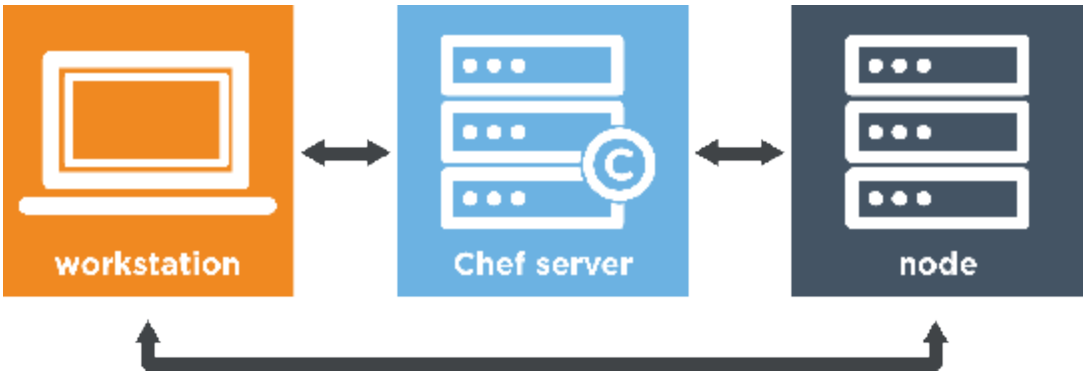
Chef - Manage a node

Introduction

Chef is a configuration management and automation platform from Opscode. Chef helps you describe your infrastructure with code (IaC). Because your infrastructure is managed with code, it is flexible, versionable, and human-readable. Also, it can be automated, tested and reproduced with ease.

Typically, Chef is comprised of three parts – your workstation, a Chef server, and nodes.

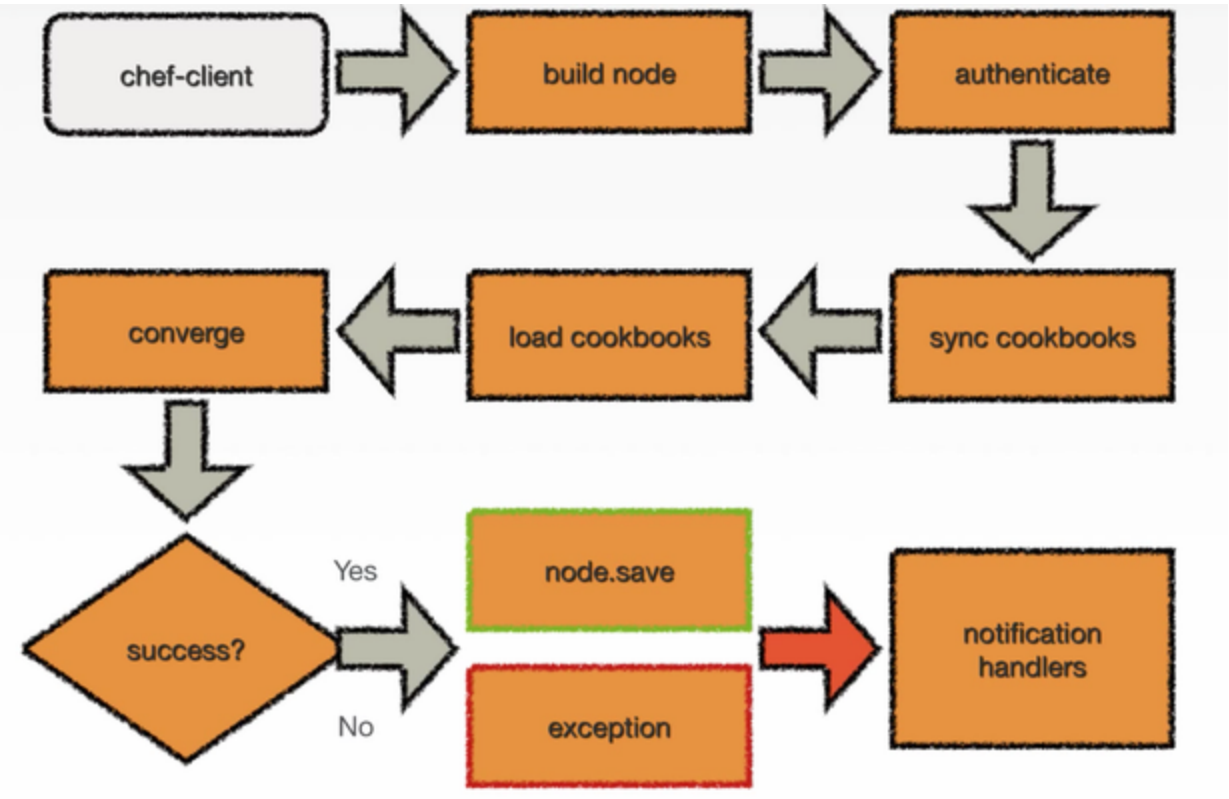


Workstation is the computer from which you write cookbooks and administer your network. A workstation can be any OS - Linux, Mac OS, or Windows.

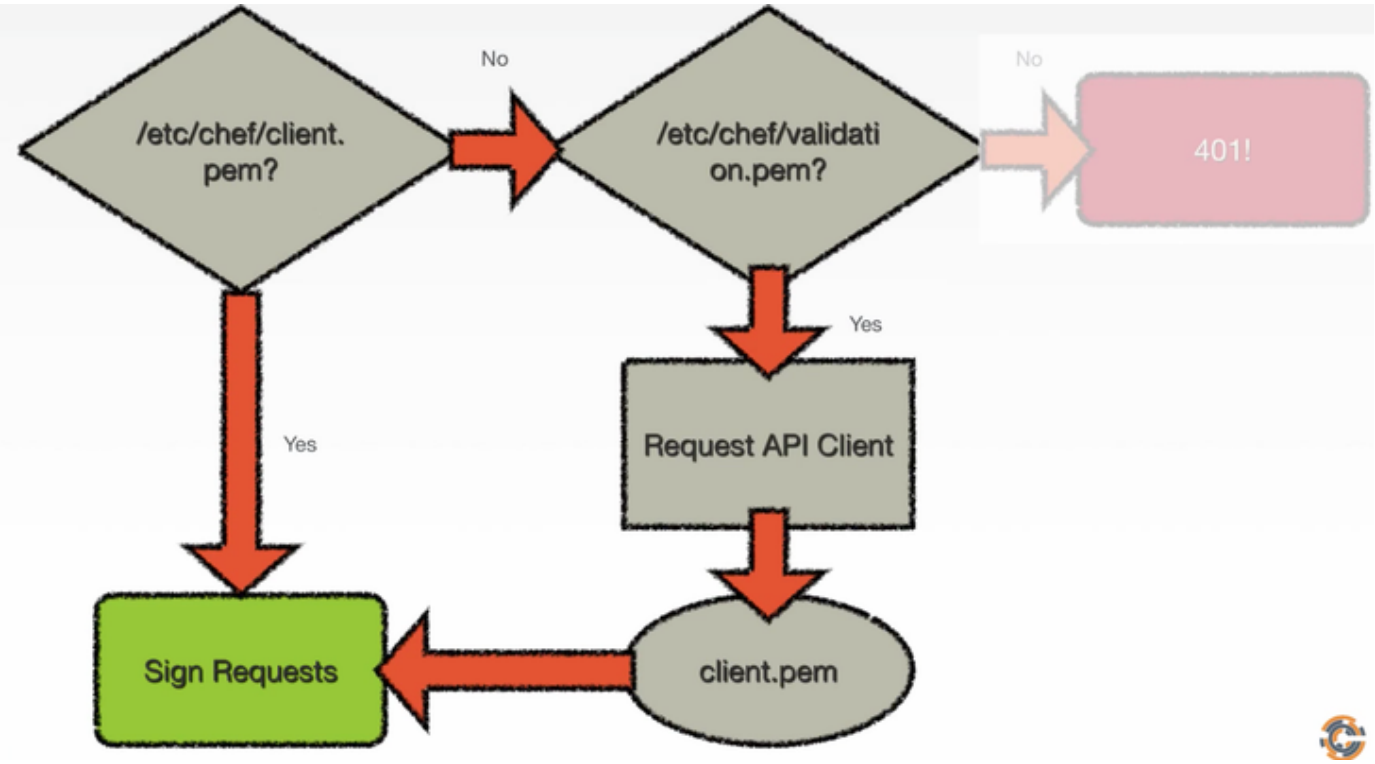
Chef server acts as a central repository for your cookbooks as well as for information about every node it manages.

Node is any computer that is managed by a Chef server. Every node has the Chef client installed on it. A node downloads the latest code from the Chef server and runs that code to bring your node's configuration up to date. **You might set up your node to check in periodically with the Chef server or update your node on demand when your configuration policy changes.**

Chef-client Workflow



Chef-client Security Model



Install and configure Chef server (Standalone)

Chef server acts as a central repository for cookbooks as well as for information about every node it manages.

There are three configuration scenarios for the Chef server:

- Standalone (everything on a single machine);
- High availability (machines configured for front-end and back-end, allowing for failover on the back-end and load-balancing on the front-end, as required);
- Tiered (machines configured for front-end and back-end, with a single back-end and load-balancing on the front-end, as required).

Install Chef Server

Chef Server - Stable Release: https://downloads.chef.io/chef-server

```
$ sudo yum install
https://packages.chef.io/files/stable/chef-server/12.15.6/el/7/chef-serv
er-core-[YOUR_VERSION].el7.x86_64.rpm
# Run the following to start all of the services:
$ sudo chef-server-ctl reconfigure
...
Chef Client finished, 492/1080 resources updated in 03 minutes 29
seconds
Chef Server Reconfigured!
# Create an administrator:
# chef-server-ctl user-create USER_NAME FIRST_NAME LAST_NAME EMAIL
'PASSWORD' --filename FILE_NAME
$ sudo chef-server-ctl user-create admin Ygor Almeida
ygor.almeida@organization.com '123456' --filename
/home/ygor/chef-server/ygor.pem
# Create an organization:
# chef-server-ctl org-create ORG_NAME ORG_FULL_NAME -f FILE_NAME
$ sudo chef-server-ctl org-create organization 'Organization'
--association_user admin --filename
/home/ygor/chef-server/org-validator.pem
```

Set up your Chef workstation on CentOS 7

Install Chef Development Kit

```
$ sudo yum install
https://packages.chef.io/files/stable/chefdk/1.3.43/el/7/chefdk-[YOUR_VE
RSION].el7.x86_64.rpm
$ sudo yum install git
$ chef --version
$ git --version
$ git config --global user.email "MY_EMAIL@myemail.com"
$ git config --global user.name "MY_USER"
```

Knife is the command-line tool that provides an interface between the workstation and the Chef server. Knife enables to upload cookbooks to the Chef server and work with nodes, the managed servers.

Knife requires two files to authenticate with the Chef server:

- **an RSA private key**.
  Every request to the Chef server is authenticated through an RSA public key pair. The Chef server holds the public part; you hold the private part.

- **a knife configuration file.**
  The configuration file is typically named knife.rb. It contains information such as the **Chef server's URL**, the location of **RSA private key**, and the default location of cookbooks. Both of these files are typically located in a directory named **.chef**. By default, every time knife runs, it looks in the current working directory for the .chef directory. If the .chef directory does not exist, knife searches up the directory tree for a .chef directory.

The **.chef directory** is used to store three files:

- knife.rb
- ORGANIZATION-validator.pem
- USER.pem

Set up the .chef directory

```
# An “application” supports multiple cookbooks. This part is for
LEARNING purposes ONLY!
$ sudo chef generate app chef-repo
$ mkdir -p /chef-repo/.chef
$ echo '.chef' >> chef-repo/.gitignore
# The pem files must be moved to the .chef directory after they are
downloaded from the Chef server. See above.
$ mv ygor.pem chef-repo/.chef
$ vi chef-repo/.chef/knife.rb
current_dir = File.dirname(__FILE__)
log_level                :info
log_location             STDOUT
node_name                'admin'
client_key               "#{current_dir}/ygor.pem"
chef_server_url
'https://MY_CHEF_SERVER/organizations/organization'
cache_type               'BasicFile'
cache_options( :path => "#{ENV['HOME']}/.chef/checksums" )
cookbook_path            ["#{current_dir}/../cookbooks"]
$ knife ssl fetch
$ knife ssl check
```

Upload a cookbook to Chef server

Most Chef users maintain their cookbooks in a version control system such as Git. However, also maintaining a copy of your cookbooks on the Chef server provides these benefits:

1. A central location for your cookbooks that's accessible from every node in your network.
2. A versioning mechanism that enables you to associate different cookbook versions among your nodes. This enables you to roll out new configuration policies only when you're ready.

```
$ cd chef-repo/cookbooks
# Clone the learn_chef_httpd cookbook from GitHub.
$ git clone https://github.com/learn-chef/learn_chef_httpd.git
```

In practice, you would typically run your cookbook on a temporary instance, such as a virtual machine, before you upload it to the Chef server. You might also run a series of quality tests to ensure your cookbook does what you expect.

```
$ knife cookbook upload learn_chef_httpd
$ knife cookbook list
learn_chef_httpd   0.1.0
```

Get a node to bootstrap

Now, it's time to bootstrap your node and run the learn_chef_httpd cookbook on it. Let's see how the bootstrap process triggers chef-client to run on your node remotely, from your workstation.

Knife bootstrap is the command to use to bootstrap a node. Key-based authentication is typically recommended over password authentication because it is more secure, over SSH, but you can bootstrap your node using either method. The **--node-name argument uniquely identifies the node with the Chef server**. Its value can be whatever you want. If it previously used the name node1-centos to bootstrap a different node, you can respond with 'Y' to overwrite that node's entry on your Chef server. Alternatively, you can choose a different name or remove the previous node from your Chef server. How to bootstrapping a Microsoft Azure instance: #azure-node.

The knife bootstrap command establishes an SSH connection to the node and installs chef-client.

```
# Copying your Public Key Using SSH to the Node
$ cat ~/.ssh/id_rsa.pub | ssh username@remote_host "mkdir -p ~/.ssh &&
cat >> ~/.ssh/authorized_keys && chmod 600 ~/.ssh/authorized_keys"
# From workstation
$ knife bootstrap ADDRESS --ssh-user USER --sudo --identity-file
IDENTITY_FILE --node-name node1-centos --run-list
'recipe[learn_chef_httpd]'
$ knife node list
$ knife node show node1-centos
$ curl MY_NODE_IP
```

First, the node was associated with your Chef server and, second, it did an initial check-in with the Chef server and ran the learn_chef_httpd cookbook. That's because you specifed learn_chef_httpd as the --run-list argument.

Update node's configuration

Bootstrapping is a one-time process. The knife ssh command enables to update node's configuration when your cookbook changes.

The bootstrap process downloaded and installed chef-client, downloaded the latest cookbooks, and executed the run-list. Chef provides information about nodes that you can access from your cookbooks. Now, we will update the home page to display your node's fully-qualified domain name (FQDN).

The Chef server created what's called a **node object**. This node object contains a number of attributes that describe the node, and these attributes are saved on the Chef server. When a recipe runs, a node object is loaded into the program. Chef loads the node's attributes from the Chef server into memory. **You can access these attributes from your Chef recipes.**

```
# From workstation
$ vi cookbooks/learn_chef_httpd/templates/index.html.erb
...
    <h1>hello from <%= node['fqdn'] %></h1>               --- <%= %>
syntax enables you to provide placeholders in your template file.
...
# Update cookbook's version metadata
$ vi metadata.rb
...
version '0.2.0'
...
$ knife cookbook upload learn_chef_httpd

Uploading learn_chef_httpd [0.2.0]
Uploaded 1 cookbook.
```

Most Chef cookbooks follow the Semantic Versioning scheme. Version numbers are typically written as MAJOR.MINOR.PATCH, where:

- **MAJOR specifies a change that's incompatible with previous versions.**
- **MINOR specifies new functionality that's backwards-compatible with previous versions.**
- **PATCH specifies backwards-compatible bug fixes.**

Update your node's configuration

The updated cookbook is now on the Chef server. The chef-client command pulls from Chef server the latest cookbooks from the node's run-list and applies the run-list to the node.

To run chef-client on the node remotely from your workstation, you can use the knife ssh command. A benefit to using the knife ssh command is that it enables to run chef-client (or any other command) on **multiple nodes at the same time, see** chef search.

If you're working with an Amazon EC2, or Microsoft Azure instance, replace the `ipaddress` part of the `--attribute ipaddress` argument with the corresponding entry from this table.

Cloud provider	Attribute	Notes
EC2	`cloud.public_hostname`	Chef sets this attribute during the bootstrap process.
Azure	`cloud.public_ip`	This is the attribute you set in the previous part when you bootstrapped your node.

For example, if you're working with an EC2 instance, you would specify `--attribute cloud.public_hostname`.

Update your node using key-based authentication

```
$ knife ssh 'name:node1-centos' 'sudo chef-client' --ssh-user USER
--identity-file IDENTITY_FILE --attribute ipaddress
# Verify the result
$ curl NODE_IP
```

It didn't have to specify the run-list because you already set that up when you bootstrapped the node.

How to run chef-client periodically

Chef users start out by using **knife to update their servers' configuration on demand**. It's possible to move to a continuous delivery system, such as Chef Automate – pricing. A continuous delivery system might run chef-client on your nodes when an explicit change is made to your configuration policy, for example, when a change is committed to your version control system.

One reason to run chef-client periodically is to help ensure that your servers are free from configuration drift. On Linux nodes, you might use a daemon, cron job, or service. Another way is to use the chef-client cookbook from Chef Supermarket. On Linux, the **chef-client cookbook sets up chef-client to run as a service**. Another benefit to using the chef-client cookbook is that it works on multiple platforms.

How often chef-client is run is controlled by two node attributes (source code):

- node['chef_client']['interval'] – interval specifies the number of seconds between chef-client runs. The default value is 1,800 (30 minutes).
- node['chef_client']['splay'] – splay specifies a maximum random number of seconds that is added to the interval. Splay helps balance the load on the Chef server by ensuring that many chef-client runs are not occurring at the same interval. The default value is 300 (5 minutes).

To update your node's run-list, you could use the knife node run_list set command. However, that does not set the appropriate node attributes. To accomplish both tasks, you'll use a role. **Roles enable you to focus on the function your node performs collectively rather than each of its individual components** (its run-list, node attributes, and so on). For example, you might have a web server role, a database role, or a load balancer role.

A common way is to create **a file (in JSON format) that describes your role** and then run the knife role from file command to upload that file to the Chef server. The advantage of creating a file is that you can store that file in a version control system such as Git.

Berkshelf is a tool that helps you resolve cookbook dependencies. Berkshelf can retrieve the cookbooks that your cookbook depends on and can upload your cookbooks to your Chef server. Berkshelf comes with the Chef DK.

```
$ cd chef-repo
$ berks install
$ ls ~/.berkshelf/cookbooks
# Run berks upload to upload the chef-client cookbook and its
dependencies to Chef server. For production, if possible use a trusted
SSL certificate.
$ berks upload --no-ssl-verify
$ mkdir chef-repo/roles
$ vi roles/web.json
{
   "name": "web",
   "description": "Web server role.",
   "json_class": "Chef::Role",
   "default_attributes": {
     "chef_client": {
       "interval": 300,
       "splay": 60
     }
   },
   "override_attributes": {
   },
   "chef_type": "role",
   "run_list": ["recipe[chef-client::default]",
                "recipe[chef-client::delete_validation]",
                "recipe[learn_chef_httpd::default]"
   ],
   "env_run_lists": {
   }
}
$ knife role from file roles/web.json
# To view the roles on your Chef server.
$ knife role list
# To view the role's details.
$ knife role show web
# set your node's run-list.
$ knife node run_list set node1-centos "role[web]"
$ knife node show node1-centos --run-list
```

The role sets the run-list to contain the chef-client cookbook as well as the learn_chef_httpd cookbook. chef-client::delete_validation recipe deletes the validation certificate from your node. This certificate is used during the bootstrap process to authorize the node to connect to the Chef server, and is no longer needed.

This time, replace the search query `'name:node1-centos'` with `'role:web'`. **If you had multiple nodes with the *web* role, `chef-client` would run on each of them.**

```
$ knife ssh 'role:web' 'sudo chef-client' --ssh-user USER
--identity-file ~/.ssh/private_key --attribute ipaddress
...
NODE_IP Recipe: chef-client::systemd_service ===> new service created!
...
```

As we can see from the output that the `chef-client` cookbook set up **`chef-client` as a service on your node**.

How to clean up your environment

Delete the node from the Chef server

```
# --yes argument suppresses any prompts for confirmation.
$ knife node delete node1-centos --yes
$ knife client delete node1-centos --yes

# Delete cookbooks from the Chef server. --all argument, you'll be
prompted to select which version to delete.
$ knife cookbook delete learn_chef_httpd --all --yes

# Delete the role from the Chef server.
$ knife role delete web --yes

# Delete the RSA private key from your node.
$ sudo rm /etc/chef/client.pem
```

Run knife node delete to remove the node's metadata from the Chef server and knife client delete to delete the entry (including the public part of the RSA key pair) from the Chef server's API client list.

During the bootstrap process, an RSA private key is generated on your node to enable your node to make API calls to the Chef server. The default location of this key is `/etc/chef/client.pem` on Linux systems.

References

- https://learn.chef.io/skills/
- https://docs.chef.io/
- https://learn.chef.io/modules/beyond-the-basics/attribute-precedence-explained