command.

*$ chef generate attribute FOO*

## File

Use the cookbook\_file resource to transfer files from a sub-directory of COOKBOOK\_NAME/files/ to a specified path located on a host that is running the chef-client. The command to create a ‘file’ folder in cookbook

*$ chef generate file COOKBOOK\_NAME*

## Policy

Policy maps business and operational requirements, process, and workflow to settings and objects stored on the Chef server.

By using policies, you can apply a specific set of cookbooks to a node or nodes with a single document.

<https://blog.chef.io/2015/10/05/policyfiles-why-what-and-how/>

# Installing ChefDK

Use the Chef Development Kit installer to set up ChefDK on a workstation. ChefDK includes the chef-client, an embedded version of Ruby, RubyGems, and OpenSSL, as well as our tools: Test Kitchen, Cookstyle, Foodcritic and ChefSpec.

The chef installer will put everything into a unique directory (/opt/chefdk) so that these components will not interfere with other applications.

Download the [Chef](https://downloads.chef.io/chefdk) download page and download the appropriate application package.

## Configuring ChefDK

After [installing](https://docs.chef.io/install_dk.html) chef, the next step will be to configure Chef.

* + 1. Creating the chef repo

Use the “chef generate repo” command to create the Chef Repository.

Ex: chef generate repo <chef-repo-name>

* + 1. Create the .chef directory

The “.chef” directory is the most important folder in the Chef installation process. It usually consists of the following three files :-

* Config.rb (or knife.rb)
* Organization-validator.pem
* User.pem

These files must be present in the “.chef” directory in order for the ChefDK to be able to connect with the Chef Server.

* + 1. Starter Kit

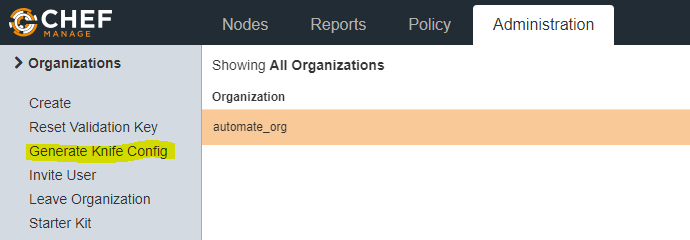
If you have access to Chef server through Automate or Chef Manage, you can download the starter kit.

The starter kit will create the necessary configuration files: the .chef directory, config.rb, ORGANIZATION-validator.pem, and USER.pem. Simply download the starter kit and then move it to the desired location on your ChefDK machine

## Configure the Chef Repository – with Web UI

Log on to the Chef Server and download the following files

* Config.rb (or knife.rb)



The config.rb (or knife.rb) will be automatically downloaded. The following is a sample file

# See https://docs.getchef.com/config\_rb\_knife.html for more information on knife configuration options

current\_dir = File.dirname(\_\_FILE\_\_)

log\_level :info

log\_location STDOUT

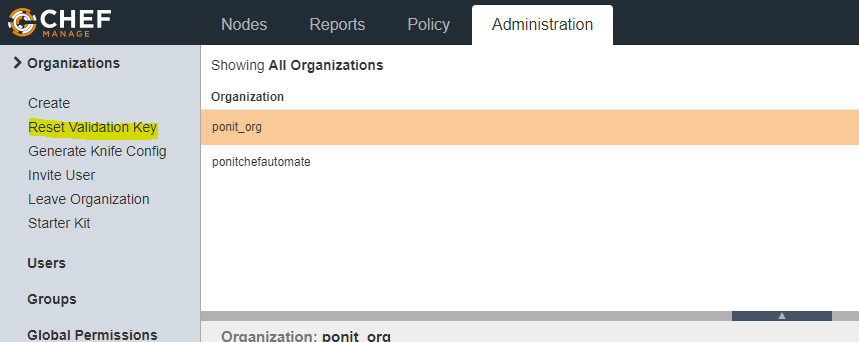
node\_name "delivery"

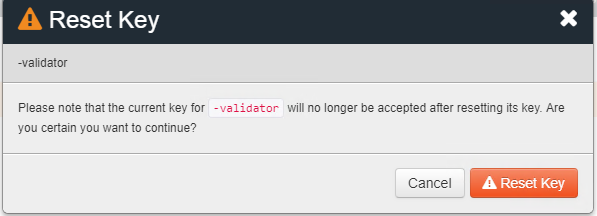
client\_key "#{current\_dir}/delivery.pem"

chef\_server\_url "https://spaapp007/organizations/automate\_org"

cookbook\_path ["#{current\_dir}/../cookbooks"]

* Organization-validator.pem

The private key can be downloaded from the ‘Reset Validation Key’ option. You will be prompted a warning message



The private key can be downloaded. Please save the same.

**NOTE: Please don’t reset the validation key, doing so will disrupt all the existing connections.**

* User.pem

The private key can be downloaded from the Change Password section of the Account Management Page.

Place all the three file in the “ .chef” directory in the chef-repo

## Configure the Chef Repository – without Web UI

* + 1. Create an Organization on the Chef Server

On the Chef Server machine, create an Organization using the chef-server-ctl command.

$ chef-server-ctl org-create ORG\_NAME ORG\_FULL\_NAME -f FILE\_NAME

Where

ORG\_NAME: Represents the Organization Shortname

ORG\_FULL\_NAME: Represents the Organization Full/Complete Name

FILE\_NAME: The location where the Organization.pem file will be downloaded to.

Ex.

chef-server-ctl org-create test "TESTINF INC." -f /home/chefadmin/testing.pem

You can use the ‘chef-server-ctl org-list’ command to list the created organization.

* + 1. Create the User

On the Chef server machine create the USER.pem from the command line using chef-server-ctl.

$ chef-server-ctl user-create USER\_NAME FIRST\_NAME LAST\_NAME EMAIL PASSWORD -f FILE\_NAME

Where

USER\_NAME: Name of User

FIRST\_NAME:

LAST\_NAME

EMAIL

PASSWORD

FILE\_NAME: Location where the .pem file will be downloaded to.

$ chef-server-ctl user-create vijayvijay tiger VijayKumar.V4@xxx.com temp123 -f /home/chefadmin/vijay.pem

* + 1. Create the config.rb file

Navigate to the ~/chef-repo/.chef directory and create the config.rb using the knife configure tool.

* + 1. Get SSL Certificate

Chef server 12 enables SSL verification by default for all requests made to the server, such as those made by knife and the chef-client. The certificate generated during the installation of Chef Server is self-signed. this certificate must be downloaded to any machine from which knife and/or the chef-client will make requests to the Chef server.

Use the ‘*knife ssl fetch*’ to pull the certificates from the Chef Server to the local machine.

* + 1. Verify Installation

The ChefDK is installed correctly when it is able to use knife to communicate with the Chef server. Use any knife command to check if the connection is established or not.

Ex: knife node list, knife list, knife ssl check

# Installing Chef Server

The standalone installation of Chef server creates a working installation on a single server. This installation is also useful when you are installing Chef server in a virtual machine.

## Installation Process

* Download the package from <https://downloads.chef.io/chef-server/>
* Upload the package on the machine on which Chef-Server will be loaded
* Record the location of the package.
* Assuming that the package has been uploaded to /tmp directory
* As a root user, install the package
  + RHEL/Centos: $ sudo rpm -Uvh /tmp/chef-server-core-<version>.rpm
  + Ubuntu: $ sudodpkg -i /tmp/chef-server-core-<version>.deb
* Once the Chef-Server has been installed, run the following command to start the service
  + $ sudo chef-server-ctl reconfigure
* Next, execute the command to create an administrator
* $ sudo chef-server-ctl user-create USER\_NAME FIRST\_NAME LAST\_NAME EMAIL PASSWORD' --filename FILE\_NAME
  + Ex: chef-server-ctl user-create delivery Delivery user Gayathri.G6@xxxx.com 'delivery' --filename delivery\_automate.pem
* The RSA Private key will be generated automatically and stored in the path ‘FILE\_NAME’
* This file needs to be preserved.
* Now to create an organization
  + $ sudo chef-server-ctl org-create short\_name 'full\_organization\_name' --association\_useruser\_name --filename ORGANIZATION-validator.pem
  + Ex: chef-server-ctl org-create automate\_org 'Automate Chef Org' --filename automate\_org-validator.pem -a delivery
* The RSA private key is generated automatically. This is the chef-validator key and should be saved to a safe location.

# Installing Chef Automate

A Chef Automate Server installation consists of a minimum of two server

* Chef Server
* Chef Automate Server

And optimal servers include

* Push job Server
* Runners/build nodes.

Chef Automate associates with a Chef server for application/cookbook building and testing, as well as for the proxying of cluster data from nodes for visualization purposes

Create a Chef Server Organization and User if not already created.

Create a user:

*sudo chef-server-ctl user-create delivery FIRST\_NAME LAST\_NAME EMAIL\_ADDRESS 'PASSWORD' --filename AUTOMATE\_CHEF\_USER\_KEY*

Creating an Organization and associating the created user:

*sudo chef-server-ctl org-create AUTOMATE\_CHEF\_ORG 'org description' --filename ~/AUTOMATE\_CHEF\_ORG-validator.pem -a delivery*

Run the following commands on the Chef Server to complete the configuration on Chef Server

*sudo chef-server-ctl reconfigure*

## Chef Automate Server Installation and Configuration

* Download and install the latest stable Chef Automate Package from [https://downloads.chef.io/automate](https://downloads.chef.io/automate/)
* Execute the ‘*preflight-check*’ to uncover common environmental problems.

sudo automate-ctl preflight-check

* Ensure that the Chef Automate license file and the delivery user key you created earlier in the Chef Server setup are located on the Chef Automate server.
* Run the setup command. This command requires root user privileges. Any unsupplied arguments will be prompted for

sudo automate-ctl setup --license AUTOMATE\_LICENSE \

--key AUTOMATE\_CHEF\_USER\_KEY \

--server-url https://CHEF\_SERVER\_FQDN/organizations/AUTOMATE\_CHEF\_ORG \

--fqdn AUTOMATE\_SERVER\_FQDN \

--enterprise ENTERPRISE\_NAME \

--supermarket-fqdn SUPERMARKET\_FQDN (Optional)

Ex.

automate-ctl setup --license automate.license --key delivery\_automate.pem --server-url https://chefserverprod.ponit.com/organizations/automate\_org --fqdn chefautomateprod.chef.com --enterprise automate\_org

Where

AUTOMATE\_LICENSE: The path of the license key

AUTOMATE\_CHEF\_USER\_KEY: pem file of the user created in Chef Server

server-url: Chef Server url

## Configuring node data collection

After you have set up your Chef Server and Chef Automate server, you must perform some simple configuration steps to visualize node data in Chef Automate.

Nodes should send their run data to Chef Automate through the Chef server automatically. To enable this functionality, you must perform the following steps

* Configure a Data Collector token in Chef Automate
* Configure your Chef server to point to Chef Automate
  + 1. Configure a Data Collector token in Chef Automate

All messages sent to Chef Automate are performed over HTTP and are authenticated with a pre-shared key called a token.

* To set your own token, add the following to your /etc/delivery/delivery.rb file

*data\_collector['token'] = 'sometokenvalue'*

* To apply changes, use the command

*sudo automate-ctl reconfigure*

* + 1. Configure your Chef server to point to Chef Automate

In addition to forwarding Chef run data to Automate, Chef server will send messages to Chef Automate whenever an action is taken on a Chef server object, such as when a cookbook is uploaded to the Chef server.

* Channel the token setting through the veil secrets library because the token is considered a secret, and cannot appear in /etc/opscode/chef-server.rb

sudo chef-server-ctl set-secret data\_collector token 'TOKEN'

sudo chef-server-ctl restart nginx

sudo chef-server-ctl restart opscode-erchef

* Finally, in the file /etc/opscode/chef-server.rb on Chef Server added these

data\_collector['root\_url'] = 'https://my-automate-server.mycompany.com/data-collector/v0/'

# Add for compliance scanning

profiles['root\_url'] = 'https://my-automate-server.mycompany.com'

# Save and close the file

* To apply changes, use the command

*sudo chef-server-ctl reconfigure*

Chef Automate requires a license from Chef to install.

These are the steps to configure Chef Automate

* The Chef Server should already have been setup
* We have created an organization in the Chef Server
* <https://getchef.zendesk.com/hc/en-us/articles/115000582846-Automate-Setup-Instructions>
* Fetch the license and place it in the /home/chefadmin folder.
* Download & Install the appropriate package from

<https://downloads.chef.io/automate/1.8.96>

*wget*[*https://packages.chef.io/files/stable/automate/1.8.68/el/7/automate-1.8.68-1.el7.x86\_64.rpm*](https://packages.chef.io/files/stable/automate/1.8.68/el/7/automate-1.8.68-1.el7.x86_64.rpm)

*rpm -Uvh automate-1.8.68-1.el7.x86\_64.rpm*

* Execute the pre-flight check command. This will help identify if there are some configurational issues.
* automate-ctl setup --license /home/chefadmin/automate.license --key /home/chefadmin/delivery\_automate.pem --server-url https://spaapp007/organizations/automate\_org --fqdn spaapp029 --enterprise automate\_org

# Bootstrapping a node

A node is any physical, virtual, or cloud machine that is configured to be maintained by a chef-client. In order to bootstrap a node, you will first need a working installation of the Chef Software package.

A bootstrap is a process that installs the chef-client on a target system so that it can run as a chef-client and communicate with a Chef server.

## Knife bootstrap

The knife bootstrap command is a common way to install the chef-client on a node. The default for this approach assumes that a node can access the Chef website so that it may download the chef-client package from that location.

The Chef installer puts everything into a unique directory (/opt/chef/) so that the chef-client will not interfere with other applications that may be running on the target machine.

* Identify the FQDN or IP address of the target node. The knife bootstrap command requires the FQDN or the IP address for the node in order to complete the bootstrap operation.
* Once the workstation machine is configured, it can be used to install the chef-client on one (or more) nodes across the organization using a knife bootstrap operation
* The knife bootstrap command is used to SSH into the target machine
* It will install the chef-client executable (if necessary), generate keys, and register the node with the Chef server.
* The bootstrap operation requires the IP address or FQDN of the target system, the SSH credentials (username, password or identity file) for an account that has root access to the node

Ex: $ knife bootstrap 123.45.6.789 -x username -P password –sudo

* In order to verify the operation, run the following command

$ knife client show name\_of\_node

## Prerequisites

* In the node’s host file, we must add an entry for the Chef-Server.
  + Ex:10.227.70.220 spaapp007
* From the node on which we need to install the chef-client. We need to be able to telnet to the ports 22, 443, 80.
  + telnet 10.227.70.220 22
  + telnet 10.227.70.220 443
  + telnet 10.227.70.220 80
* From the chef-server, we should be able to connect to the Node on port 22
  + telnet <node-ip> 22
* Boot strap command must be run from the .chef directory
* Command to bootstrap a node
  + knife bootstrap <ip> --sudo -x chefadmin -P 'Q7trskse15#' -N <hostname>

Ex:

* + knife bootstrap 10.225.198.181 --sudo -x chefadmin -P 'Q7trskse15#' -N SAAAPP005
  + knife bootstrap 10.225.198.92 --sudo -x chefadmin -P 'Q7trskse15#' -N SAAAPP006
  + knife bootstrap 10.225.198.21 --sudo -x chefadmin -P 'Q7trskse15#' -N SAAMONI001
  + knife bootstrap 10.225.202.24 --sudo -x chefadmin -P 'Q7trskse15#' -N SAAWEB001

# Berkshelf

Berkshelf is a dependency manager for Chef cookbooks. With it, you can easily depend on community cookbooks and have them safely included in your workflow.

Running a ‘chef generate cookbook’ command will by default create a file called Berksfile in the root of the cookbook, alongside cookbook’s metadata.

The default Berksfile, will not have any dependencies.

# frozen\_string\_literal: true

source 'https://supermarket.chef.io'

metadata

The metadata.rb will look like the following.

name 'vijaya'

maintainer 'The Authors'

maintainer\_email 'you@example.com'

license 'All Rights Reserved'

description 'Installs/Configures vijaya'

long\_description 'Installs/Configures vijaya'

version '0.1.0'

chef\_version '>= 12.14' if respond\_to?(:chef\_version)

# The `issues\_url` points to the location where issues for this cookbook are

# tracked. A `View Issues` link will be displayed on this cookbook's page when

# uploaded to a Supermarket.

#

# issues\_url 'https://github.com/<insert\_org\_here>/vijaya/issues'

# The `source\_url` points to the development repository for this cookbook. A

# `View Source` link will be displayed on this cookbook's page when uploaded to

# a Supermarket.

#

# source\_url 'https://github.com/<insert\_org\_here>/vijaya'

You will have to add the cookbook’s dependencies to the metadata.

Add your cookbook dependencies to the metadata.rb, like

name 'my\_first\_cookbook'

version '0.1.0'

depends 'apt', '~> 5.0'

Now, when we execute the ‘*berks install*’, the apt code can be downloaded from Supermarket into the Cache.

Running the install command creates a Berksfile.lock.

Now you can upload all the cookbook to the Chef Server with the “*berks upload*”

## Using Cookbooks from Supermarket

In order to perform the audit operation, we use the audit cookbook from the Chef Supermarket. The audit cookbook is available at <https://github.com/chef-cookbooks/audit>

In order to send the compliance information to Chef Automate from the audit cookbook, follow the instructions at <https://docs.chef.io/audit_cookbook.html>

In our chef-repo, I have renamed the audit cookbook as (audit\_master) since there was already a cookbook named audit already created.

Generate a wrapper cookbook: point\_\_audit. (*chef generate cookbook point\_\_audit*).

Add the following lines of code in the metadata.rb file ‘*depends audit*’

Add the following the point\_\_audit /recipes/default.rb: ‘include\_recipe 'audit\_master::default'

Generate the default attributes file if not already existing ‘*chef generate attribute default*’

Configure the audit cookbook reporter to send scan data to Automate in the

*point\_\_audit /attributes/default.rb file*

default['audit']['reporter'] = 'chef-server-automate'

case node['os']

when 'linux'

default['audit']['profiles'] = [

{

name: 'DevSec Linux Patch Benchmark',

compliance: 'admin/linux-patch-baseline',

},

]

when 'windows'

default['audit']['profiles'] = [

{

name: 'DevSec Windows Security Baseline',

compliance: 'admin/windows-baseline',

},

]

end

Use Berkshelf to install cookbook dependencies and upload it to all Chef Servers.

*berks install*&&*berks upload*

# Linux Patching though Chef

The steps to be taken during the Linux Patching are as follows. The Linux server are patched every quarter. Here are the details

## Prerequisites

* + 1. Each node must be bootstrapped.

In case a new server has been added to the repository. The same needs to be bootstrapped before we can start the process of running chef-client on them

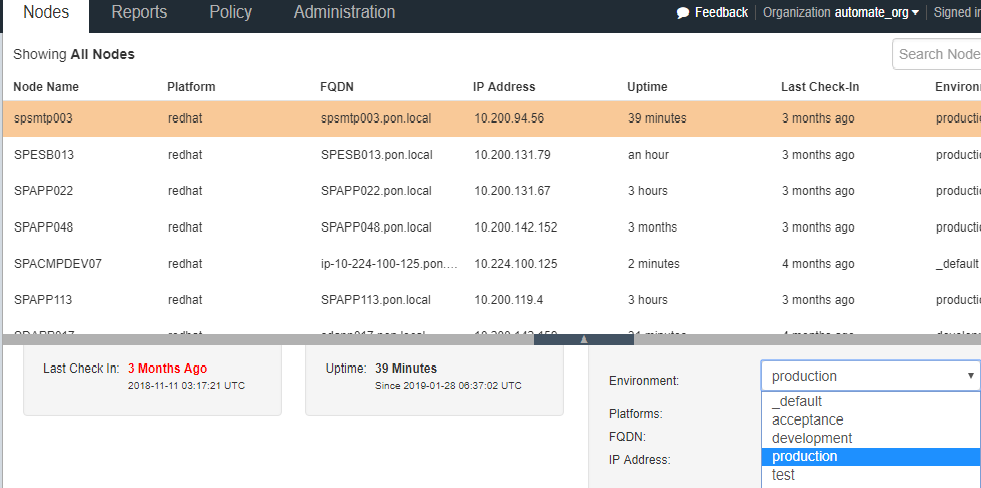
Use the IP, Hostname and bootstrap the node as explained earlier.

Once the node is bootstrapped, the same will be displayed in the Chef Server UI.

* + 1. Each of the server must be categorized based on environments.

Each of the server need be categorized based on environments as ‘acceptance’, ‘development’, ‘production’ or test.

To map a server to the environment, search and select the node. Choose the appropriate environment from the drop down and save.



## Getting the List of Nodes to be patched from Linux Team

Atleast a week before the schedule the Linux team will publish the list of servers to be patched. Please follow up with team in case they have not done so. At times there could be issues from the Chef’s end and hence it’s better to have some buffer time.

Please don’t use the existing repo, since the same gets updated regularly by the Linux Team.

## Pulling the list of available patches.

We need the provide the list of available patches to the Linux team, they will publish them in the meeting with the stakeholders.

In order to get the list of available patches, we need to execute the point\_\_audit cookbook for all the nodes listed.

* + 1. Update the audit.txt

Navigate to */home/ec2-user/automate-repo/scripts/patch\_scripts/pre\_patch/Tushar* folder

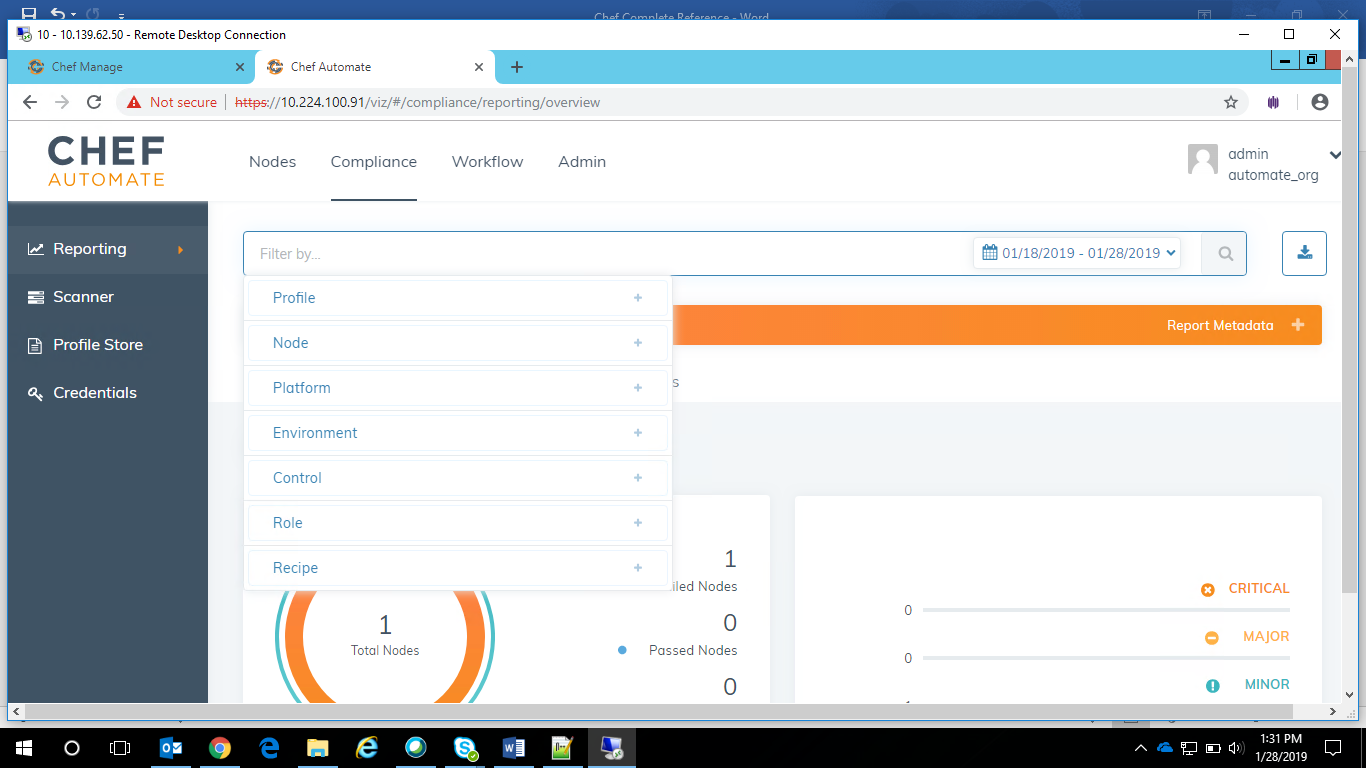
This file contains the list of the IP address of the nodes that needs to be patched. Remove the existing entries and add the list of the IP address provided by Linux Team.

* + 1. Execute the cookbook

Run script ‘*ip\_audit.sh*’ to execute the ‘*ponit\_\_audit*’ cookbook for all the nodes. This script generates logs which are stored as ‘*chef-client.log*’, in the same folder location.

* + 1. Generate the reports.

The report to be sent across to the Linux team, need to be generated from the Chef Automate. Log into Chef Automate and navigate to the compliance tab.



From the filter criteria, select the ‘Node’ and provide the hostname of the nodes. This will give provide the following information. Check the same for all the nodes

* Is the node complaint

If the node is complaint, it suggests that no patches are available to be installed.

* If not, the list of patches that are available to be installed.

In case there are patches available to be installed, the entire list of patches can download in CSV or JSON formats from the download option, just besides the filter text box.

The cumulative report of all the nodes can be downloaded.

Send the downloaded report to the Linux team.

## Applying the patches

In order to apply the patches, we need to execute the ‘*linux\_patch*’ cookbook for all the nodes. In order to do so execute the script ‘*ip\_patching.sh*’ Again the generated logs will be stored in chef-client.log.

After the patching, execute the ‘*ponit\_\_audit*’ to check if all the nodes are complaint or not.