How To Understand the Chef Configuration Environment on a VPS

Posted Nov 20, 2013

Top of Form

Bottom of Form

12

 35.6kviews [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Ruby](https://www.digitalocean.com/community/tags/ruby?type=tutorials) [Scaling](https://www.digitalocean.com/community/tags/scaling?type=tutorials)

Tutorial Series

This tutorial is part 1 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-understand-the-chef-configuration-environment-on-a-vps#tutorial_series_10)

**Introduction**

Configuration management tools provide an avenue for deploying consistent, predictable code and configurations to a variety of client computers from a centralized management server. **Chef** is one of the most popular configuration management tools. It uses Ruby and handles configuration by packing details into what it calls recipes.

Chef provides a way to quickly deploy entire environments instead of only single applications. In any situation where you would install a piece of software and then modify its configuration files, Chef can be used to automate this process.

In this guide, we will provide a general overview of how Chef organizes its files and what tools and systems it uses to accomplish its objectives.

If you would like to follow along, there is a tutorial on [how to install Chef 12 on Ubuntu 14.04](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers) here.

Chef Terminology

It is important to understand the different components that make up Chef.

**Chef Operating Infrastructure**

We will start by discussing the different models that make up the high level deployment strategy.

The Chef system is defined by the roles that each machine or resource plays in the deployment process:

* **Chef Server**: This is the central location that stores configuration recipes, cookbooks, and node and workstation definitions. It is the central machine that every other machine in the organization will use for deployment configuration.
* **Chef Nodes**: Chef nodes are the deployment targets that are configured by Chef. Each node represents a separate, contained machine environment that can be on physical hardware or virtualized.

These operating system environments each contain a Chef client application that can communicate with the Chef Server.

* **Chef Workstations**: Chef workstations are where Chef configuration details are created or edited. The configuration files are then pushed to the Chef server, where they will be available to deploy to any nodes.

The configuration of these different components allows you to have multiple workstations and nodes. Nodes can be configured as soon as they are online and connected to the server.

While the above outline gives the impression that these are separate entities, it is possible for one machine to fulfill two or all of these roles. There is a project called **chef-solo** which allows you to forgo the use of a server and operate by configuring the computer which it is installed.

**Server Details**

The server is the central control point that is accessed by all of the other chef machines, whether as a client or a manager. It is basically a large repository or database of all of the configuration details.

It handles connections and permissions from nodes and workstations and organizes data so that it can easily be pulled by clients. The server can also include a web interface in order to manage or configure some details.

**Node Details**

As mentioned above, a node can be a physical or virtual machine. Its only requirements are that it has access to the network and can communicate with the chef server. The user running the chef software also needs to be able to install software and make system changes.

Each node communicates with the central server using an application called **chef-client**. This handles pulling data off of the server and executing the configuration steps necessary to get the node into its final state. The chef-client program and the chef server communicate through the use of RSA key-based authentication.

Chef-client uses a tool called **ohai** to get statistics about the node. These are used in order to set up certain configuration details and populate variables contained within the files.

**Workstation Details**

A workstation has the tools necessary to create and modify configuration details for any of the available nodes and can communicate with the chef server to make these available.

An important tool to manage chef on a workstation is called **knife**. Knife acts as a gateway in which you can configure anything that would be stored on the server. It can manage nodes and configurations and can generally be used to access the server in a "chef-specific" way. While it would be possible to log into the server with SSH and make changes to all of the data that it handles manually, this is not really adhering to the processes that chef implements.

Configurations and definitions that are created and modified on a workstation are committed to version control and then pushed to the server. The repository is called the **chef-repo**. It holds all of the data needed for the configuration of chef.

Chef Repo File Structure

Chef handles its configuration and dependency information on a workstation within a specified directory structure. It is important to understand this hierarchy in order to effectively create recipes and push changes.

As we mentioned above, the server configuration files should be kept in version control in repository referred to as the "chef-repo". This is just a normal directory that contains the chef files.

In this directory, we can find a structure that looks like this:

* **certificates/**: Contains the SSL certificates that can be associated with clients for authentication.
* **chefignore**: Lists the files and directories within the structure that should not be included in the push to the server.
* **config/**: Contains one of the two repository configuration files
  + **rake.rb**: Defines some variable declarations for creating SSL certificates and some general options.
* **cookbooks/**: Contains the cookbooks that configure the infrastructure for your organization.
* **data\_bags/**: Contains various data bags for your configuration.

Data bags are protected sub-directories that contain sensitive configuration details. They are only accessible to those nodes that have matching SSL certificates and contain JSON formated files with configuration details.

* **environments/**: Contains a top-level location to contain details for deploying the environment.

Every environment that diverges from the default environment must be defined in this directory.

* **Rakefile**: This file defines the tasks that chef can perform in its configurations.
* **roles/**: Contains files that define the roles that can be assigned to nodes.

**Chef Cookbook File Structure**

Within the cookbooks directory in the chef-repo, sub-directories define specific cookbooks for applications. Within each separate application configuration directory is a structure that defines how this service should be installed and what changes must be made to make it work correctly.

Within the application, you will find files and definitions that define how an application must be installed and configured.

The **metadata.rb** or **metadata.json** files contains metadata information about the service. This includes basic information like the name of the cookbook and the version, but it also is the place where the dependency information is stored. If this cookbook depends on other cookbooks to be installed, it can list them in this file and chef will install and configure them prior to the current cookbook.

The **attributes** directory contains attribute definitions that can be used to override or define settings for the nodes that will have this service.

The **definitions** directory contains files that declare resources. This means that you can group functionality together under one heading.

The **files** directory describes how chef should distribute files throughout the node on which this cookbook is deployed.

The **recipes** directory contains the "recipes" that define how the service should be configured. Recipes are generally small files that configure specific aspects of the larger system. If a cookbook used to install and configure a web server, a recipe may enable a module or set up a sane firewall default.

The **templates** directory is used to provide more complex configuration management. You can provide entire configuration files that contain embedded Ruby commands. The variables that are printed can be defined in other files.

Conclusion

While this guide may not help you get started writing your own Chef configurations, it should give you a good overview as to what the individual components are in a complex deployment environment. Once you begin to understand how the node, server, workstation interaction works, and can find your way around the chef-repo, you can begin to start understanding how some of the cookbooks available operate.

In [the next aricle](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers), we will discuss how to set up Chef 12 on Ubuntu 14.04 servers. Later on, we will also demonstrate [how to create some of your own cookbooks](https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu) and configure an environment that can be deployed to other machines within your network.

# How To Set Up a Chef 12 Configuration Management System on Ubuntu 14.04 Servers

Posted Mar 3, 2015

Top of Form

Bottom of Form

18

 67.7kviews [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

## Tutorial Series

This tutorial is part 2 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers#tutorial_series_10)

### Introduction

As your infrastructure requirements expand, managing each server by hand becomes an increasingly difficult task. This difficulty is compounded by the requirement for reproducibility, which becomes necessary if a node fails or if horizontal scaling is needed.

Configuration management solutions are designed to address these issues by turning your infrastructure administration into a code base. Instead of performing individual tasks on a number of machines, these tools allow you to commit your requirements to a central location where each component can connect, pull down their configuration, and apply it.

In a [previous guide](https://www.digitalocean.com/community/articles/how-to-understand-the-chef-configuration-environment-on-a-vps), we talked, on a conceptual level, about the general structure of Chef components and the way in which they interact to achieve the administrator's objectives. We talked about relevant terminology and discussed the responsibility of each piece.

In this guide, we will install the actual software. We will set up a centralized Chef server which will store and serve configuration instructions and node profiling information. We will also set up a workstation where the administrator can work with the code base and alter the characteristics of the infrastructure. We will follow this up by bootstrapping a new node to bring it under the management of the Chef ecosystem.

## Prerequisites and Goals

We will be setting up version 12 of Chef in this guide. Configuration can be significantly different between versions, so ensure that you are operating within the same major version number as this guide for best results.

The [Chef documentation](http://docs.chef.io/server/install_server_pre.html#hardware-software) tells us that your Chef server should have at least 4 cores and 4 GB of RAM. It should also have a [64-bit operating system](http://docs.chef.io/server/install_server.html#prerequisites). For our guide, we will be using an 4 core / 8 GB DigitalOcean Droplet with 64-bit Ubuntu 14.04.

The workstation and nodes have very few requirements. We will use Ubuntu 14.04 on those as well for consistency.

When we are finished, we will have a centralized Chef server to store and serve our configuration data. Our workstation will be used to make changes, upload them to the server, and bootstrap and manage new nodes. The node represents a single server within our infrastructure.

## Configure the Chef Server

We will begin by setting up the Chef server. Remember, Chef recommends at least 4 cores and 4 GB of RAM for this server, so plan accordingly.

### Ensure that the Server is Accessible by Hostname

Once you are logged into the server you plan on installing the Chef server onto, the first task you need to perform is to ensure that the hostname of the server is a resolvable fully qualified domain name (FQDN) or IP address. You can check this by typing:

hostname -f

The result should be an address where the server can be reached. If this is not the case, you can set this to a domain name or IP address where the server can be reached by editing this file:

sudo nano /etc/hosts

The file will look similar to this:

127.0.1.1 current\_hostname current\_hostname\_alias

127.0.0.1 localhost

. . .

Modify the top line to reflect the fully qualified domain name or the IP address, followed by a space and any alias you want to use for your host. Add a line beneath the two lines shown that has your server's public IP address in the first column, and the information that you modified at the end of the 127.0.1.1line to the end. It should look something like this:

127.0.1.1 fqdn\_or\_IP\_address host\_alias

127.0.0.1 localhost

IP\_address fqdn\_or\_IP\_address host\_alias

So, if I do not have a domain name, my public IP address is 123.123.123.123, and if I also want my host reachable by the hostname "chef", I could have a file that looks like this:

127.0.1.1 123.123.123.123 chef

127.0.0.1 localhost

123.123.123.123 123.123.123.123 chef

If, on the other hand, this server has the fully qualified domain name of chef.example.com and an IP address of 234.234.234.234, my file might look something like this instead:

127.0.1.1 chef.example.com chef

127.0.0.1 localhost

234.234.234.234 chef.example.com chef

Save and close the file when you are finished. You can check that the value was set correctly by typing:

hostname -f

The result should be a value that you can use to reach your Chef server from anywhere in your infrastructure.

### Download and Install the Chef 12 Server software

Next, we can go ahead and download the Chef 12 server software. You can find the package that must be installed by visiting the Chef site. Specifically, for an Ubuntu installation, you can follow [this link](https://downloads.chef.io/chef-server/ubuntu/#/).

Under the "Ubuntu Linux 14.04" header, right-click on the download link and copy the link location:



Back on your server, change to your home directory. Paste the link you copied and use the wgetcommand to download the package. The link you copied may be different from the one below if there has been a minor version update since this writing:

cd ~

wget https://web-dl.packagecloud.io/chef/stable/packages/ubuntu/trusty/chef-server-core\_12.0.5-1\_amd64.deb

Once the download is complete, install the package by typing:

sudo dpkg -i chef-server-core\_\*.deb

This will install the base Chef 12 system onto the server. If you have selected a server with less powerful hardware than the recommended amount, this step may fail.

Once the installation is complete, you must call the reconfigure command, which configures the components that make up the server to work together in your specific environment:

sudo chef-server-ctl reconfigure

### Create an Admin User and Organization

Next, we need to create an admin user. This will be the username that will have access to make changes to the infrastructure components in the organization we will be creating.

We can do this using the user-create subcommand of the chef-server-ctl command. The command requires a number of fields to be passed in during the creation process. The general syntax is:

chef-server-ctl user-create USERNAME FIRST\_NAME LAST\_NAME EMAIL PASSWORD

We will include this information, and will also add -f, an additional flag, onto the end in order to specify a filename in which to output our new user's private RSA key. We will need this in order to authenticate using the knife management command later.

For our example, we will create a user with the following information:

* **Username**: admin
* **First Name**: admin
* **Last Name**: admin
* **Email**: admin@example.com
* **Password**: examplepass
* **Filename**: admin.pem

The command needed to create a user with this information is (you should change this to reflect your information, especially the password):

sudo chef-server-ctl user-create admin admin admin admin@example.com examplepass -f admin.pem

You should now have a private key called admin.pem in your current directory.

Now that you have a user, you can create an organization with the org-create subcommand. An organization is simply a grouping of infrastructure and configuration within Chef. The command has the following general syntax:

chef-server-ctl org-create SHORTNAME LONGNAME --association\_user USERNAME

The short name is the name that you will use to refer to the organization from within Chef. The long name is the actual name of the organization. The --association\_user specifies the username that has access to administer the organization. Again, we will add the -f flag so that we can specify the name of the file to place the private key. The key that will be created is used to validate new clients as part of the organization until they can get their own unique client key.

We will create an organization with the following qualities:

* **Short Name**: digitalocean
* **Long Name**: DigitalOcean, Inc.
* **Association User**: admin
* **Filename**: digitalocean-validator.pem

To create an organization with the above qualities, we will use the following command:

sudo chef-server-ctl org-create digitalocean "DigitalOcean, Inc." --association\_user admin -f digitalocean-validator.pem

Following this, you should have two .pem key files in your home directory. In our case, they will be calledadmin.pem and digitalocean-validator.pem. We will need to connect to this server and download these keys to our workstation momentarily. For now though, our Chef server installation is complete.

## Configure a Chef Workstation

Now that our Chef server is up and running, our next course of action is to configure a workstation. The actual infrastructure coordination and configuration does not take place on the Chef server. This work is done on a workstation which then uploads the data to the server to influence the Chef environment.

### Clone the Chef Repo

The Chef configuration for your infrastructure is maintained in a hierarchical file structure known collectively as a Chef repo. The general structure of this can be found in a GitHub repository provided by the Chef team. We will use git to clone this repo onto our workstation to work as a basis for our infrastructure's Chef repository.

First, we need to install git through the apt packaging tools. Update your packaging index and install the tool by typing:

sudo apt-get update

sudo apt-get install git

Once you have git installed, you can clone the Chef repository onto your machine. For this guide, we will simply clone it to our home directory:

cd ~

git clone https://github.com/chef/chef-repo.git

This will pull down the basic Chef repo structure into a directory called chef-repo in your home directory.

### Putting your Chef Repo Under Version Control

The configurations authored within the Chef repo itself are best managed within a version control system in the same way that you would manage code. Since we cloned the repo above, a git repo has already been initialized.

To set your workstation up for new commits, you should do a few things.

First, set the name and email that git will use to tag any commits you make. This is a requirement for gitto accept commits. We set this globally so that any git repo we create will use these values:

git config --global user.name "Your Name"

git config --global user.email "username@domain.com"

Next, we will tell git to ignore any information contained within the ~/chef-repo/.chef directory. We will create this directory in a few minutes to store some sensitive information. For now, we can add this location to our .gitignore file so that git does not store data that should not be exposed to other people:

echo ".chef" >> ~/chef-repo/.gitignore

Since we have made a change to the .gitignore file, we can go ahead and make our first new commit to the version control system. First, add all of the modified files to the current staging area:

cd ~/chef-repo

git add .

Now, commit the changes. We will use the -m flag to specify an in-line commit message describing the changes we are making:

git commit -m "Excluding the ./.chef directory from version control"

Our Chef repo is now under version control. As we author configurations for our infrastructure, we can use the above two commands to keep our git repo up-to-date.

### Download and Install the Chef Development Kit

Next, we need to install the Chef Development Kit, a suite of software designed for Chef workstations. This includes many utilities that will be useful when designing configurations for your infrastructure. The tool we are interested in at this point is the bundled knife command, which can communicate with and control both the Chef server and any Chef clients.

We can find the Chef 12 Development Kit on the Chef website. Since we are using Ubuntu 14.04 as our workstation, the page [here](https://downloads.chef.io/chef-dk/ubuntu/#/) will contain the latest download link. Note that at the time of this writing, the download link only references Ubuntu 12.04 and Ubuntu 13.10, but it should still install without issue on Ubuntu 14.04.

Right-click on the download button under "Ubuntu Linux" and copy the link location:



Back on your workstation, change to your home directory. Paste the link you copied and use the wgetcommand to download the package. The link you copied may be different from the one below if a newer development kit version has been released:

cd ~

wget https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/x86\_64/chefdk\_0.4.0-1\_amd64.deb

Once the .deb package has been downloaded, you can install it by typing:

sudo dpkg -i chefdk\_\*.deb

After the installation, you can verify that all of the components are available in their expected location through the new chef command:

chef verify

If your workstation will primarily be used to manage Chef for your infrastructure, you will likely want to default to the version of Ruby installed with Chef. You can do this by modifying your .bash\_profile so that Chef's Ruby takes precedence:

echo 'eval "$(chef shell-init bash)"' >> ~/.bash\_profile

Afterwards, you can source your .bash\_profile file to set the correct environmental variables for the current session:

source ~/.bash\_profile

If you wish to manage your Ruby versions independently, you can skip the above steps.

### Download the Authentication Keys to the Workstation

At this point, your workstation has all of the software needed to interact with a Chef server and compose infrastructure configurations. However, it is not yet configured to interact with your Chef server and your environment. In this section, we'll download the credentials we created on the Chef server.

We will use the scp utility to download the user key and the organization validator key that we created on the Chef server. Before doing so, we will create the hidden directory where we will store these files:

mkdir ~/chef-repo/.chef

The method that you use to connect to the Chef server will determine how exactly we go about downloading the keys. Follow the method below that matches your setup:

#### How To Download Keys when Connecting to a Chef Server with Passwords

If you connect to your Chef server through SSH using password-based authentication, the scp command will work without significant modification.

On your workstation, specify the username and domain name or IP address used to connect to the Chef server. Follow this immediately with a colon (:) and the path to the file you wish to download. After adding a space, indicate the directory on the local computer where you wish the download the files to be placed (~/chef-repo/.chef in our case).

If you log into the Chef server using the root user account, your commands will look something like this. Remember to change both the domain name or IP address and the name of the key files you are trying to download to match your environment:

scp root@server\_domain\_or\_IP:/root/admin.pem ~/chef-repo/.chef

scp root@server\_domain\_or\_IP:/root/digitalocean-validator.pem ~/chef-repo/.chef

If you connect to your Chef server using a non-root user, the commands will look more like this:

scp username@server\_domain\_or\_IP:/home/username/admin.pem ~/chef-repo/.chef

scp username@server\_domain\_or\_IP:/home/username/digitalocean-validator.pem ~/chef-repo/.chef

#### How To Download Keys when Connecting to a Chef Server Using SSH Keys

If, instead, you connect to your Chef server using SSH keys (recommended), you will need to perform some additional steps.

First, leave your SSH session with the workstation. We will need to reconnect momentarily with a new parameter:

exit

Once you are back on your local computer, you will need to add the SSH keys you use to connect to the Chef server to an SSH agent. OpenSSH, the standard SSH suite, includes an SSH agent that can be started by typing:

eval $(ssh-agent)

You should see output that looks like this (the number will likely be different):

Agent pid 13881

Once the agent is started, you can add your SSH key to it:

ssh-add

Identity added: /home/demo/.ssh/id\_rsa (rsa w/o comment)

This will keep your SSH key stored in memory. Now, you can forward the stored key to your workstation as you connect by using the -A option with ssh. This will allow you to connect to any computer from your workstation as if you were connecting from your local computer:

ssh -A username@workstation\_domain\_or\_IP

Now, you can connect to your Chef server without needing a password using the forwarded SSH credentials. If the keys on your Chef server were available through the root user, the commands you will need will look similar to this. Remember to change the Chef server domain name or IP address and the key names as needed:

scp root@server\_domain\_or\_IP:/root/admin.pem ~/chef-repo/.chef

scp root@server\_domain\_or\_IP:/root/digitalocean-validator.pem ~/chef-repo/.chef

If the SSH key configured for the Chef server instead is used to authenticate you to a regular user account, your commands will look like this instead:

scp username@server\_domain\_or\_IP:/home/username/admin.pem ~/chef-repo/.chef

scp username@server\_domain\_or\_IP:/home/username/digitalocean-validator.pem ~/chef-repo/.chef

### Configuring Knife to Manage your Chef Environment

Now that you have your Chef credentials available on your workstation, we can configure the knifecommand with the information it needs to connect to and control your Chef infrastructure. This is done through a knife.rb file that we will place in the ~/chef-repo/.chef directory along with our keys.

Open up a file called knife.rb in that directory in your text editor:

nano ~/chef-repo/.chef/knife.rb

In this file, paste the following information:

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

log\_level :info

log\_location STDOUT

node\_name "name\_for\_workstation"

client\_key "#{current\_dir}/name\_of\_user\_key"

validation\_client\_name "organization\_validator\_name"

validation\_key "#{current\_dir}/organization\_validator\_key"

chef\_server\_url "https://server\_domain\_or\_IP/organizations/organization\_name"

syntax\_check\_cache\_path "#{ENV['HOME']}/.chef/syntaxcache"

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

The following items should be adjusted to suit your infrastructure:

* **node\_name**: This specifies the name that knife will use to connect to your Chef server. This should match your user name.
* **client\_key**: This should be the name and path to the user key that you copied over from the Chef server. We can use the #{current\_dir} snippet to fill in the path if the key is in the same directory as the knife.rb file.
* **validation\_client\_name**: This is the name of the validation client that knife will use to bootstrap new nodes. This will take the form of your organization short name, followed by -validator.
* **validation\_key**: Like the client\_key, this includes the name and path to the validation key you copied from the Chef server. Again, you can use the #{current\_dir} Ruby snippet to specify the current directory if the validation key is in the same directory as the knife.rb file.
* **chef\_server\_url**: This is the URL where the Chef server can be reached. It should begin withhttps://, followed by your Chef server's domain name or IP address. Afterwards, the path to your organization should be specified by appending /organizations/your\_organization\_name.

For our guide, the knife.rb file will look similar to this. You still need to adjust the server's domain name or IP address if you are following along:

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

log\_level :info

log\_location STDOUT

node\_name "admin"

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

validation\_client\_name "digitalocean-validator"

validation\_key "#{current\_dir}/digitalocean-validator.pem"

chef\_server\_url "https://server\_domain\_or\_IP/organizations/digitalocean"

syntax\_check\_cache\_path "#{ENV['HOME']}/.chef/syntaxcache"

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

When you are finished, save and close the knife.rb file.

Now, we will test the configuration file by trying out a simple knife command. We need to be in our~/chef-repo directory for our configuration file to be read correctly:

cd ~/chef-repo

knife client list

This first attempt should fail with an error that looks like this:

ERROR: SSL Validation failure connecting to host: server\_domain\_or\_IP - SSL\_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed

ERROR: Could not establish a secure connection to the server.

Use `knife ssl check` to troubleshoot your SSL configuration.

If your Chef Server uses a self-signed certificate, you can use

`knife ssl fetch` to make knife trust the server's certificates.

Original Exception: OpenSSL::SSL::SSLError: SSL\_connect returned=1 errno=0 state=SSLv3 read server certificate B: certificate verify failed

This occurs because we do not have our Chef server's SSL certificate on our workstation. We can acquire this by typing:

knife ssl fetch

This should add the Chef server's certificate file to a list in our ~/chef-repo/.chef directory:

WARNING: Certificates from server\_domain\_or\_IP will be fetched and placed in your trusted\_cert

directory (/home/demo/chef-repo/.chef/trusted\_certs).

Knife has no means to verify these are the correct certificates. You should

verify the authenticity of these certificates after downloading.

Adding certificate for server\_domain\_or\_IP in /home/demo/chef-repo/.chef/trusted\_certs/server\_domain\_or\_IP.crt

After the SSL certificate has been fetched, the previous command should now work:

knife client list

digitalocean-validator

If the above command correctly returns, your workstation is now set up to control your Chef environment.

## Bootstrapping a New Node with Knife

With our Chef server and workstation configured, we can begin using Chef to configure new servers within our infrastructure.

This happens through a process called "bootstrapping" in which the Chef client executable is installed on the new computer and the organizational validator key is passed along as well. The new node then contacts the Chef server with the validator key and, in return, receives its own unique client key and any configuration that has been assigned to it. This process gets the new server into its initial state and sets it up for any future management.

To connect to the new server, we will need a few pieces of information about the new node:

* The domain name or IP address where it can be reached
* The username used to complete administrative actions. This can be either root, or a user configured with sudo privileges.
* A method of logging in as the above user. This can be either the password, or the ability to use an SSH key.
* A method of performing administrative tasks. For root users, this is unnecessary. For users relying on sudo privileges, a password is generally necessary.

The general syntax of the command will be:

knife bootstrap node\_domain\_or\_IP [options]

Some common options you may end up using are:

* **-x**: Used to specify the username to authenticate with through SSH. This is usually required.
* **-N**: The new name for the node, as displayed within Chef. Leaving this out will usually result in the hostname being used for the Chef node name.
* **-P**: Used to specify the password for the username on the remote server. This is necessary if **either**the SSH session requires password authentication **or** if the username requires a password for sudocommands.
* **--sudo**: If the username on the remote server will need to use sudo to perform administrative actions, this flag is needed. By default, it will prompt for the sudo password.
* **--use-sudo-password**: If you are already providing the password for the user with the -P flag, using this flag **in addition** to the --sudo flag will use the -P password without prompting.
* **-A**: This option forwards SSH keys to the remote host to login rather than using password authentication.

When using the -A option, you must start an SSH agent on your local computer, add the SSH key that can be used to connect to the new node, and forward that information to your workstation by connecting with the -A flag initially. More information about how to do this can be found in the workstation configuration section regarding downloading the keys from the Chef server.

Using the above information, it is possible to construct the correct bootstrapping commands for a variety of situations.

For example, to bootstrap a node with the name "testing", using the username demo, which is configured with sudo privileges, and which needs a password for SSH and the sudo validation, we can type:

knife bootstrap node\_domain\_or\_IP -N testing -x demo -P password --sudo --use-sudo-password

If we want to bootstrap using the root user, with SSH key authentication using keys available on the workstation, and wish to keep use the node's hostname as the Chef node name, we can type:

knife bootstrap node\_domain\_or\_IP -x root -A

If we want to use SSH keys to authenticate to a sudo user, we will still need to provide a password using the -P flag, the --sudo flag, and the --use-sudo-password flag to avoid prompts:

knife bootstrap node\_domain\_or\_IP -x demo -A -P password --sudo --use-sudo-password -N name

If you are in the above scenario, but do not mind being promted for the sudo password, you can instead just type this:

knife bootstrap node\_domain\_or\_IP -x demo -A --sudo -N name

Once your new node is bootstrapped, you should have a new client:

knife client list

digitalocean-validator

name

You should also have a new node of the same name:

knife node list

name

You can use the above procedure to easily set up new Chef clients on any number of new servers.

If you want to learn about how to automatically add your new DigitalOcean Droplets to your existing Chef infrastructure without having to bootstrap each one, [check out this tutorial](https://www.digitalocean.com/community/tutorials/how-to-automatically-add-new-droplets-to-your-configuration-managment-system).

## Conclusion

After following this guide, you should have a fully functional Chef server configured for your infrastructure. We have also set up a workstation that can be used to manage and maintain the configurations that Chef will apply to your infrastructure. We have demonstrated how to use the knife command to bootstrap the servers that will be configured by Chef.

In the [next guide](https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu), we will demonstrate how to design configurations for your nodes using some Chef constructs. We will go over the fundamentals of Chef recipes and cookbooks as ways to control your infrastructure with declarative configs

How to Install a Chef Server, Workstation, and Client on Ubuntu VPS Instances

Posted Jan 30, 2014

Top of Form

Bottom of Form

8

 152.2kviews [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Git](https://www.digitalocean.com/community/tags/git?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

Tutorial Series

This tutorial is part 3 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances#tutorial_series_10)

Warning

This guide is targeted at Chef 11. The Chef 12 platform introduces some significant configuration differences. You can find a guide on how to set up a Chef 12 server, workstation, and node [here](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers).

**Introduction**

As your organizational structure grows and the separate components necessary to manage your environment expand, administering each server and service can become unmanageable.

Configuration management solutions are designed to simplify the management of systems and infrastructure. The goal of configuration management tools are to allow you to manage your infrastructure as a code base. **Chef** is a configuration management solution that allows you to manage large numbers of servers easily.

In a [previous guide](https://www.digitalocean.com/community/articles/how-to-understand-the-chef-configuration-environment-on-a-vps), we discussed the general structure of the Chef components and the way the system operates on a conceptual level. We went over some key terminology and the relationship between many different components.

In this guide, we will work to install a small Chef 11 setup. This will be one Chef server used to store configuration data and administer access rights. This will serve as a hub for our other machines.

We will also install a workstation that will allow us to interact with our server and build our configuration policies. This is where we will do the work to manage our infrastructure environment.

Finally, we will bootstrap a node, which will represent one of the servers in our organization that will be managed through Chef. We will do this using the server and workstation that we configured.

All three of these machines will be using Ubuntu 12.04 x86\_64 VPS instances for simplicity's sake. We will be targeting the Chef 11 release as it is stable and well tested.

Server Installation

The first component that we need to get online is the Chef server. Because this is central to the communication of our other components, it needs to be available for our other machines to complete their setup.

Before doing this, it is important to set up a domain name for your Chef server to resolve requests correctly. You can see our guide on getting a [domain name set up with DigitalOcean](https://digitalocean.com/community/articles/how-to-set-up-a-host-name-with-digitalocean) here.

If you do not have a domain name, you will need to edit the /etc/hosts file on each of the VPS instances that you will be using, so that they can all resolve the Chef server by name. If you *do* have a domain name, this should only be necessary on the VPS you will be using as the Chef server. You can do this by typing this on the VPS you will use as the Chef server:

sudo nano /etc/hosts

Inside, add the IP address of this computer and then the name you would like to use to connect to the server. You can then add a short name after that. Something like this:

111.222.333.444 chef.domain.com chef

Change the 111.222.333.444 to your Chef server's IP address and change the other two values to whatever you'd like to use to refer to your server as. Add this line to point to your Chef server to this file on each of the machines you plan to use if you are not using a domain name.

You can check that this is setup correctly by typing:

hostname -f

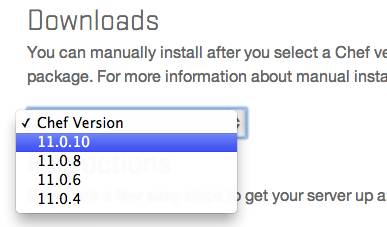
This should give you the name that is used to reach this server.

You can get the chef server package by visiting [this page](http://www.getchef.com/chef/install/) in your web browser.

Click on the "Chef Server" tab and then select the menus that match your operating system:



Select the most recent version of the Chef 11 server available to you on the right-hand side:



You will be presented with a link to a deb file. Right-click on this and select the option that is similar to "copy link location".

In the VPS instance that you will be using as the server, change to your user's home directory and use thewget utility to download the deb. At the time of this writing, the most recent link is this:

cd ~

wget https://opscode-omnibus-packages.s3.amazonaws.com/ubuntu/12.04/x86\_64/chef-server\_11.0.10-1.ubuntu.12.04\_amd64.deb

This will download the installation package that you can then install like this:

sudo dpkg -i chef-server\*

This will install the server component on this machine.

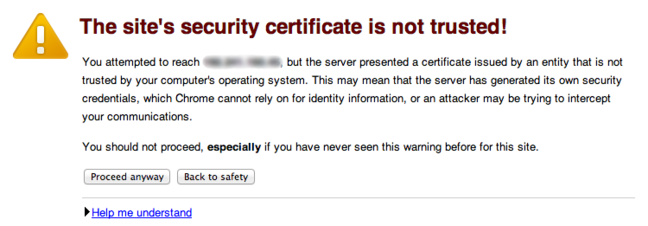
It prints to the screen afterwards that you should run this next command to actually configure the service around your specific machine. This will configure everything automatically:

sudo chef-server-ctl reconfigure

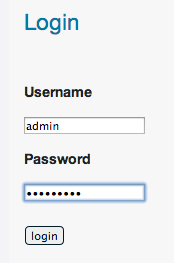
Once this step is complete, the server should be up and running. You can access the web interface immediately by typing https:// followed by your server's domain name or IP address.

https://server\_domain\_or\_IP

Because the SSL certificates were signed by an authority that your browser does not recognize by default, you will see a warning message appear:



Click the "Proceed anyway" button to bypass this screen and access the login screen. It will look something like this:

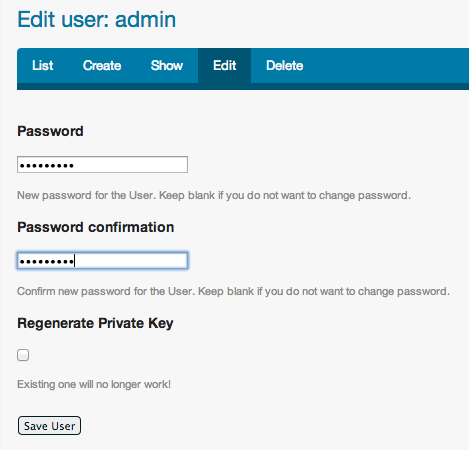


The default login credentials are as follows:

Default Username: admin

Default Password: p@ssw0rd1

When you log in for the first time, you will be immediately prompted to change your password. Select a new password and then click on the "Save User" button on the bottom:



You have now configured the server to a point where we can leave it and begin our workstation configuration.

Workstation Installation

Our workstation computer is the VPS that we will use to create and edit the actual policies that dictate our infrastructure environments. This machine has a copy of the Chef repo that describes our machines and services and it uploads those to the Chef server for implementation.

We will start by simply installing git for version control:

sudo apt-get update

sudo apt-get install git

This actually has two purposes. The obvious use is that we will be keeping our configuration under version control to track changes. The second purpose is to temporarily cache our password with sudo so that the following command works.

We will now download and run the client installation script from the Chef website. Type this command to complete all of these steps:

curl -L https://www.opscode.com/chef/install.sh | sudo bash

Our Chef workstation component is now installed. However it is very far from being configured.

The next step is to acquire the "chef-repo" directory structure for a properly formatted Chef repository from GitHub. We can clone the structure into our home directory by typing:

cd ~

git clone https://github.com/opscode/chef-repo.git

This will create a directory called chef-repo in your home directory. This is where the entire configuration for your setup will be contained.

We will create a configuration directory for the Chef tools themselves within this directory:

mkdir -p ~/chef-repo/.chef

Within this directory, we will need to put some of the authentication files from our Chef server. Specifically, we need two private keys.

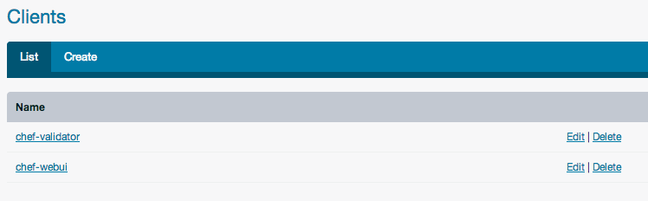
**Generating and Copying Keys from the Server**

Go back to your Chef server in your web browser:

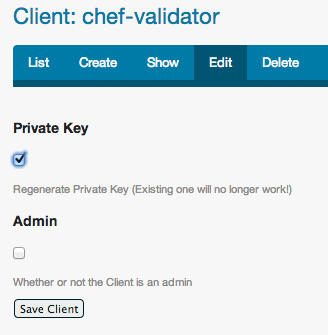
https://server\_domain\_or\_IP

Log in using the admin user's credentials that you changed before.

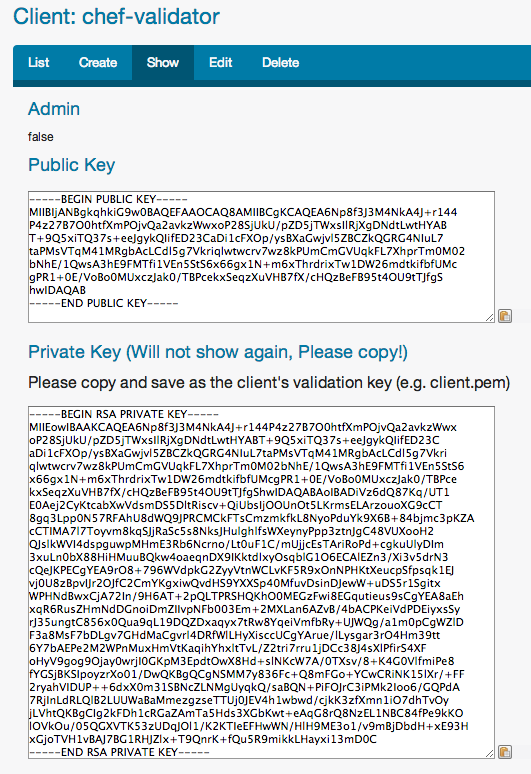
Click on the "Clients" tab in the top navigation bar. You will see two two clients called chef-validator and chef-webui:



Click on the "Edit" button associated with the chef-validator client. Regenerate the private key by selecting that box and clicking "Save Client":



You will be taken a screen with the newly generated values for the key file.



**Note:** This key will only be available once, so don't click out of this page! If you do, you will need to regenerate the key again.

Copy the value of the private key field (the one at the bottom).

On your workstation machine, change to the Chef configuration directory we created in the repo:

cd ~/chef-repo/.chef

Open a new file for the validator key we just created:

nano chef-validator.pem

In this file, paste the contents of the key you copied from the server's web interface (some lines have been removed for brevity here):

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA6Np8f3J3M4NkA4J+r144P4z27B7O0htfXmPOjvQa2avkzWwx

oP28SjUkU/pZD5jTWxsIlRjXgDNdtLwtHYABT+9Q5xiTQ37s+eeJgykQIifED23C

aDi1cFXOp/ysBXaGwjvl5ZBCZkQGRG4NIuL7taPMsVTqM41MRgbAcLCdl5g7Vkri

. . .

. . .

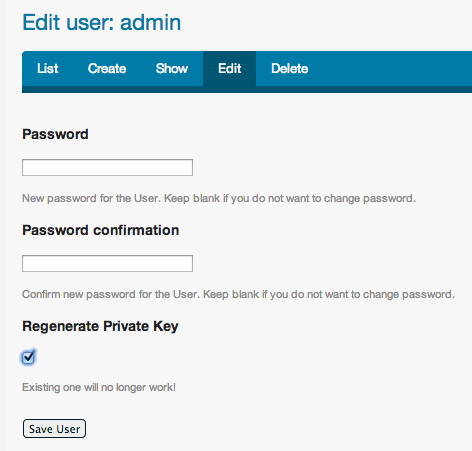
xGjoTVH1vBAJ7BG1RHJZlx+T9QnrK+fQu5R9mikkLHayxi13mD0C

-----END RSA PRIVATE KEY-----

Ensure that there are not extra blank lines above or below the key. Save and close the file.

We will follow the same procedure to regenerate and save the admin user's key file. This time, the key is for a user, so click on the "Users" tab on the top.

Again, click on the "Edit" button associated with the admin user, check the "Regenerate Private Key" box and click the "Save User" button:



Copy the Private key value on the next screen. Once again, **this will not be shown again**, so copy it correctly the first time.

Back on your workstation computer, you will need to create another file for the admin user in the same directory:

nano admin.pem

Paste the contents of the key you copied from the server's interface (again, this is shortened):

-----BEGIN RSA PRIVATE KEY-----

MIIEpAIBAAKCAQEA/apu0+F5bkVtX6qGYcfoA6sIW/aLFUEc3Bw7ltb50GoZnUPj

0Ms1N1Rv/pdVZXeBa8KsqICAhAzvwSr0H9j+AoURidbkLv4urVC9VS4dZyIRfwvq

PGvAKop9bbY2WJMs23SiEkurEDyfKaqXKW687taJ9AKbH2yVx0ArPI2RwS3Sze3g

. . .

. . .

VTkNpg3lLRSGbQkvRUP6Kt20erS2bfETTtH6ok/zW4db8B/vnBlcZg==

-----END RSA PRIVATE KEY-----

Verify that there are no extra lines above or below the pasted key lines. Save and close the file.

**Configure the Knife Command**

We now have to configure the knife command. This command is the central way of communicating with our server and the nodes that we will be configuring. We need to tell it how to authenticate and then generate a user to access the Chef server.

Luckily, we've been laying the groundwork for this step by acquiring the appropriate credential files. We can start the configuration by typing:

knife configure --initial

This will ask you a series of questions. We will go through them one by one:

WARNING: No knife configuration file found

Where should I put the config file? [/home/your\_user/.chef/knife.rb]

The values in the brackets ([]) are the default values that knife will use if we do not select a value.

We want to place our knife configuration file in the hidden directory we have been using:

/home/your\_user/chef-repo/.chef/knife.rb

In the next question, type in the domain name or IP address you use to access the Chef server. This should begin with https:// and end with :443:

https://server\_domain\_or\_IP:443

You will be asked for a name for the new user you will be creating. Choose something descriptive:

Please enter a name for the new user: [root] station1

It will then ask you for the admin name. This you can just press enter on to accept the default value (we didn't change the admin name).

It will then ask you for the location of the existing administrators key. This should be:

/home/your\_user/chef-repo/.chef/admin.pem

It will ask a similar set of questions about the validator. We haven't changed the validator's name either, so we can keep that as chef-validator. Press enter to accept this value.

It will then ask you for the location of the validation key. It should be something like this:

/home/your\_user/chef-repo/.chef/chef-validator.pem

Next, it will ask for the path to the repository. This is the chef-repo folder we have been operating in:

/home/your\_user/chef-repo

Finally, it will ask you to select a password for your new user. Select anything you would like.

This should complete our knife configuration. If we look in our chef-repo/.chef directory, we should see a knife configuration file and the credentials of our new user:

ls ~/chef-repo/.chef

admin.pem chef-validator.pem knife.rb station1.pem

**Cleaning up and Testing the Workstation**

Our configuration for our workstation is almost complete. We need to do a few things to clean up and verify that our connections work.

First, we should get our Chef repository under version control. Because Chef configuration operates as source code, we can handle it in the same way as we would with the files for any program.

First, we need to initialize our git name and email. Type:

git config --global user.email "your\_email@domain.com"

git config --global user.name "Your Name"

Since our "chef-repo" directory structure was pulled straight from GitHub, it is under git version control already.

However, we do not want to include the "chef-repo/.chef" directory in this version control. This contains our private keys and the knife configuration file. They do not have anything to do with our infrastructure we want to design.

Add this directory to the ignore list by opening the .gitignore file:

nano ~/chef-repo/.gitignore

At the bottom of the file, type .chef to include the entire directory:

.rake\_test\_cache

###

# Ignore Chef key files and secrets

###

.chef/\*.pem

.chef/encrypted\_data\_bag\_secret

.chef

Save and close the file.

Now, we can commit our current state (which probably won't have any changes beside the .gitignorefile we just modified) by typing:

git add .

git commit -m 'Finish configuring station1'

We also want to make sure that our user uses the version of Ruby packaged with our Chef installation. Otherwise, calls made by Chef could be interpreted by the system's Ruby installation, which may be incompatible with the rest of our tools.

We can just modify our path by adding a line to the bottom of our .bash\_profile file.

Type this in to add the line:

echo 'export PATH="/opt/chef/embedded/bin:$PATH"' >> ~/.bash\_profile

Now, we can implement these changes into our current environment by typing:

source ~/.bash\_profile

We can test whether we can connect successfully with the Chef server by requesting some information from the server using the knife command.

This will return a list of all of our users:

knife user list

admin

station1

If this is successful, then our workstation can successfully communicate with our server.

Bootstrapping a Client Node

Now that we have the Chef server and a workstation online, we can try to bootstrap a Chef client on a sample node. We will use another Ubuntu instance.

The bootstrapping process involves setting up Chef client on a node. Chef client is a piece of software that communicates with the server in order to receive directions for its own configuration. The client then brings the node it is installed on in-line with the policy given to it by the server.

This process will simply configure our new VPS instance to be under the umbrella of our Chef management system. We can then configure it however we would like by creating policies on our workstation and uploading them to our server.

To complete this process, we only need to know three pieces of information about the VPS we want to install the client software on:

* IP address or domain name
* Username (accessible through SSH and with sudo privileges)
* Password

With these pieces of information, we can install the appropriate packages by using our knife tool on our workstation.

You want to type a command that looks like this:

knife bootstrap node\_domain\_or\_IP -x username -P password -N name\_for\_node --sudo

Let's break this down a bit. The domain name/IP address tells knife which server to connect to. The username and password provide the login credentials.

If the user you are using is not root, then the --sudo option is necessary in order for the bootstrapping process to successfully install software on the remote computer. It will prompt you for the password once you log in to use the sudo command.

The name for the node is a name that you select that is used internally by Chef. This is how you will refer to this machine when crafting policies and using knife.

After the command is run, the client software will be installed on the remote node. It will be configured to communicate with the Chef server to receive instructions.

We can query our list of clients by typing:

knife client list

chef-validator

chef-webui

client1

We can see the two clients that are configured by default during the Chef server installation (chef-validator and chef-webui), as well as the client we just created.

You can just as easily set up other nodes to bring them under configuration control of your Chef system.

Conclusion

You should now have a Chef server, a separate workstation to create your configurations, and an example node.

We have not done any actual configuration of the node through Chef at this point, but we are set up to begin this process. In future tutorials, we will discuss how to implement policies and create recipes and cookbooks to manage your nodes.

How To Create Simple Chef Cookbooks to Manage Infrastructure on Ubuntu

Posted Feb 3, 2014

Top of Form

Bottom of Form

16

 152.2kviews [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Nginx](https://www.digitalocean.com/community/tags/nginx?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

Tutorial Series

This tutorial is part 4 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu#tutorial_series_10)

**Introduction**

Chef is a configuration management system designed to allow you to automate and control vast numbers of computers in an automated, reliable, and scalable manner.

In previous tutorials, we have looked at some [common Chef terminology](https://www.digitalocean.com/community/articles/how-to-understand-the-chef-configuration-environment-on-a-vps) and discussed how to install a Chef server, workstation, and nodes (with [Chef 12](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers) or [Chef 11](https://www.digitalocean.com/community/articles/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances)). In this guide, we will use these guides as a jumping off point to begin talking about how to automate your environment.

In this article, we will discuss the basics of creating a Chef cookbook. Cookbooks are the configuration units that allow us to configure and perform specific tasks within Chef on our remote nodes. We build cookbooks and then tell Chef which nodes we want to run the steps outlined in the cookbook.

In this guide, we will assume that you are starting with the three machines that we ended the [last lesson](https://www.digitalocean.com/community/articles/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances)with. You should have a server, a workstation, and at least one node to push configuration changes to.

Basic Cookbook Concepts

Cookbooks serve as the fundamental unit of configuration and policy details that Chef uses to bring a node into a specific state. This just means that Chef uses cookbooks to perform work and make sure things are as they should be on the node.

Cookbooks are usually used to handle one specific service, application, or functionality. For instance, a cookbook can be created to use NTP to set and sync the node's time with a specific server. It may install and configure a database application. Cookbooks are basically packages for infrastructure choices.

Cookbooks are created on the workstation and then uploaded to a Chef server. From there, recipes and policies described within the cookbook can be assigned to nodes as part of the node's "run-list". A run-list is a sequential list of recipes and roles that are run on a node by chef-client in order to bring the node into compliance with the policy you set for it.

In this way, the configuration details that you write in your cookbook are applied to the nodes you want to adhere to the scenario described in the cookbook.

Cookbooks are organized in a directory structure that is completely self-contained. There are many different directories and files that are used for different purposes. Let's go over some of the more important ones now.

**Recipes**

A recipe is the main workhorse of the cookbook. A cookbook can contain more than one recipe, or depend on outside recipes. Recipes are used to declare the state of different resources.

Chef resources describe a part of the system and its desired state. For instance, a resource could say "the package x should be installed". Another resource may say "the x service should be running".

A recipe is a list related resources that tell Chef how the system should look if it implements the recipe. When Chef runs the recipe, it checks each resource for compliance to the declared state. If the system matches, it moves on to the next resource, otherwise, it attempts to move the resource into the given state.

Resources can be of many different types. You can learn about the [different resource types](http://docs.opscode.com/resource.html) here. Some common ones are:

* **package**: Used to manage packages on a node
* **service**: Used to manage services on a node
* **user**: Manage users on the node
* **group**: Manage groups
* **template**: Manage files with embedded ruby templates
* **cookbook\_file**: Transfer files from the files subdirectory in the cookbook to a location on the node
* **file**: Manage contents of a file on node
* **directory**: Manage directories on node
* **execute**: Execute a command on the node
* **cron**: Edit an existing cron file on the node

**Attributes**

Attributes in Chef are basically settings. Think of them as simple key-value pairs for anything you might want to use in your cookbook.

There are several different kinds of attributes that can be applied, each with a different level of precedence over the final settings that a node operates under. At the cookbook level, we generally define the default attributes of the service or system we are configuring. These can be overridden later by more specific values for a specific node.

When creating a cookbook, we can set attributes for our service in the attributes subdirectory of our cookbook. We can then reference these values in other parts of our cookbook.

**Files**

The files subdirectory within the cookbook contains any static files that we will be placing on the nodes that use the cookbook.

For instance, any simple configuration files that we are not likely to modify can be placed, in their entirety, in the files subdirectory. A recipe can then declare a resource that moves the files from that directory into their final location on the node.

**Templates**

Templates are similar to files, but they are not static. Template files end with the .erb extension, meaning that they contain embedded Ruby.

These are mainly used to substitute attribute values into the file to create the final file version that will be placed on the node.

For example, if we have an attribute that defines the default port for a service, the template file can call to insert the attribute at the point in the file where the port is declared. Using this technique, you can easily create configuration files, while keeping the actual variables that you wish to change elsewhere.

**Metadata.rb**

The metadata.rb file is used, not surprisingly, to manage the metadata about a package. This includes things like the name of the package, a description, etc.

It also includes things like dependency information, where you can specify which cookbooks this cookbook needs to operate. This will allow the Chef server to build the run-list for the nodes correctly and ensure that all of the pieces are transfered correctly.

Create a Simple Cookbook

To demonstrate some of the work flow involved in working with cookbooks, we will create a cookbook of our own. This will be a very simple cookbook that installs and configures the Nginx web server on our node.

To begin, we need to go to our ~/chef-repo directory on our workstation:

cd ~/chef-repo

Once there, we can create a cookbook by using knife. As we mentioned in previous guides, knife is a tool used to configure most interactions with the Chef system. We can use it to perform work on our workstation and also to connect with the Chef server or individual nodes.

The general syntax for creating a cookbook is:

knife cookbook create cookbook\_name

Since our cookbook will deal with installing and configuring Nginx, we will name our cookbook appropriately:

knife cookbook create nginx

\*\* Creating cookbook nginx

\*\* Creating README for cookbook: nginx

\*\* Creating CHANGELOG for cookbook: nginx

\*\* Creating metadata for cookbook: nginx

What knife does here is builds a simple structure within our cookbooks directory for our new cookbook. We can see our cookbook structure by navigating into the cookbooks directory, and into the directory with the cookbook name.

cd cookbooks/nginx

ls

attributes CHANGELOG.md definitions files libraries metadata.rb providers README.md recipes resources templates

As you can see, this has created a folder and file structure that we can use to build our cookbook. Let's begin with the biggest chunk of the configuration, the recipe.

**Create a Simple Recipe**

If we go into the recipes subdirectory, we can see that there is already a file called default.rb inside:

cd recipes

ls

default.rb

This is the recipe that will be run if you reference the "nginx" recipe. This is where we will be adding our code.

Open the file with your text editor:

nano default.rb

#

# Cookbook Name:: nginx

# Recipe:: default

#

# Copyright 2014, YOUR\_COMPANY\_NAME

#

# All rights reserved - Do Not Redistribute

#

The only thing that is in this file currently is a comment header.

We can begin by planning the things that need to happen for our Nginx web server to get up and running the way that we want it to. We do this by configuring "resources". Resources do not describe *how* to do something; they simply describe what a part of the system should look like when it is complete.

First of all, we obviously need to make sure the software is installed. We can do this by creating a "package" resource first.

package 'nginx' do

action :install

end

This little piece of code defines a package resource for Nginx. The first line begins with the type of resource (package) and the name of the resource ('nginx'). The rest is a group of actions and parameters that declare what we want to happen with the resource.

In this resource, we see action :install. This line tells Chef that the resource we are describing should be installed. The node that runs this recipe will check that Nginx is installed. If it is, it will check that off the list of things to do. If not, it will install the program using the methods available on the client system and then check it off.

After we install the service, we probably want to adjust its current state on the node. By default, Ubuntu does not start Nginx after installation, so we will want to change that:

service 'nginx' do

action [ :enable, :start ]

end

Here, we see a resource of the "service" type. This declares that for the Nginx service component (the part that allows us to manage the server with init or upstart), we want to start the service right now, and also enable it to start automatically when the machine is restarted.

The final resource we will be declaring is the actual file that we will be serving. Since this is just a simple file that we will not be modifying, we can simply declare the location where we want the file and tell it where in the cookbook to get the file:

cookbook\_file "/usr/share/nginx/www/index.html" do

source "index.html"

mode "0644"

end

We use the "cookbook\_file" resource type to tell Chef that this file is available within the cookbook itself and can be transfered as-is to the location. In our example, we are transferring a file into Nginx's document root.

In our case, we specify the file name that we are trying to create in the first line. In the "source" line, we tell it the name of the file to look for within the cookbook. Chef looks for this file within the "files/default" subdirectory in the cookbook.

The "mode" line sets the permissions on the file we are creating. In this case, we are allowing the root user read and write permissions and everyone else read permissions.

Save and close this file when you are finished.

**Creating the Index file**

As you saw above, we defined a "cookbook\_file" resource which should move a file called "index.html" into the document root on the node. We need to create this file.

We should put this file in the "files/default" subdirectory of our cookbook. Go there now by typing:

cd ~/chef-repo/cookbooks/nginx/files/default

Inside this directory, we will create the file we referenced:

nano index.html

This file will just be a really simple HTML document meant to demonstrate that our resources have operated the way we wanted them to.

Paste this into the file:

<html>

<head>

<title>Hello there</title>

</head>

<body>

<h1>This is a test</h1>

<p>Please work!</p>

</body>

</html>

Save and close the file when you are finished.

**Create a Helper Cookbook**

Before we go any further, let's preemptively solve a small problem. When our node tries to run the cookbook that we've created as it is now, chances are, it will fail.

That is because it will attempt to install Nginx from the Ubuntu repositories, and the package database on our node is most likely out-of-date. Usually, we run "sudo apt-get update" prior to running package commands.

To address this issue, we can create a simple cookbook whose only purpose is to ensure that the package database is updated.

We can do this using the same knife syntax we used before. Let's call this cookbook "apt":

knife cookbook create apt

This will create the same kind of directory structure that we had when we first started with our Nginx cookbook.

Let's cut straight to the chase and edit the default recipe for our new cookbook.

nano ~/chef-repo/cookbooks/apt/recipes/default.rb

In this file, we will declare an "execute" resource. This is simply a way of defining a command that we want to run on the node.

Our resource looks like this:

execute "apt-get update" do

command "apt-get update"

end

The first line gives a name for our resource. In our case, we are calling the resource this for simplicity's sake. If the "command" attribute is defined (as we have done), then this is the actual command that is executed.

Since these are exactly the same, it does not matter in the slightest.

Save and close the file.

Now that we have our new cookbook, there are a number of ways that we can make sure that we execute this before our Nginx cookbook. We could add it to the node's run-list before the Nginx cookbook, but we can also tie it into the Nginx cookbook itself.

This is probably the better option because we will not have to remember to add the "apt" cookbook before the "nginx" cookbook on every node we want to configure for Nginx.

We need to adjust a few things in the Nginx cookbook to make this happen. First, let's open the Nginx recipe file again:

nano ~/chef-repo/cookbooks/nginx/recipes/default.rb

At the top of this cookbook, before the other resources that we have defined, we can read in the "apt" default recipe by typing:

include\_recipe "apt"

package 'nginx' do

action :install

end

service 'nginx' do

action [ :enable, :start ]

end

cookbook\_file "/usr/share/nginx/www/index.html" do

source "index.html"

mode "0644"

end

Save and close the file.

The other file that we need to edit is the metadata.rb file. This file is checked when the Chef server sends the run-list to the node, to see which other recipes should be added to the run-list.

Open the file now:

nano ~/chef-repo/cookbooks/nginx/metadata.rb

At the bottom of the file, you can add this line:

name 'nginx'

maintainer 'YOUR\_COMPANY\_NAME'

maintainer\_email 'YOUR\_EMAIL'

license 'All rights reserved'

description 'Installs/Configures nginx'

long\_description IO.read(File.join(File.dirname(\_\_FILE\_\_), 'README.md'))

version '0.1.0'

depends "apt"

With that finished, our Nginx cookbook now relies on our apt cookbook to take care of the package database update.

Add the Cookbook to your Node

Now that our basic cookbooks are complete, we can upload them to our chef server.

We can do that individually by typing:

knife cookbook upload apt

knife cookbook upload nginx

Or, we can upload everything by typing:

knife cookbook upload -a

Either way, our recipes will be uploaded to the Chef server.

Now, we can modify the run-list of our nodes. We can do this easily by typing:

knife node edit name\_of\_node

If you need to find the name of your available nodes, you can type:

knife node list

client1

For our purposes, when we type this, we get a file that looks like this:

knife node edit client1

{

"name": "client1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

]

}

You may need to set your EDITOR environmental variable before this works. You can do this by typing:

export EDITOR=name\_of\_editor

As you can see, this is a simple JSON document that describes some aspects of our node. We can see a "run\_list" array, which is currently empty.

We can add our Nginx cookbook to that array using the format:

"recipe[name\_of\_recipe]"

When we are finished, our file should look like this:

{

"name": "client1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

"recipe[nginx]"

]

}

Save and close the file to implement the new settings.

Now, we can SSH into our node and run the Chef client software. This will cause the client to check into the Chef server. Once it does this, it will see the new run-list that has been assigned it.

SSH into your node and then run this:

sudo chef-client

Starting Chef Client, version 11.8.2

resolving cookbooks for run list: ["nginx"]

Synchronizing Cookbooks:

- apt

- nginx

Compiling Cookbooks...

Converging 4 resources

Recipe: apt::default

\* execute[apt-get update] action run

- execute apt-get update

Recipe: nginx::default

\* package[nginx] action install (up to date)

\* service[nginx] action enable

- enable service service[nginx]

\* service[nginx] action start (up to date)

\* cookbook\_file[/usr/share/nginx/www/index.html] action create (up to date)

Chef Client finished, 2 resources updated

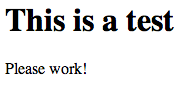
As you can see, our apt cookbook was sent over and run as well, even though it wasn't in the run-list we created. That is because Chef intelligently resolved dependencies and modified the actual run-list before executing it on the node.

**Note**: There are various methods of ensuring that one cookbook or recipe is run before another. Adding a dependency is only one choice, and other methods may be preferred.

We can verify that this works by going to our node's IP address or domain name:

http://node\_domain\_or\_IP

You should see something that looks like this:



Congratulations, you have configured your first node using Chef cookbooks!

Conclusion

Although this was a very simple example that probably didn't save you much time over configuring your server manually, hopefully you can begin to see the possibilities of this method of building infrastructure.

Not only does it allow for rapid deployment and configuration of different kinds of servers, it ensures that you know the exact configuration of all of your machines. This lets you validate and test your infrastructure, and also gives you the framework you need to quickly redeploy your infrastructure on a whim.

# How To Use Roles and Environments in Chef to Control Server Configurations

Posted Feb 4, 2014

Top of Form

Bottom of Form

10

 103.2kviews [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials)

## Tutorial Series

This tutorial is part 5 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-use-roles-and-environments-in-chef-to-control-server-configurations#tutorial_series_10)

### Introduction

As you build out your infrastructure, managing your many servers, services, users, and applications can become unwieldy very quickly. Configuration management systems can be used to help you manage this confusion.

**Chef** is an excellent configuration management system that can allow you to configure different components of your overall system very easily. In previous guides, we discussed [Chef terminology](https://www.digitalocean.com/community/articles/how-to-understand-the-chef-configuration-environment-on-a-vps), how to install a Chef server, workstation, and node (with [Chef 12](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers) or [Chef 11](https://www.digitalocean.com/community/articles/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances)), and [how to create simple cookbooks to manage configuration](https://www.digitalocean.com/community/articles/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu).

In this guide, we will continue to explore how you can manage your environment with Chef. This time, we will talk about how to use roles and environments to differentiate your servers and services based on what kind of functionality they should exhibit.

We will assume that you have installed your server, workstation, and client and that you have the cookbooks we created in our last guide available.

## Roles and Environments

### What is a Role?

In your organization, if your infrastructure grows to meet the demands of higher traffic, there are likely to be multiple, redundant servers that all perform the same basic tasks. For instance, these might be web servers that a load balancer passes requests to. They would all have the same basic configuration and could be said to each satisfy the same "role".

Chef's view of **roles** is almost entirely the same as the regular definition. A role in Chef is a categorization that describes what a specific machine is supposed to do. What responsibilities does it have and what software and settings should be given to it.

In different situations, you may have certain machines handling more than one role. For instance, if you are testing your software, one server may include the database and web server components, while in production, you plan on having these on separate servers.

With Chef, this can be as easy as assigning the first server to both roles and then assigning each role to separate computers for your production machines. Each role will contain the configuration details necessary to bring the machine to a fully operational state to fulfill its specific role. This means you can gather cookbooks that will handle package installations, service configuration, special attributes for that role, etc.

### What is an Environment?

Related to the idea of a role is the concept of Chef **environments**. An environment is simply a designation meant to help an administrator know what stage of the production process a server is a part of. Each server can be part of exactly one environment.

By default, an environment called "\_default" is created. Each node will be placed into this environment unless another environment is specified. Environments can be created to tag a server as part of a process group.

For instance, one environment may be called "testing" and another may be called "production". Since you don't want any code that is still in testing on your production machines, each machine can only be in one environment. You can then have one configuration for machines in your testing environment, and a completely different configuration for computers in production.

In the above example given in roles, you could specify that in your testing environment, the web and database server roles will be on a single machine. In your production environment, these roles should be tackled by individual servers.

Environments also help with the testing process itself. You can specify that in production, a cookbook should be a stable version. However, you can specify that if a machine is part of the testing environment, it can receive a more recent version of the cookbook.

## How To Use Roles

### Create a Role Using the Ruby DSL

We can create roles using the roles directory in our chef-repo directory on our workstation.

Log into your workstation and move into this directory now:

cd ~/chef-repo/roles

Within this directory, we can create different files that define the roles we want in our organization. Each role file can be written either in Chef's Ruby DSL, or in JSON.

Let's create a role for our web server:

nano web\_server.rb

Inside of this file, we can begin by specifying some basic data about the role:

name "web\_server"

description "A role to configure our front-line web servers"

These should be fairly straight forward. The name that we give cannot contain spaces and should generally match the file name we selected for this role, minus the extension. The description is just a human-readable message about what the role is supposed to manage.

Next, we can specify the runlist that we wish to use for this specific role. The runlist of a role can contain cookbooks (which will run the default recipe), recipes from cookbooks (as specified using the cookbook::recipe syntax), and other roles. Remember, a run\_list is always executed sequentially, so put the dependency items before the other items.

If we wanted to specify that the run\_list should be exactly what we configured in the last guide, we would have something that looked like this:

name "web\_server"

description "A role to configure our front-line web servers"

run\_list "recipe[apt]", "recipe[nginx]"

We can also use environment-specific run\_lists to specify variable configuration changes depending on which environment a server belongs to.

For instance, if a node is in the "production" environment, you could want to run a special recipe in your "nginx" cookbook to bring that server up to production policy requirements. You could also have a recipe in the nginx cookbook meant to configure special changes for testing servers.

Assuming that these two recipes are called "configprod" and "configtest" respectively, we could create some environmental specific run lists like this:

name "web\_server"

description "A role to configure our front-line web servers"

run\_list "recipe[apt]", "recipe[nginx]"

env\_run\_lists "production" => ["recipe[nginx::config\_prod]"], "testing" => ["recipe[nginx::config\_test]"]

In the above example, we have specified that if the node is part of the production environment, it should run the "configprod" recipe within the "nginx" cookbook. However, if the node is in the testing environment, it will run the "configtest" recipe. If a node is in a different environment, then the default run\_list will be applied.

Similarly, we can specify default and override attributes. You should be familiar with default attributes at this point. In our role, we can set default attributes which can override any of the default attributes set anywhere else.

We can also set override attributes, which have a higher precedence than many other attribute declarations. We can use this to try to force nodes that are assigned this role to behave in a certain way.

In our file, these could be added like this:

name "web\_server"

description "A role to configure our front-line web servers"

run\_list "recipe[apt]", "recipe[nginx]"

env\_run\_lists "production" => ["recipe[nginx::config\_prod]"], "testing" => ["recipe[nginx::config\_test]"]

default\_attributes "nginx" => { "log\_location" => "/var/log/nginx.log" }

override\_attributes "nginx" => { "gzip" => "on" }

Here we have set a default log location for any servers in the node. We have also specified that despite what some other attribute declarations have stated in other locations, that nodes in this role should have the gzip attribute set to "on". This can be overridden in a few more places, but is generally a high precedence declaration.

### Create a Role Using JSON

The other format that you can use to configure roles is JSON. In fact, we can explore this format by using knife to automatically create a role in this format. Let's create a test role:

knife role create test

Your text editor will be opened with a template role file preloaded. It should look something like this:

{

"name": "test",

"description": "",

"json\_class": "Chef::Role",

"default\_attributes": {

},

"override\_attributes": {

},

"chef\_type": "role",

"run\_list": [

],

"env\_run\_lists": {

}

}

This is basically the same information that we entered into the Ruby DSL-formatted file. The only differences are the formatting and the addition of two new keys called json\_class and chef\_type. These are used internally and should not be modified.

Other than that, we can easily recreate our other file in JSON with something like:

{

"name": "web\_server",

"description": "A role to configure our front-line web servers",

"json\_class": "Chef::Role",

"default\_attributes": {

"nginx": {

"log\_location": "/var/log/nginx.log"

}

},

"override\_attributes": {

"nginx": {

"gzip": "on"

}

},

"chef\_type": "role",

"run\_list": [

"recipe[apt]",

"recipe[nginx]"

],

"env\_run\_lists": {

"production": [

"recipe[nginx::config\_prod]"

],

"testing": [

"recipe[nginx::config\_test]"

]

}

}

This should have pretty much the same functionality as the Ruby version above.

### Transferring Roles between the Workstation and Server

When we save a JSON file created using the knife command, the role is created on the Chef server. In contrast, our Ruby file that we created locally is not uploaded to the server.

We can upload the ruby file to the server by running a command that looks like this:

knife role from file path/to/role/file

This will upload our role information specified in our file to the server. This would work with either the Ruby DSL formatted file or a JSON file.

In a similar vein, if we want to get our JSON file from the server, we can tell the knife command to show that role file in JSON and then pipe that into a file like this:

knife role show web\_server -Fjson > path/to/save/to

### Assigning Roles to Nodes

So now, regardless of the format we used, we have our role on the Chef server. How do we assign a node a certain role?

We assign a role to a node just as we would a recipe, in the node's run\_list.

So to add our role to a node, we would find the node by issuing:

knife node list

And then we would give a command like:

knife node edit node\_name

This will bring up the node's definition file, which will allow us to add a role to its run\_list:

{

"name": "client1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

"recipe[nginx]"

]

}

For instance, we can replace our recipe with our role in this file:

{

"name": "client1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

"role[web\_server]"

]

}

This will perform the same steps as our previous recipes, but instead it simply speaks to the role that the server should have.

This allows you to access all servers in a specific role by search. For instance, you could search for all of the database servers in your production environment by searching a role and environment:

knife search "role:database\_server AND chef\_environment:prod" -a name

This will give you a list of the nodes that are configured as a database server. You could use this internally in your cookbooks to configure a web server to automatically add all of the production database servers into its pool to make read requests from.

## How To Use Environments

### Creating an Environment

In some ways, environments are fairly similar to roles. They are also used to differentiate different servers, but instead of differentiating by the function of the server, environments differentiate by the phase of development that a machine belongs to.

We discussed some of this earlier when talking about roles. Environments that coincide with your actual product life-cycle make the most sense. If you run your code through testing, staging, and production, you should have environments to match.

As with roles, we can set up the definition files either in the Ruby DSL or in JSON.

In our "chef-repo" directory on our workstation, we should have an environments directory. This is where we should put our environment files.

cd ~/chef-repo/environments

Within this directory, if we were going to define an environment for development, we could make a file like this:

nano development.rb

name "development"

description "The master development branch"

cookbook\_versions({

"nginx" => "<= 1.1.0",

"apt" => "= 0.0.1"

})

override\_attributes ({

"nginx" => {

"listen" => [ "80", "443" ]

},

"mysql" => {

"root\_pass" => "root"

}

})

As you can see, one of the major advantages of incorporating environments into your system is that you can specify version constraints for the cookbooks, and recipes that are deployed.

We could also use the JSON format. The knife tool can generate the template of an environment file by typing:

knife environment create development

This will open our editor (again, you can set your editor with export EDITOR=nano) with a preloaded environment file with the name filled in.

We could create the same file by typing in:

{

"name": "development",

"description": "The master development branch",

"cookbook\_versions": {

"nginx": "<= 1.1.0",

"apt": "= 0.0.1"

},

"json\_class": "Cheff:Environment",

"chef\_type": "environment",

"default\_attributes": {

},

"override\_attributes": {

"nginx": {

"listen": [

"80",

"443"

]

},

"mysql": {

"root\_pass": "root"

}

}

}

This file should be functionally the same as the Ruby file we demonstrated above. As with the JSON role files, the environment JSON files have two extra pieces of information (json\_class and chef\_type) which should be left alone.

### Moving Environment Files to and from the Server

At this point, if you used the Ruby DSL, your file is on the workstation and if you used JSON, your file is only on the server. We can easily move files back and forth through knife.

We could upload our Ruby file to the Chef server by typing this:

knife environment from file ~/chef-repo/environments/development.rb

For our JSON file, we can get the environment file off of the server by typing something like:

knife environment show development -Fjson > ~/chef-repo/environments/development.json

This will display the JSON file from the server and pipe the results into a local file within the environments subdirectory.

### Setting Environments in Nodes

Each node can be in exactly one environment. We can specify the environment that a node belongs to by editing its node information.

For instance, to edit a node called client1, we could type this:

knife node edit client1

This will open up a JSON formatted file with the current node parameters:

{

"name": "client1",

"chef\_environment": "\_default",

"normal": {

"tags": [

]

},

"run\_list": [

"role[web\_server]"

]

}

As you can see, the chef\_environment is set to \_default originally. We can simply modify that value to put the node into a new environment.

When you are done, save and close the file. On the next chef-client run on the node, it will pick up the new attributes and version constraints and modify itself to align with the new policy.

## Conclusion

By now, you should have a good understanding of different ways you can work with roles and environments to solidify the state that your machines should be in. Using these categorization strategies, you can begin to manage the way that Chef treats servers in different contexts.

# How To Use the DigitalOcean Plugin for Knife to Manage Droplets in Chef

Posted Feb 9, 2014

Top of Form

Bottom of Form

4

 15.1kviews [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [DigitalOcean](https://www.digitalocean.com/community/tags/digitalocean?type=tutorials)

## Tutorial Series

This tutorial is part 6 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-plugin-for-knife-to-manage-droplets-in-chef#tutorial_series_10)

### Introduction

Chef is a configuration management system that allows you to build and manage your infrastructure in a controlled and repeatable way. With Chef, you can ensure that your infrastructure decisions are all centrally documented and can be reimplemented easily at any time.

In previous articles, we have discussed [basic Chef terminology](https://www.digitalocean.com/community/articles/how-to-understand-the-chef-configuration-environment-on-a-vps), how to install the software(with [Chef 12](https://www.digitalocean.com/community/tutorials/how-to-set-up-a-chef-12-configuration-management-system-on-ubuntu-14-04-servers) or[Chef 11](https://www.digitalocean.com/community/articles/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances)), [how to work with cookbooks](https://www.digitalocean.com/community/articles/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu), and [how to work with roles and environments](https://www.digitalocean.com/community/articles/how-to-use-roles-and-environments-in-chef-to-control-server-configurations).

In this article, we will discuss how to use a plugin for knife, the Chef configuration tool, to work with your DigitalOcean droplets. Using this tool, we can create infrastructure droplets and configure them easily from within our Chef system.

We will assume that you have installed and configured your server and workstation, and that you know how to create and bootstrap new nodes.

## Install the Knife DigitalOcean Plugin

Before we can begin working with DigitalOcean droplets from within our Chef system, we need to install the DigitalOcean plugin for the knife tool.

We can do this simply by installing a gem.

With the way our workstation is configured, if we were to type in:

gem install knife-digital\_ocean

We would get an error like this:

Fetching: knife-digital\_ocean-0.4.0.gem (100%)

ERROR: While executing gem ... (Gem::FilePermissionError)

You don't have write permissions into the /opt/chef/embedded/lib/ruby/gems/1.9.1 directory.

Similarly, if we run this command with sudo, we will get a command not found:

sudo gem install knife-digital\_ocean

sudo: gem: command not found

The problem is that we have specified the path to our Chef executables within our user's environment, but we do not have the privileges necessary to execute the command correctly.

We can get around this by using the full path to the Chef gem executable:

sudo /opt/chef/embedded/bin/gem install knife-digital\_ocean

This will install the DigitalOcean knife plugin, allowing you to take advantage of the included functionality.

## Configure the Knife Plugin with your DigitalOcean Credentials

In order to use the knife plugin, you need to give knife authorization to use your DigitalOcean account. This can be done using your account's API key and Client ID.

In your DigitalOcean control panel, click on the API link in the left-hand navigation menu. You should see the "Client ID" of your account here:



You will need this value for your configuration file in a moment.

If you have already generated an API key, you will need to collect that value from another application that is using it to connect to your DigitalOcean account. It is not possible to retrieve the current API value from the interface, as a security measure.

If you have not generated an API key yet, or if you have lost or do not need your old key anymore, you can generate a new key by typing the "Generate New Key" button:



Once you generate a new key, you will need to copy the value generated before leaving the screen. As mentioned above, you will **not** be able to access this value after leaving this page, so you will either have to generate a new value or retrieve your API key from an application that is already using it.

Once you have both the Client ID and the API key, you are ready to configure knife to use these values.

On your workstation, navigate to the Chef configuration directory within your Chef repo:

cd ~/chef-repo/.chef

We will be editing the knife.rb file located within:

nano knife.rb

log\_level :info

log\_location STDOUT

node\_name 'station1'

client\_key '/home/demo/chef-repo/.chef/station1.pem'

validation\_client\_name 'chef-validator'

validation\_key '/home/demo/chef-repo/.chef/chef-validator.pem'

chef\_server\_url 'https://chef\_server.com:443'

syntax\_check\_cache\_path '/home/demo/chef-repo/.chef/syntax\_check\_cache'

cookbook\_path [ '/home/demo/chef-repo/cookbooks' ]

At the bottom of the file, we need to add our credentials, like this:

log\_level :info

log\_location STDOUT

node\_name 'station1'

client\_key '/home/demo/chef-repo/.chef/station1.pem'

validation\_client\_name 'chef-validator'

validation\_key '/home/demo/chef-repo/.chef/chef-validator.pem'

chef\_server\_url 'https://chef\_server.com:443'

syntax\_check\_cache\_path '/home/demo/chef-repo/.chef/syntax\_check\_cache'

cookbook\_path [ '/home/demo/chef-repo/cookbooks' ]

knife[:digital\_ocean\_client\_id] = 'your\_client\_id'

knife[:digital\_ocean\_api\_key] = 'your\_api\_key'

Save and close the file when you are done.

Now, we can test to see if knife can connect to the DigitalOcean servers by asking for a list of our currently active droplets. Make sure you are in your Chef repo before running the knife command:

cd ~/chef-repo

knife digital\_ocean droplet list

ID Name Size Region IPv4 Image Status

111111 irssi 512MB New York 1 111.111.111.111 479972 (N/A) active

222222 try 4GB New York 2 222.222.222.222 1575388 (Ubuntu-Init) active

333333 nftables 4GB New York 2 333.333.333.333 308287 (Debian 7.0 x64) active

4444444 snmp 4GB New York 2 444.444.444.444 1575388 (Ubuntu-Init) active

5555555 node 4GB New York 2 555.555.555.555 1575388 (Ubuntu-Init) active

If you get a list of your current droplets, you've connected successfully and your authentication is working.

## Configure SSH Keys on your Chef Workstation

Now that you can connect to your DigitalOcean account using the knife tool, we need to go one step further to help automate the droplet creation process. We need to create an SSH key for our Chef workstation to embed in the droplets we will be created.

On our Workstation computer, we should create a new SSH key by typing:

ssh-keygen

Press enter through the prompts to accept the default values. This will create a hidden subdirectory in your user's home directory called .ssh with a public and private key inside.

Change into this directory right now by typing:

cd ~/.ssh

ls

id\_rsa id\_rsa.pub known\_hosts

The file that we need right now is the id\_rsa.pub file. You can view the contents of this file by typing:

cat id\_rsa.pub

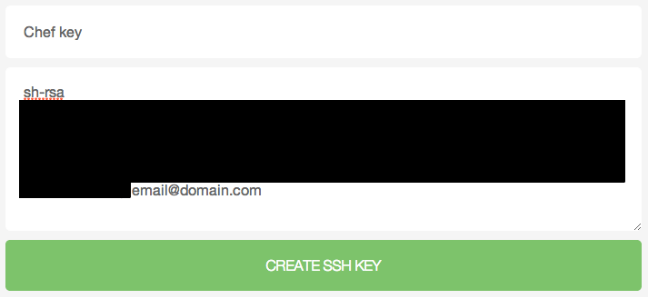
sh-rsa AAAAB3NzaC1yc2EAAAAxAQABAAABAQCv60WjxoM39LgPDbiW7ne3gu18q0NIV.....................lANpl5qmhDQ+GS/sO............mHWL2irjuB9xBXr00+44vSd2q/vtXdGXhdSMTf4/XK17fjKSG/9y3yD6nml6q9XgQxx9Vf/IKaKdlK0hbC1ds0+8/83PTb9dF3L7hf3Ch/ghvj5++twJFdFeG+VI7EDuKNA4zL8C5FdY.....................XIIeZvZ/z9Kp.....................nyiIuEAmn8fbnosWcsovw0IS1Hz6HsjYo4bu/gA82LWt3sdRUBZ/7ZsVD3ELip

email@domain.com

Back in the DigitalOcean control panel, click on the "SSH Keys" link in the left-hand menu. Click on the "Add SSH Key" button in the upper-right corner:



On the next page, create a name for your new key and then paste the contents of the file into the area given:



Click on the "Create SSH Key" button below when you are finished.

You now should have the ability to embed the SSH key from your Chef server into new droplets. This should allow you to create droplets from within the interface and then login to configure your server with the cookbooks and roles that you've created, all in a single step.

## Working with the Knife Plugin

Let's begin working with the knife plugin.

We can start by once again querying for the droplets associated with your DigitalOcean account:

knife digital\_ocean droplet list

ID Name Size Region IPv4 Image Status

111111 irssi 512MB New York 1 111.111.111.111 479972 (N/A) active

222222 try 4GB New York 2 222.222.222.222 1575388 (Ubuntu-Init) active

333333 nftables 4GB New York 2 333.333.333.333 308287 (Debian 7.0 x64) active

4444444 snmp 4GB New York 2 444.444.444.444 1575388 (Ubuntu-Init) active

5555555 node 4GB New York 2 555.555.555.555 1575388 (Ubuntu-Init) active

This gives us an overview of our droplets.

The knife plugin is rather simple, but contains all of the pieces you need to control the droplets within your account, perhaps with the exception of DNS functionality. The main functionality is creating and destroying droplets with the information that you can get from querying.

### Querying for Information Using the Knife Plugin

We can find out all of the information we may need for our create and destroy commands using different knife queries.

To find out which standard DigitalOcean images that we have available to use as base images, we can type this. The -G flag stands for "global":

knife digital\_ocean image list -G

ID Distribution Name Global

361740 Arch Linux Arch Linux 2013.05 x32 +

350424 Arch Linux Arch Linux 2013.05 x64 +

1602 CentOS CentOS 5.8 x32 +

1601 CentOS CentOS 5.8 x64 +

376568 CentOS CentOS 6.4 x32 +

. . .

This will give all of the standard DigitalOcean images, including all one-click applications that you can use to deploy a new droplet. The ID column is the item that you need to pay attention to in order to deploy from the correct image.

Similarly, if you want a list of your snapshots, backups, etc., you can type the same command without the -G flag:

knife digital\_ocean image list

ID Distribution Name Global

11xxxxx Ubuntu Dokku -

15xxxxx Ubuntu Ubuntu-Init -

15xxxxx Ubuntu Ubuntu-LAMP -

15xxxxx Ubuntu Ubuntu-WP -

You can use the image IDs from this list as base images as well for increased flexibility.

To find out which regions are available to deploy your droplet, type:

knife digital\_ocean region list

ID Name

1 New York 1

2 Amsterdam 1

3 San Francisco 1

4 New York 2

5 Amsterdam 2

For a list of sizes to deploy your droplet, you can type:

knife digital\_ocean size list

ID Name

63 1GB

62 2GB

64 4GB

65 8GB

61 16GB

. . .

To list the SSH keys that we can embed within an image, we can issue this command:

knife digital\_ocean sshkey list

ID Name

11111 Home key

22222 Chef key

## Creating and Destroying Droplets

We now know how to get all of the information necessary to create and destroy droplets.

We can start with destroying a droplet, since destruction always requires less effort than creation.

All we need to destroy a droplet is the droplet's ID. This is available in the first column of the droplet listing:

knife digital\_ocean droplet list

ID Name Size Region IPv4 Image Status

111111 irssi 512MB New York 1 111.111.111.111 479972 (N/A) active

222222 try 4GB New York 2 222.222.222.222 1575388 (Ubuntu-Init) active

333333 nftables 4GB New York 2 333.333.333.333 308287 (Debian 7.0 x64) active

4444444 snmp 4GB New York 2 444.444.444.444 1575388 (Ubuntu-Init) active

5555555 node 4GB New York 2 555.555.555.555 1575388 (Ubuntu-Init) active

Once you have the ID of the droplet you wish to delete, you can use it by typing:

knife digital\_ocean droplet destroy 111111

This should queue the droplet for deletion.

In order to create a droplet, you will need more information. Any of the fields that you usually need to fill out in the DigitalOcean control panel will need to be entered as parameters into this command.

The command will look like this:

knife digital\_ocean droplet create --server-name name\_for\_server --image image\_id --location region\_id --size size\_id --ssh-keys ssh\_key\_ids --bootstrap --run-list "nodes\_run\_list"

For instance, to create an x86\_64 Ubuntu 12.04 instance called "hello" on a 1G droplet in the NY2 region with an SSH key of 1111, we can type this:

knife digital\_ocean droplet create --server-name hello --image 1505447 --location 4 --size 63 --ssh-keys 22222

Make sure that you are passing in the SSH key of your Chef workstation so that you will be able to connect to it is online.

If we want to bootstrap the droplet as it is created and pass a run\_list to install and configure some software, we can add the additional parameters. Here, we can tell it to configure our new server as a web server:

knife digital\_ocean droplet create --server-name hello --image 1505447 --location 4 --size 63 --ssh-keys 22222 --bootstrap --run-list "role[web\_server]"

This will apply all of the recipes and attributes inherent in the role just as if we had edited the node's run\_list after creating it.

## Conclusion

Using the DigitalOcean knife plugin, you can easily spin up new infrastructure resources as you require them. For instance, if your database is getting more traffic than it previously was, it is trivial to spin up additional database servers to manage that load.

By including functionality to create new VPS instances in the same environment where you manage your infrastructure, it becomes easy to scale out your infrastructure with well-designed cookbooks, roles, and attributes. Scaling can simply become a question of when instead of how.

How To Manage Your Cluster with Chef and Knife on Ubuntu

Posted Oct 31, 2014

Top of Form

Bottom of Form

2

 17.3kviews [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials) [DigitalOcean](https://www.digitalocean.com/community/tags/digitalocean?type=tutorials) [Ubuntu](https://www.digitalocean.com/community/tags/ubuntu?type=tutorials)

Tutorial Series

This tutorial is part 7 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-manage-your-cluster-with-chef-and-knife-on-ubuntu#tutorial_series_10)

**Introduction**

Chef is a configuration management system – it's designed to give you a repeatable set of *recipes* for building your infrastructure, allowing you to automate, version, and test your infrastructure in much the same way as your application code.

This is the sixth tutorial in the [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorial_series/getting-started-managing-your-infrastructure-using-chef) series. In this guide, we'll assume that you've completed the other five tutorials, and so you've got a Chef server, workstation, and one or more nodes up and running.

Our Goal

**knife** is a command line tool packaged with Chef. You've likely already used knife to create and manage Chef cookbooks, data bags, or roles. When you issue a command with knife, you usually type something along the lines of:

knife cookbook create

The example command above uses the cookbook knife subcommand. This guide will introduce you to some new knife subcommands for issuing commands and getting information about your Chef cluster.

We'll cover:

* knife status
* knife ssh
* knife node

Prerequisites

This tutorial assumes you've followed along until the fifth guide, [How To Use the DigitalOcean Plugin for Knife to Manage Droplets in Chef](https://www.digitalocean.com/community/tutorials/how-to-use-the-digitalocean-plugin-for-knife-to-manage-droplets-in-chef), in the [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorial_series/getting-started-managing-your-infrastructure-using-chef) series.

Create Example Roles and Servers

If you don't have an established Chef cluster, or you'd like to follow this guide closely and see the same output, we can set up some example roles and servers.

First, on your **workstation**, change to your chef-repo directory:

cd ~/chef-repo

We're going to add a **backend** role to our existing **web\_server** role. For now it'll be blank, but later on you could turn this into a database or application server.

nano roles/backend.rb

Add this content to the backend.rb file:

name "backend"

description "Backend for application servers"

Then upload the new role to your Chef server.

knife role from file roles/backend.rb

Once you've done that, we can create some sample nodes using the DigitalOcean Knife plugin.

(Note: [This plugin](https://github.com/rmoriz/knife-digital_ocean) became unmaintained as of October 2014. It's up to you whether you want to start using it.)

knife digital\_ocean droplet create --server-name frontend01 --image 6918990 --location 4 --size 63 --ssh-keys 22222 --bootstrap --run-list "role[web\_server]"

knife digital\_ocean droplet create --server-name frontend02 --image 6918990 --location 4 --size 63 --ssh-keys 22222 --bootstrap --run-list "role[web\_server]"

knife digital\_ocean droplet create --server-name backend01 --image 6918990 --location 4 --size 63 --ssh-keys 22222 --bootstrap --run-list "role[backend]"

knife digital\_ocean droplet create --server-name backend02 --image 6918990 --location 4 --size 63 --ssh-keys 22222 --bootstrap --run-list "role[backend]"

Note: If your domain names for the various hosts in your Chef cluster don't resolve externally, and you're making them connect to each other with edited /etc/hosts files, the provisioning will not work, as the new server instance will be given the default /etc/hosts file.

You may still have some nodes active that you've created during the Chef tutorial series. That means some of the commands we'll run during this tutorial will have extra lines or information. If you'd like to follow along extactly, you can always remove these nodes by clicking **Nodes > Delete** from your Chef server's web interface.

Showing Status with knife status

The status subcommand is designed to show status information about your nodes. To use knife status, just change to your chef-repo directory and type:

knife status

You'll see a list of the nodes your Chef server knows about, including the times of their last chef-clientruns, node names, fully-qualified domain names, IP addresses, and platforms.

3 minutes ago, frontend01, fe1.yourdomain.com, 111.111.111.111, ubuntu 14.04.

3 minutes ago, frontend02, fe2.yourdomain.com, 222.222.222.222, ubuntu 14.04.

20 hours ago, backend01, be1.yourdomain.com, 333.333.333.333, ubuntu 14.04.

3 minutes ago, backend02, be2.youdomain.com, 333.333.333.333, ubuntu 14.04.

We can immediately see from this that we need to take a closer look at **backend01** - it hasn't succesfully run chef-client ("checked in" in Chef parlance) in about 20 hours.

If you've got a node in a similar situation, check your chef-client error logs, or use **Reports > Run History** from Chef server's web interface.

Issuing Commands with knife ssh

We can use knife ssh to issue commands to our nodes (or some subset of nodes) at once. For example, we can use knife ssh to restart Nginx on all our nodes with the role **frontend**.

You'll need a user authorized to SSH into the server (i.e., you can run ssh yourusername@fe1.yourdomain.com and get a shell). If you don't have SSH keys set up, you can use the -P option to prompt for a password.

knife ssh "role:web\_server" "service nginx restart" -x yourusername -a ipaddress

You should get output that looks like:

111.111.111.111 \* Restarting nginx nginx [ OK ]

222.222.222.222 \* Restarting nginx nginx [ OK ]

We just ran a command on all our **frontend** servers by issuing a single knife command. Let's pull apart the arguments and learn a bit more about how knife ssh works.

knife ssh "web\_server" "service nginx restart" -x yourusername -a ipaddress

The first argument to knife ssh is a Chef search query - usually you'll want something likerole:YOUR\_ROLE\_NAME, but you can also search via many other attributes (and combine them with Boolean operators). For instance, to run a command on only nodes with **ubuntu** in their **platform** attribute, you could run:

knife ssh "platform:ubuntu\*" "service nginx restart" -x yourusername -a ipaddress

The **\*** in the above is a wildcard character. It will match zero or more characters in the attribute value. In this case, it will allow us to match the value **ubuntu 14.04** as displayed in the knife status output we looked at earlier.

knife ssh "role:web\_server" "service nginx restart" -x yourusername -a ipaddress

The second argument to knife ssh is the command that you want to run on the servers (that match the search query). It can be almost anything you would type into a shell. You can even join commands with a semicolon (**;**).

knife ssh "role:web\_server" "uptime;date" -x yourusername -a ipaddress

Which would output something like:

111.111.111.111 12:53:36 up 2 days, 15:25, 1 user, load average: 0.08, 0.03, 0.05

111.111.111.111 Wed Oct 22 12:53:36 UTC 2014

222.222.222.222 12:53:30 up 2 days, 15:21, 1 user, load average: 0.00, 0.01, 0.05

222.222.222.222 Wed Oct 22 12:53:30 UTC 2014

The -x argument we've already covered - it's the SSH username to use for logging in.

The -a argument specifies which node attribute to use to as the address for SSH. By default, it's the FQDN of your node (remember, you can find this with the knife status command shown earlier), so if you can resolve your servers by visiting **http://fe1.yourdomain.com**, then you can omit the -a option.

Interactive knife ssh

knife ssh also has the ability to put you into an interactive shell, where you can issue a series of commands and see the results very quickly. You can start an interactive knife ssh shell by usinginteractive in place of your SSH command.

knife ssh "role:web\_server" interactive -x yourusername -a ipaddress

This will show us:

Connected to 111.111.111.111 and 222.222.222.222

To run a command on a list of servers, do:

on SERVER1 SERVER2 SERVER3; COMMAND

Example: on latte foamy; echo foobar

To exit interactive mode, use 'quit!'

knife-ssh>

You can issue commands to all servers in your search result by simply typing the command and pressing**Enter**.

knife-ssh> uptime

111.111.111.111 18:43:55 up 2 days, 21:16, 1 user, load average: 0.01, 0.03, 0.05

222.222.222.222 18:43:49 up 2 days, 21:11, 1 user, load average: 0.00, 0.01, 0.05

If you'd like to refine your server list further, just use on, as the help message suggests. Note that you should replace**SERVER1** in the example command with the attribute you've used as -a; for us, this is the node's IP address.

knife-ssh> on 111.111.111.111; echo hello digitalocean

111.111.111.111 hello digitalocean

One really cool use of interactive knife ssh is using it to tail server logs. For instance, if your access log for Nginx is in the default location (/var/log/nginx/access.log), you can tail the logs with the -f(follow) option, and the results will continually print to your console.

knife-ssh> tail -f /var/log/nginx/access.log

(Depending on how Nginx is set up, you may have to use sudo in front of this command.)

If you visit the IP address for one of your nodes in a web browser, you'll see an entry appear in the Nginx access log, without having to do anything!

Managing Nodes with knife node

If you've followed the [How To Create Simple Chef Cookbooks to Manage Infrastructure on Ubuntu](https://www.digitalocean.com/community/tutorials/how-to-create-simple-chef-cookbooks-to-manage-infrastructure-on-ubuntu) guide, then you've already used knife node to list all nodes on your Chef server, and to edit node attributes.

knife node list

knife node edit frontend01

You can also use knife node to delete nodes.

knife node delete frontend01

Or, show more detailed node attributes:

knife node show frontend01

\*\*\*\*\*\*\*

Node Name: frontend01

Environment: \_default

FQDN: fe01.yourdomain.com

IP: 111.111.111.111

Run List: role[frontend]

Roles: web\_server

Recipes: apt, nginx, apt::default, nginx::default

Platform: ubuntu 14.04

Tags:

You can get a full listing of your node's attributes using the -l option:

knife node show -l frontend01

This will return a very long list of attributes, most of them populated automatically by **Ohai**, which is a Chef tool that passes attributes automatically to chef-client each time it's run.

A long printed list of attributes is not neccessarily very useful to us. However, knife has us covered there too. We can specify the --format option in order to get a JSON or YAML representation of a Node's attributes.

knife node show frontend01 --format json

\*\*\*\*\*\*\*

{

"name": "frontend01",

"chef\_environment": "\_default",

"run\_list": [

"role[web\_server]"

]

,

"normal": {

"tags": [

]

}

}

You can also retrieve a single node attribute with -a.

knife node show frontend01 --format json -a ipaddress

\*\*\*\*\*\*\*

{

"frontend01": {

"ipaddress": "111.111.111.111"

}

}

JSON or YAML output might be very useful if we wanted to build more complex scripts that involved knife, or even for displaying dashboards and metrics.

Conclusion

knife is a powerful tool, not just for creating and updating the various objects in your Chef cluster, but also for viewing and managing the state of your cluster.

With knife ssh you can write one command and have it run on many nodes simultaneously – a very powerful tool for any devops engineer.

How To Automatically Add New Droplets to your Configuration Management System

Posted Feb 26, 2015

Top of Form

Bottom of Form

3

 7.8kviews [API](https://www.digitalocean.com/community/tags/api?type=tutorials) [Chef](https://www.digitalocean.com/community/tags/chef?type=tutorials) [Configuration Management](https://www.digitalocean.com/community/tags/configuration-management?type=tutorials)

Tutorial Series

This tutorial is part 8 of 8 in the series: [Getting Started Managing Your Infrastructure Using Chef](https://www.digitalocean.com/community/tutorials/how-to-automatically-add-new-droplets-to-your-configuration-management-system#tutorial_series_10)

**Introduction**

Using the DigitalOcean metadata service, administrators can provide instructions that allow new servers to configure themselves automatically. While this is useful, many organizations like to handle all of their infrastructure configuration within a configuration management tool like Chef or Puppet.

In this guide, we will demonstrate how to bootstrap a DigitalOcean server using the metadata service and CloudInit to connect to an existing configuration management deployment. The actual configuration of the server can then be handled by the config management service. We will demonstrate how to bootstrap both Chef and Puppet nodes.

Prerequisites

In order to complete this guide, you will have to have some familiarity with the DigitalOcean metadata service. You can find out more about how to enter information into and retrieve information from the metadata service in [this guide](https://www.digitalocean.com/community/tutorials/an-introduction-to-droplet-metadata).

This guide will leverage a type of script called cloud-config that is consumed at first boot by the CloudInit service on your Droplet in order to perform first-run configuration. You should get some basic familiarity with cloud-config scripts, their syntax, and behavior in order to better understand how to modify the scripts presented in this guide. You can find an introduction to cloud-config scripting [here](https://www.digitalocean.com/community/tutorials/an-introduction-to-cloud-config-scripting). For a more practical example (along with some discussion on the limitations of the format), you can read our guide on performing some basic tasks using cloud-config [here](https://www.digitalocean.com/community/tutorials/how-to-use-cloud-config-for-your-initial-server-setup).

Using Cloud-Config Scripts to Bootstrap a Chef Node

Using the DigitalOcean metadata service, you can easily hook your new servers into an existing Chef-controlled infrastructure with cloud-config scripting.

To add your new server to this system, you must already have a Chef server configured that your new server can contact to receive configuration instructions. If you need help deploying a Chef server and management workstation, you can follow [this guide](https://www.digitalocean.com/community/tutorials/how-to-install-a-chef-server-workstation-and-client-on-ubuntu-vps-instances) to get started.

**General Plan**

When a new server is brought online, it must be brought under the control of the Chef server. Typically, this may be accomplished by connecting to the new server with the knife management command and using the bootstrap subcommand. This would connect to the new server, install the Chef client and the validation credentials that allow the new node to connect to the Chef server. Afterwards, the Chef client connects to the server, validates itself, receives new client credentials, pulls down its configuration from the server, and performs any actions necessary to bring itself into the desired state.

In this guide, we will use a cloud-config script to replace the manual bootstrapping step, allowing the new node to automatically connect to the Chef server, validate itself, receive client credentials, and perform an initial Chef client run. The server will do this automatically at first boot without any manual assistance from the administrator.

**Gathering the Necessary Data from the Knife Config File**

In order for our cloud-config script to successfully bootstrap, it will need access to the credentials typically available to the knife command. Specifically, we need the following pieces of information:

* The Chef validation name
* The validation key
* The URL where the Chef server can be reached

All of this information is available, in the correct format, in the knife configuration file on workstation used to manage the Chef infrastructure. Inside of the Chef repo, there should be a hidden directory called.chef which contains this file.

Assuming that your Chef repo is located in your home directory on the workstation and is called chef-repo, you can output the contents of the file by typing:

cat ~/chef-repo/.chef/knife.rb

The pieces of information you need are highlighted below:

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

log\_level :info

log\_location STDOUT

node\_name "jellingwood"

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

validation\_client\_name "digitalocean-validator"

validation\_key "#{current\_dir}/digitalocean-validator.pem"

chef\_server\_url "https://your\_server.com/organizations/digitalocean"

syntax\_check\_cache\_path "#{ENV['HOME']}/.chef/syntaxcache"

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

The validation name and the Chef server URL can be taken directly as-is from the file. Copy these values so that you can use them in the cloud-config file.

The validation\_key points to the location where the actual key is kept. In the above example, this indicates that it is located in the same directory as the knife.rb file and is called digitalocean-validator.pem. This will likely be different for your configuration.

We need to contents of this file, so use the cat command again. Modify the command to point it to the location given for your validator key:

cat ~/chef-repo/.chef/digitalocean-validator.pem

You will see an RSA private key:

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA3O60HT5pwEo6xUwcZ8WtExBUhoL3bTjlsvHVXg1JVmBUES+f

V9jLu2N00uSZEDZneCIQyHLBXnqD/UNvWEPNvPzt1ecXzmw2BytB7lPDW4/F/8tJ

vAVrKqC7B04VFGmcFY2zC8gf8BWmX8CNRDQooM7UO5OWe/H6GDGPPRIITerO3GrU

. . .

sWyRAoGBAKNc/ZUM8ljRV0UJxQ9nbdozXRZjtUaNgXMNiw+oP2HYYdHrlkKnGHYJ

Js63rvjpq8pocjE8YI+2H0v4/4uWqW8GEBfrWbLMzGsYPnRyiHR5+hgjCUU50RB3

eFoNbURwLYcq2Z/IAQZpDpJWpofz3OVMpMXtei1cIflrAAd2wtWO

-----END RSA PRIVATE KEY-----

Copy the entirety of the validation key so that you can use it in the cloud-config script momentarily.

**Basic Cloud-Config Chef Client Installation**

Once you have the data above, you can build out the script. Chef configuration can be accomplished through a dedicated cloud-config module called chef. The cloud-config must contain valid YAML and must have #cloud-config as the first line of the script.

Starting off, your script will look like this:

#cloud-config

chef:

The cloud-config documentation claims to be able to install the Chef client either from a Ruby gem, a package, or using the traditional "omnibus" installation method. However, in practice, both the gem and package methods tend to fail, so we will use the "omnibus" method. Although it is usually not necessary, we will also explicitly list the location of the omnibus installer.

We will set force\_install to "false". This way, if for some reason the Chef client is already installed on the image (for instance, if you are deploying from a snapshot), the client will not be reinstalled. So far, our script looks like this:

#cloud-config

chef:

install\_type: "omnibus"

omnibus\_url: "https://www.opscode.com/chef/install.sh"

force\_install: false

Next, we have the option of selecting a name for the new server within the Chef infrastructure by using thenode\_name directive. If you do not set this, Chef will use the server's hostname, so this is optional. However, this must be unique in your Chef environment.

Afterwards, we can add all of the connection information that we took from our Chef workstation. We will set the server\_url option to the location of the Chef server exactly as it was in the knife.rb file. The same is true for the validation\_name option.

For the validation key, we will use the YAML pipe symbol (|) to enter the entire validation key that we found on the workstation:

#cloud-config

chef:

install\_type: "omnibus"

omnibus\_url: "https://www.opscode.com/chef/install.sh"

force\_install: false

node\_name: "new\_node"

server\_url: "https://your\_server.com/organizations/digitalocean"

validation\_name: "digitalocean-validator"

validation\_key: |

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA3O60HT5pwEo6xUwcZ8WtExBUhoL3bTjlsvHVXg1JVmBUES+f

V9jLu2N00uSZEDZneCIQyHLBXnqD/UNvWEPNvPzt1ecXzmw2BytB7lPDW4/F/8tJ

vAVrKqC7B04VFGmcFY2zC8gf8BWmX8CNRDQooM7UO5OWe/H6GDGPPRIITerO3GrU

. . .

sWyRAoGBAKNc/ZUM8ljRV0UJxQ9nbdozXRZjtUaNgXMNiw+oP2HYYdHrlkKnGHYJ

Js63rvjpq8pocjE8YI+2H0v4/4uWqW8GEBfrWbLMzGsYPnRyiHR5+hgjCUU50RB3

eFoNbURwLYcq2Z/IAQZpDpJWpofz3OVMpMXtei1cIflrAAd2wtWO

-----END RSA PRIVATE KEY-----

At this point, your script has all of the authentication needed to connect to your Chef server and create client credentials.

**Configuring Chef Environment, run\_list, and Attributes**

While the above details provides enough information for the client to connect to the Chef server, we haven't given the node any information about how to actually configure itself. We can provide this information in the cloud-config script as well.

To specify the environment that the new node should be placed in, use the environment option. If this is not set, the \_default environment will be set, which is the generic default for Chef nodes that have not been given another environment.

chef:

environment: "staging"

Our run\_list can be specified as a simple list of items that the client should apply in order. These can be either recipes or roles.

chef:

run\_list:

- "recipe[lamp]"

- "role[backend-web]"

You can specify the new node's initial attributes using an initial\_attributes hierarchy. This will set the initial attributes that will affect how the run\_list is applied:

chef:

initial\_attributes:

lamp:

apache:

port: 80

mysql:

username: webclient

pass: $#fjeaiop34S

When hooked up to the previous cloud-config script, it might look something like this:

#cloud-config

chef:

install\_type: "omnibus"

omnibus\_url: "https://www.opscode.com/chef/install.sh"

force\_install: false

node\_name: "new\_node"

server\_url: "https://your\_server.com/organizations/digitalocean"

validation\_name: "digitalocean-validator"

validation\_key: |

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA3O60HT5pwEo6xUwcZ8WtExBUhoL3bTjlsvHVXg1JVmBUES+f

V9jLu2N00uSZEDZneCIQyHLBXnqD/UNvWEPNvPzt1ecXzmw2BytB7lPDW4/F/8tJ

vAVrKqC7B04VFGmcFY2zC8gf8BWmX8CNRDQooM7UO5OWe/H6GDGPPRIITerO3GrU

. . .

sWyRAoGBAKNc/ZUM8ljRV0UJxQ9nbdozXRZjtUaNgXMNiw+oP2HYYdHrlkKnGHYJ

Js63rvjpq8pocjE8YI+2H0v4/4uWqW8GEBfrWbLMzGsYPnRyiHR5+hgjCUU50RB3

eFoNbURwLYcq2Z/IAQZpDpJWpofz3OVMpMXtei1cIflrAAd2wtWO

-----END RSA PRIVATE KEY-----

environment: "staging"

run\_list:

- "recipe[lamp]"

- "role[backend-web]"

initial\_attributes:

lamp:

apache:

port: 80

mysql:

username: webclient

pass: $#fjeaiop34S

**Redirecting Output and Configuring Chef Client Run**

The above script contains all of the information needed under the chef: section. However, there are a few other things we should do using some other cloud-config modules.

First, we should specify that we wish to redirect the output from every command and subcommand into the CloudInit process's output log. This is located at /var/log/cloud-init-output.log by default. We can do this with the output module like this:

output: {all: '| tee -a /var/log/cloud-init-output.log'}

The other thing we want to do is set the Chef client up to actually run once it has been installed and configured. At the time of this writing, the omnibus installation method does not do this automatically.

We can force this behavior by waiting until the chef-client executable is installed on the server before calling the command. Using a simple bash loop, we will check for the existence of this file every five seconds. When it is found, we will run chef-client in order to implement the initial configuration we have specified.

The runcmd module can be used to issue arbitrary commands. It is the ideal location for our bash loop:

runcmd:

- while [ ! -e /usr/bin/chef-client ]; do sleep 5; done; chef-client

Also, optionally, you can add another cloud-config directive to null-route the metadata endpoint after the first boot. This is useful because we are putting a private key in our user data. Without null-routing the metadata endpoint, this would be accessible to any user on the server. Implement this by adding:

disable\_ec2\_metadata: true

Combining these with the script we've constructed thus far, we can get the complete script necessary to bootstrap our node and connect it to our Chef infrastructure:

#cloud-config

chef:

install\_type: "omnibus"

omnibus\_url: "https://www.opscode.com/chef/install.sh"

force\_install: false

node\_name: "new\_node"

server\_url: "https://your\_server.com/organizations/digitalocean"

validation\_name: "digitalocean-validator"

validation\_key: |

-----BEGIN RSA PRIVATE KEY-----

MIIEowIBAAKCAQEA3O60HT5pwEo6xUwcZ8WtExBUhoL3bTjlsvHVXg1JVmBUES+f

V9jLu2N00uSZEDZneCIQyHLBXnqD/UNvWEPNvPzt1ecXzmw2BytB7lPDW4/F/8tJ

vAVrKqC7B04VFGmcFY2zC8gf8BWmX8CNRDQooM7UO5OWe/H6GDGPPRIITerO3GrU

. . .

sWyRAoGBAKNc/ZUM8ljRV0UJxQ9nbdozXRZjtUaNgXMNiw+oP2HYYdHrlkKnGHYJ

Js63rvjpq8pocjE8YI+2H0v4/4uWqW8GEBfrWbLMzGsYPnRyiHR5+hgjCUU50RB3

eFoNbURwLYcq2Z/IAQZpDpJWpofz3OVMpMXtei1cIflrAAd2wtWO

-----END RSA PRIVATE KEY-----

environment: "staging"

run\_list:

- "recipe[lamp]"

- "role[backend-web]"

initial\_attributes:

lamp:

apache:

port: 80

mysql:

username: webclient

pass: $#fjeaiop34S

output: {all: '| tee -a /var/log/cloud-init-output.log'}

runcmd:

- while [ ! -e /usr/bin/chef-client ]; do sleep 5; done; chef-client

disable\_ec2\_metadata: true

The above script can be tweaked as necessary for each new server in your infrastructure.

Using Cloud-Config Scripts to Bootstrap a Puppet Node

If your infrastructure relies on Puppet for configuration management, you can use the puppet module instead. Like the Chef example, bootstrapping a Puppet node involves using cloud-config to attach the new server to the existing configuration management infrastructure.

Before you get started, you should have a Puppet master server configured for your infrastructure. If you need help getting a Puppet server up and running, check out [this guide](https://www.digitalocean.com/community/tutorials/how-to-install-puppet-to-manage-your-server-infrastructure).

**General Plan**

When a new Puppet server is brought online, a Puppet agent is installed so that it can communicate with the Puppet master server. This agent is responsible for receiving and applying the information that dictates the desired state of the node. To do this, the agent connects with the master, uploads data about itself, pulls down the current catalog describing its desired state, and performs the actions necessary to reach that state.

Before this happens though, on its first run, the agent must register itself with the master server. It creates a certificate signing request and sends it to the master to sign. Typically, the agent will reconnect to the master periodically until the certificate is signed, but you can configure your Puppet to automatically sign incoming requests with certain characteristics if that is suitable for your environment (we will cover this later).

Using our cloud-config script, we will configure our new server with the information that it needs to connect to the master for the first time. At that point, it can retrieve configuration details from the Puppet master server in the form of a catalog.

**Gathering the Necessary Data from the Puppet Master**

The first thing we need to do prior to building our cloud-config file is gather the data from our Puppet master server that we will need to connect. We only need a few pieces of information.

First, you need to get the Puppet master server's fully qualified domain name (FQDN). You can do this by typing:

hostname -f

In most cases, it should return something like this:

puppet.example.com

You can also check your Puppet master configuration file to see if the dns\_alt\_names option is set:

cat /etc/puppet/puppet.conf

. . .

dns\_alt\_names = puppet,puppet.example.com

. . .

If your Puppet master's SSL certificates were generated after setting these options, they may be usable as well.

The other item that we need to collect is the Puppet master's certificate authority certificate. This can be found in either /var/lib/puppet/ssl/certs/ca.pem or /var/lib/puppet/ssl/ca/ca\_crt.pem:

sudo cat /var/lib/puppet/ssl/certs/ca.pem

The results will look something like this:

-----BEGIN CERTIFICATE-----

MIIFXjCCA0agAwIBAgIBATANBgkqhkiG9w0BAQsFADAcMRowGAYDVQQDDBFQdXBw

ZXQgQ0E6IHB1cHBldDAeFw8xNTAyMTkxOTA0MzVaFw0yMDAyMTkxOTA0MzVaMBwx

GjAYBgNVBAMMEVB1cHBldCBDQTogcHVwcGV0MIICIjANBgkqhkiG9w0BAQEFAAOC

. . .

arsjZT5/CtIhtP33Jl3mCp7U2F6bsk4/GDGRaAsFXjJHvBbL93NzgpkZ7elf0zUP

rOcSGrDrUuzuJk8lEAtrZr/IfAgfKKXPqbyYF95V1qN3OMY+aTcrK20XTydKVWSe

l5UfYGY3S9UJFrSn9aBsZzN+10HXPkaFKo7HxpztlYyJNI8UVSatcRF4aYYqt9KR

UClnR+2WxK5v7ix0CVd4/KpYH/6YivvyTwxrhjF2AksZKg==

-----END CERTIFICATE-----

Copy the certificate in its entirety. We will be including this in our cloud-config file so that our new servers can verify that they are connecting to the correct Puppet master.

Once you have these pieces of information, you can begin building the cloud-config file so that the new server can plug itself into the existing Puppet infrastructure.

**Basic Cloud-Config Puppet Node Installation**

The cloud-config configuration for new Puppet nodes is fairly simple. All Puppet-specific configuration is located within the puppet: section of the file. As with every cloud-config file, the very first line must contain #cloud-config on its own:

#cloud-config

puppet:

Beneath this, there are only two subsections. The first is the ca\_cert key. This will use the pipe character to start a YAML text block so that the CA certificate can be given in its entirety as an indented block:

#cloud-config

puppet:

ca\_cert: |

-----BEGIN CERTIFICATE-----

MIIFXjCCA0agAwIBAgIBATANBgkqhkiG9w0BAQsFADAcMRowGAYDVQQDDBFQdXBw

ZXQgQ0E6IHB1cHBldDAeFw8xNTAyMTkxOTA0MzVaFw0yMDAyMTkxOTA0MzVaMBwx

GjAYBgNVBAMMEVB1cHBldCBDQTogcHVwcGV0MIICIjANBgkqhkiG9w0BAQEFAAOC

. . .

arsjZT5/CtIhtP33Jl3mCp7U2F6bsk4/GDGRaAsFXjJHvBbL93NzgpkZ7elf0zUP

rOcSGrDrUuzuJk8lEAtrZr/IfAgfKKXPqbyYF95V1qN3OMY+aTcrK20XTydKVWSe

l5UfYGY3S9UJFrSn9aBsZzN+10HXPkaFKo7HxpztlYyJNI8UVSatcRF4aYYqt9KR

UClnR+2WxK5v7ix0CVd4/KpYH/6YivvyTwxrhjF2AksZKg==

-----END CERTIFICATE-----

Be sure to include the entire certificate along with the beginning and ending markers and to indent it appropriately.

The second section under the puppet: umbrella is the conf: section. This is used to specify key-value pairs that will be appended to a generic puppet.conf file. The key-value pairs should be placed under section headers as they would be in the puppet.conf file.

For instance, at the very least, the new server will need to know the address of the Puppet master server. In the puppet.conf file, this is found under the [agent] section, like this:

. . .

[agent]

server = puppet.example.com

. . .

To specify this in the cloud-config syntax, you would add this to what we have so far:

#cloud-config

puppet:

ca\_cert: |

-----BEGIN CERTIFICATE-----

MIIFXjCCA0agAwIBAgIBATANBgkqhkiG9w0BAQsFADAcMRowGAYDVQQDDBFQdXBw

ZXQgQ0E6IHB1cHBldDAeFw8xNTAyMTkxOTA0MzVaFw0yMDAyMTkxOTA0MzVaMBwx

GjAYBgNVBAMMEVB1cHBldCBDQTogcHVwcGV0MIICIjANBgkqhkiG9w0BAQEFAAOC

. . .

arsjZT5/CtIhtP33Jl3mCp7U2F6bsk4/GDGRaAsFXjJHvBbL93NzgpkZ7elf0zUP

rOcSGrDrUuzuJk8lEAtrZr/IfAgfKKXPqbyYF95V1qN3OMY+aTcrK20XTydKVWSe

l5UfYGY3S9UJFrSn9aBsZzN+10HXPkaFKo7HxpztlYyJNI8UVSatcRF4aYYqt9KR

UClnR+2WxK5v7ix0CVd4/KpYH/6YivvyTwxrhjF2AksZKg==

-----END CERTIFICATE-----

conf:

agent:

server: "puppet.example.com"

Note that the conf: section is in-line with the ca\_cert section and not a child element. This is the bare minimum needed to connect to the Puppet master. Any additional configuration items found inpuppet.conf can be added in a similar way by first creating a level for the section name and then defining the key-value pair.

After this, we should redirect all future output to the cloud-init-output.log file and add a runcmd line comparable to the one we added for the Chef config. This will wait until the Puppet agent is installed and then enable and restart it. We can also null-route the metadata endpoint after the first run like we did in the Chef section. These lines cloud-config directives should be placed outside of any other module sections:

. . .

conf:

agent:

server: "puppet.example.com"

output: {all: '| tee -a /var/log/cloud-init-output.log'}

runcmd:

- while [ ! -e /usr/bin/puppet ]; do sleep 5; done; puppet agent --enable; service puppet restart

disable\_ec2\_metadata: true

With this information, the new server can connect to the Puppet master server and then generate a client certificate signing request to transfer to the master. By default, client certificates must be manually signed on the Puppet master. Once this is done, at the next Puppet agent update interval (every 30 minutes by default), the node will pull down its configuration from the Puppet master. We will demonstrate a bit later how to implement a relatively secure auto-signing mechanism to avoid this delay.

**Defining the Certname for the Node**

One of the values that can be placed into the new server's puppet.conf file is a unique case. In thecloud-config file, the certname option can substitute values from the environment if certain variables are given. The following variables are recognized:

* **%i**: The instance ID of the server. This will be taken fromhttp://169.254.169.254/metadata/v1/id when the server is created. It corresponds to the Droplet ID used to uniquely identify Droplets.
* **%f**: The FQDN of the server.

With this in mind, a common certname setting would look like this:

#cloud-config

puppet:

. . .

conf:

agent:

server: "puppet.example.com"

certname: "%i.%f"

This would produce a certname with a pattern similar to this:

|-Droplet ID

|

| |-Fully Qualified Domain Name

| |

|-----||-------------------|

123456.testnode.example.com

Having Droplet ID as part of the certname can be useful for configuring secure Puppet auto-signing as we will see in the next section.

**Implement Puppet Certificate Auto Signing**

If you wish to implement a certificate auto-signing system to avoid the need for administrator intervention, there are a few options. You must set this up on your Puppet master server first.

In the puppet.conf file on the Puppet master server, you can set the autosign option under the[master] section of the file. This can take a few different values:

* **true**: This will tell the Puppet master server to sign every certificate request that comes in, without doing any checks. This is extremely dangerous in a real environment because any host can get a CSR signed and enter your infrastructure.
* **<whitelist\_filename>**: The second option is to specify a file that will function as a whitelist of hosts or host regular expressions. The Puppet master will check certificate signing requests against this list to see if the certificate should be signed. This is again not recommended as the certificate names can be spoofed easily.
* **<policy\_executable>**: The third option is to specify a script or executable that can be run to determine whether the certificate signing request should be signed. Puppet will pass the certname in as an argument and the entire CSR in through standard input. If an exit status of 0 is returned, the certificate is signed. If another status is given, the certificate will *not* be signed.

Policy-based auto-signing is the most secure way to implement automatic key signing because it allows you to be arbitrarily complex in how you distinguish between legitimate and non-legitimate requests.

To demonstrate policy-based auto-signing, you can add the certname variable to your cloud-config that includes the %i instance ID variable. We will use %i.%f so that it also includes the hostname selected as well:

#cloud-config

puppet:

conf:

agent:

server: "puppet.example.com"

certname: "%i.%f"

ca\_cert: |

. . .

Your complete cloud-config may now look something like this:

#cloud-config

puppet:

conf:

agent:

server: "puppet.example.com"

certname: "%i.%f"

ca\_cert: |

-----BEGIN CERTIFICATE-----

MIIFXjCCA0agAwIBAgIBATANBgkqhkiG9w0BAQsFADAcMRowGAYDVQQDDBFQdXBw

ZXQgQ0E6IHB1cHBldDAeFw8xNTAyMTkxOTA0MzVaFw0yMDAyMTkxOTA0MzVaMBwx

GjAYBgNVBAMMEVB1cHBldCBDQTogcHVwcGV0MIICIjANBgkqhkiG9w0BAQEFAAOC

. . .

arsjZT5/CtIhtP33Jl3mCp7U2F6bsk4/GDGRaAsFXjJHvBbL93NzgpkZ7elf0zUP

rOcSGrDrUuzuJk8lEAtrZr/IfAgfKKXPqbyYF95V1qN3OMY+aTcrK20XTydKVWSe

l5UfYGY3S9UJFrSn9aBsZzN+10HXPkaFKo7HxpztlYyJNI8UVSatcRF4aYYqt9KR

UClnR+2WxK5v7ix0CVd4/KpYH/6YivvyTwxrhjF2AksZKg==

-----END CERTIFICATE-----

output: {all: '| tee -a /var/log/cloud-init-output.log'}

runcmd:

- while [ ! -e /usr/bin/puppet ]; do sleep 5; done; puppet agent --enable; service puppet restart

disable\_ec2\_metadata: true

On the Puppet master server, we will have to set up a validation script. Since Ruby is already installed for Puppet, we can make a simple Ruby script.

Because we are using the %i.%f format for the certname, we can check whether the first part of thecertname (the part before the first dot) corresponds with a valid Droplet ID for our account. This is just a simple check which, in practice doesn't do much more than the white list file. However, you can adapt this idea to be much more complex if you wish.

To do this, we will need a personal access token from the "Apps & API" section of the DigitalOcean control panel. You will also need to install one of the DigitalOcean Ruby libraries. Below, we will show you some simplified scripts that use the [Barge](https://github.com/boats/barge) and [DropletKit](https://github.com/digitalocean/droplet_kit) DigitalOcean Ruby clients.

If you wish to use the Barge client, install the gem on your Puppet master by typing:

sudo gem install barge

The following script can be used to check whether the first portion of the certname in the certificate signing request corresponds with a valid Droplet ID:

#!/usr/bin/env ruby

**require** 'barge'

TOKEN = 'YOUR\_DIGITALOCEAN\_API\_TOKEN'

droplet\_ids = []

certname = ARGV[0]

id\_string = certname.slice(0...(certname.index('.')))

id\_to\_check = id\_string.to\_i

client = Barge::Client.new(access\_token: TOKEN)

droplets = client.droplet.all

droplets.droplets.each **do** |droplet|

droplet\_ids << droplet.id

**end**

Kernel.exit(droplet\_ids.**include**?(id\_to\_check))

If you instead wish to use DropletKit, the official DigitalOcean Ruby client, you can install the gem by typing:

sudo gem install droplet\_kit

Note that DropletKit gem is only valid for Ruby 2.0 and above, so this might not be a possibility when using the version of Ruby that comes with Puppet.

The script for DropletKit can be adapted like this:

#!/usr/bin/env ruby

**require** 'droplet\_kit'

TOKEN = 'YOUR\_DIGITALOCEAN\_API\_TOKEN'

droplet\_ids = []

certname = ARGV[0]

id\_string = certname.slice(0...(certname.index('.')))

id\_to\_check = id\_string.to\_i

client = DropletKit::Client.new(access\_token: TOKEN)

droplets = client.droplets.all

droplets.each **do** |droplet|

droplet\_ids << droplet.id

**end**

Kernel.exit(droplet\_ids.**include**?(id\_to\_check))

You can place the script that corresponds to the gem you installed in a file called/etc/puppet/validate.rb and mark it as executable by typing:

sudo chmod +x /etc/puppet/validate.rb

You can then add the following to your puppet.conf file (located at /etc/puppet/puppet.conf if using Open Source Puppet):

. . .

[master]

autosign = /etc/puppet/validate.rb

. . .

Restart the Apache service to implement the new signing policy:

sudo service apache2 restart

Now, when certificate signing requests are received by your Puppet master, it will check whether the first part of the certificate name corresponds with a valid Droplet name in your account. This is a rough example of how you can validate requests using an executable.

Conclusion

By leveraging cloud-config scripts, you can easily bootstrap your new servers and hand them off to your existing configuration management systems. This allows you to control your infrastructure immediately through your existing tools prior to making important changes outside of scope of your management solution.