* Overview  Chef
* Chef Workstation
* Chef Habitat
* Chef InSpec
* Automate
* Extension APIs

**There are three configuration scenarios for the Chef Infra 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)

Front End(Node1) -> BackEnd(Node2) -> Load Balancing(Node3)

Master Server

**An Overview of Chef**

Chef Infra is a powerful automation platform that transforms infrastructure into code. Whether you’re operating in the cloud, on-premises, or in a hybrid environment, Chef automates how infrastructure is configured, deployed, and managed across your network, no matter its size.

* **Chef Workstation** is the location where users interact with Chef. With Chef Workstation, users can author and test cookbooks using tools such as Test Kitchen and interact with the Chef Infra Server using the knife and chef command line tools.
* **Chef Infra Client nodes** are the machines that are managed by Chef. The Chef Infra Client is installed on each node and is used to configure the node to its desired state.
* **Chef Infra Server** acts as a hub for configuration data. Chef Infra Server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by Chef. Nodes use the Chef Infra Client to ask the Chef Infra Server for configuration details, such as recipes, templates, and file distributions.

## Chef Components¶

The following diagram shows the relationships between the various elements of Chef, including the nodes, the server, and the workstation. These elements work together to provide the Chef Infra Client the information and instruction that it needs so that it can do its job. As you are reviewing the rest of this topic, use the icons in the tables to refer back to this image.

Chef has the following major components:

| **Component** | **Description** |
| --- | --- |
|  | One (or more) workstations are configured to allow users to author, test, and maintain cookbooks. Cookbooks are uploaded to the Chef Infra Server from the workstation. Some cookbooks are custom to the organization and others are based on community cookbooks available from the Chef Supermarket.  Ruby is the programming language that is the authoring syntax for cookbooks. Most recipes are simple patterns (blocks that define properties and values that map to specific configuration items like packages, files, services, templates, and users). The full power of Ruby is available for when you need a programming language.  Often, a workstation is configured to use Chef Workstation as the development toolkit. Chef Workstation is a package from Chef that provides a recommended set of tooling, including Chef itself, the chef command line tool, Test Kitchen, ChefSpec, and more. |
|  | A node is any machine—physical, virtual, cloud, network device, etc.—that is under management by Chef.  A Chef Infra Client is installed on every node that is under management by Chef. The Chef Infra Client performs all of the configuration tasks that are specified by the run-list and will pull down any required configuration data from the Chef Infra Server as it is needed during the Chef Infra Client run. |
|  | The Chef Infra Server acts as a hub of information. Cookbooks and policy settings are uploaded to the Chef Infra Server by users from workstations. (Policy settings may also be maintained from the Chef Infra Server itself, via the Chef management console web user interface.)  The Chef Infra Client accesses the Chef Infra Server from the node on which it’s installed to get configuration data, performs searches of historical Chef Infra Client run data, and then pulls down the necessary configuration data. After the Chef Infra Client run is finished, the Chef Infra Client uploads updated run data to the Chef Infra Server.  Chef management console is the user interface for the Chef Infra Server. It is used to manage data bags, attributes, run-lists, roles, environments, and cookbooks, and also to configure role-based access for users and groups. |
|  | Chef Supermarket is the location in which community cookbooks are shared and managed. Cookbooks that are part of the Chef Supermarket may be used by any Chef user. How community cookbooks are used varies from organization to organization. |

Chef management console, Chef Infra Client run reporting, high availability configurations, and Chef Infra Server replication are available as part of Chef Automate.

The following sections discuss these elements (and their various components) in more detail.

## Workstations¶

A workstation is a computer running Chef Workstation that is used to author cookbooks, interact with the Chef Infra Server, and interact with nodes.

The workstation is where users do most of their work, including:

* Developing and testing cookbooks and recipes
* Testing Chef code
* Keeping the Chef repository synchronized with version source control
* Configuring organizational policy, including defining roles and environments, and ensuring that critical data is stored in data bags
* Interacting with nodes, as (or when) required, such as performing a bootstrap operation

Chef Workstation gives you everything you need to get started with Chef — ad hoc remote execution, remote scanning, configuration tasks, cookbook creation tools as well as robust dependency and testing software — all in one easy-to-install package. Chef Workstation replaces ChefDK, combining all the existing features with new features, such as ad-hoc task support and the new Chef Workstation desktop application. Chef will continue to maintain ChefDK, but new development will take place in Chef Workstation without backporting features.

### Chef Workstation Components and Tools¶

Some important tools and components of Chef Workstation include:

| **Component** | **Description** |
| --- | --- |
|  | ChefDK, short for Chef Development Kit, is a package that contains everything that is needed to start using Chef:   * Chef Infra Client * chef and knife command line tools * Testing tools such as Test Kitchen, ChefSpec, Cookstyle, and Foodcritic * Chef InSpec * Everything else needed to author cookbooks and upload them to the Chef Infra Server |
|  | Chef Workstation includes important command-line tools:   * Chef Infra: Use the chef command-line tool to work with items in a chef-repo, which is the primary location in which cookbooks are authored, tested, and maintained, and from which policy is uploaded to the Chef Infra Server * Knife: Use the knife command-line tool to interact with nodes or work with objects on the Chef Infra Server * Chef Infra Client: an agent that configures your nodes * Test Kitchen: a testing harness for rapid validation of Chef code * Chef InSpec: Chef’s open source security & compliance automation framework * chef-run: a tool for running ad-hoc tasks * Chef Workstation App: for updating and managing your chef tools |
|  | The chef-repo is the repository structure in which cookbooks are authored, tested, and maintained:   * Cookbooks contain recipes, attributes, custom resources, libraries, files, templates, tests, and metadata * The chef-repo should be synchronized with a version control system (such as git), and then managed as if it were source code   The directory structure within the chef-repo varies. Some organizations prefer to keep all of their cookbooks in a single chef-repo, while other organizations prefer to use a chef-repo for every cookbook. |
|  | Use Test Kitchen to automatically test cookbook data across any combination of platforms and test suites:   * Defined in a kitchen.yml file. See the configuration documentation for options and syntax information. * Uses a driver plugin architecture * Supports cookbook testing across many cloud providers and virtualization technologies * Supports all common testing frameworks that are used by the Ruby community * Uses a comprehensive set of base images provided by Bento |
|  | Use ChefSpec to simulate the convergence of resources on a node:   * Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby * Is the fastest way to test resources and recipes |

## Cookbooks¶

A cookbook is the fundamental unit of configuration and policy distribution. A cookbook defines a scenario and contains everything that is required to support that scenario:

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

The Chef Infra Client uses Ruby as its reference language for creating cookbooks and defining recipes, with an extended DSL for specific resources. A reasonable set of resources are available to the Chef Infra Client, enough to support many of the most common infrastructure automation scenarios; however, this DSL can also be extended when additional resources and capabilities are required.

### Components¶

Cookbooks are comprised of the following components:

| **Component** | **Description** |
| --- | --- |
|  | An attribute can be defined in a cookbook (or a recipe) and then used to override the default settings on a node. When a cookbook is loaded during a Chef Infra Client run, these attributes are compared to the attributes that are already present on the node. Attributes that are defined in attribute files are first loaded according to cookbook order. For each cookbook, attributes in the default.rb file are loaded first, and then additional attribute files (if present) are loaded in lexical sort order. When the cookbook attributes take precedence over the default attributes, the Chef Infra Client will apply those new settings and values during the Chef Infra Client run on the node. |
|  | Use the **cookbook\_file** resource to transfer files from a sub-directory of COOKBOOK\_NAME/files/ to a specified path located on a host that is running the Chef Infra Client. The file is selected according to file specificity, which allows different source files to be used based on the hostname, host platform (operating system, distro, or as appropriate), or platform version. Files that are located in the COOKBOOK\_NAME/files/default sub-directory may be used on any platform. |
|  | A library allows arbitrary Ruby code to be included in a cookbook. The most common use for libraries is to write helpers that are used throughout recipes and custom resources. A library file is a Ruby file that is located within a cookbook’s /libraries directory. Because a library is built using Ruby, anything that can be done with Ruby can be done in a library file, including advanced functionality such as extending built-in Chef classes. |
|  | Every cookbook requires a small amount of metadata. A file named metadata.rb is located at the top of every cookbook directory structure. The contents of the metadata.rb file provides information that helps Chef Infra Client and Server correctly deploy cookbooks to each node. |
|  | A recipe is the most fundamental configuration element within the organization. A recipe:   * Is authored using Ruby, which is a programming language designed to read and behave in a predictable manner * Is mostly a collection of resources, defined using patterns (resource names, attribute-value pairs, and actions); helper code is added around this using Ruby, when needed * Must define everything that is required to configure part of a system * Must be stored in a cookbook * May be included in another recipe * May use the results of a search query and read the contents of a data bag (including an encrypted data bag) * May have a dependency on one (or more) recipes * Must be added to a run-list before it can be used by the Chef Infra Client * Is always executed in the same order as listed in a run-list   The Chef Infra Client will run a recipe only when asked. When the Chef Infra Client runs the same recipe more than once, the results will be the same system state each time. When a recipe is run against a system, but nothing has changed on either the system or in the recipe, the Chef Infra Client won’t change anything.  The Recipe DSL is a Ruby DSL that is primarily used to declare resources from within a recipe. The Recipe DSL also helps ensure that recipes interact with nodes (and node properties) in the desired manner. Most of the methods in the Recipe DSL are used to find a specific parameter and then tell the Chef Infra Client what action(s) to take, based on whether that parameter is present on a node. |
|  | A resource is a statement of configuration policy that:   * Describes the desired state for a configuration item * Declares the steps needed to bring that item to the desired state * Specifies a resource type—such as package, template, or service * Lists additional details (also known as resource properties), as necessary * Are grouped into recipes, which describe working configurations   Chef has many built-in resources that cover all of the most common actions across all of the most common platforms. You can build your own resources to handle any situation that isn’t covered by a built-in resource. |
|  | A cookbook template is an Embedded Ruby (ERB) template that is used to dynamically generate static text files. Templates may contain Ruby expressions and statements, and are a great way to manage configuration files. Use the **template** resource to add cookbook templates to recipes; place the corresponding Embedded Ruby (ERB) template file in a cookbook’s /templates directory. |
|  | Testing cookbooks improves the quality of those cookbooks by ensuring they are doing what they are supposed to do and that they are authored in a consistent manner. Unit and integration testing validates the recipes in cookbooks. Syntax testing—often called linting—validates the quality of the code itself. The following tools are popular tools used for testing Chef recipes: Test Kitchen, ChefSpec, and Foodcritic. |

## Nodes¶

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

### Node Types¶

The types of nodes that can be managed by Chef include, but are not limited to, the following:

| **Node Type** | **Description** |
| --- | --- |
|  | A physical node is typically a server or a virtual machine, but it can be any active device attached to a network that is capable of sending, receiving, and forwarding information over a communications channel. In other words, a physical node is any active device attached to a network that can run a Chef Infra Client and also allow that Chef Infra Client to communicate with a Chef Infra Server. |
|  | A cloud-based node is hosted in an external cloud-based service, such as Amazon Web Services (AWS), OpenStack, Rackspace, Google Compute Engine, or Microsoft Azure. Plugins are available for knife that provide support for external cloud-based services. knife can use these plugins to create instances on cloud-based services. Once created, the Chef Infra Client can be used to deploy, configure, and maintain those instances. |
|  | A virtual node is a machine that runs only as a software implementation, but otherwise behaves much like a physical machine. |
|  | A network node is any networking device—a switch, a router—that is being managed by a Chef Infra Client, such as networking devices by Juniper Networks, Arista, Cisco, and F5. Use Chef to automate common network configurations, such as physical and logical Ethernet link properties and VLANs, on these devices. |
|  | Containers are an approach to virtualization that allows a single operating system to host many working configurations, where each working configuration—a container—is assigned a single responsibility that is isolated from all other responsibilities. Containers are popular as a way to manage distributed and scalable applications and services. |

### Chef on Nodes¶

The key components of nodes that are under management by Chef include:

| **Component** | **Description** |
| --- | --- |
|  | Chef Infra Client is an agent that runs locally on every node that is under management by Chef Infra Server. When a Chef Infra Client is run, it will perform all of the steps that are required to bring the node into the expected state, including:   * Registering and authenticating the node with the Chef Infra Server * Building the node object * Synchronizing cookbooks * Compiling the resource collection by loading each of the required cookbooks, including recipes, attributes, and all other dependencies * Taking the appropriate and required actions to configure the node * Looking for exceptions and notifications, handling each as required   RSA public key-pairs are used to authenticate the Chef Infra Client with the Chef Infra Server every time a Chef Infra Client needs access to data that is stored on the Chef Infra Server. This prevents any node from accessing data that it shouldn’t and it ensures that only nodes that are properly registered with the Chef Infra Server can be managed. |
|  | Ohai is a tool that is used to collect system configuration data, which is provided to the Chef Infra Client for use within cookbooks. Ohai is run by the Chef Infra Client at the beginning of every Chef run to determine system state. Ohai includes many built-in plugins to detect common configuration details as well as a plugin model for writing custom plugins.  The types of attributes Ohai collects include but are not limited to:   * Operating System * Network * Memory * Disk * CPU * Kernel * Host names * Fully qualified domain names * Virtualization * Cloud provider metadata   Attributes that are collected by Ohai are automatic level attributes, in that these attributes are used by the Chef Infra Client to ensure that these attributes remain unchanged after the Chef Infra Client is done configuring the node. |

## The Chef Server¶

The Chef Infra Server acts as a hub for configuration data. The Chef Infra Server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the Chef Infra Client. Nodes use the Chef Infra Client to ask the Chef Infra Server for configuration details, such as recipes, templates, and file distributions. The Chef Infra Client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef Infra Server). This scalable approach distributes the configuration effort throughout the organization.

| **Feature** | **Description** |
| --- | --- |
|  | Search indexes allow queries to be made for any type of data that is indexed by the Chef Infra Server, including data bags (and data bag items), environments, nodes, and roles. A defined query syntax is used to support search patterns like exact, wildcard, range, and fuzzy. A search is a full-text query that can be done from several locations, including from within a recipe, by using the search subcommand in knife, the search method in the Recipe DSL, the search box in the Chef management console, and by using the /search or /search/INDEX endpoints in the Chef Infra Server API. The search engine is based on Apache Solr and is run from the Chef Infra Server. |
|  | Chef management console is a web-based interface for the Chef Infra Server that provides users a way to manage the following objects:   * Nodes * Cookbooks and recipes * Roles * Stores of JSON data (data bags), including encrypted data * Environments * Searching of indexed data * User accounts and user data for the individuals who have permission to log on to and access the Chef server |
|  | Data bags store global variables as JSON data. Data bags are indexed for searching and can be loaded by a cookbook or accessed during a search. |
|  | Policy defines how business and operational requirements, processes, and production workflows map to objects that are stored on the Chef Infra Server. Policy objects on the Chef Infra Server include roles, environments, and cookbook versions. |

### Policy¶

Policy maps business and operational requirements, process, and workflow to settings and objects stored on the Chef Infra Server:

* Roles define server types, such as “web server” or “database server”
* Environments define process, such as “dev”, “staging”, or “production”
* Certain types of data—passwords, user account data, and other sensitive items—can be placed in data bags, which are located in a secure sub-area on the Chef Infra Server that can only be accessed by nodes that authenticate to the Chef Infra Server with the correct SSL certificates
* The cookbooks (and cookbook versions) in which organization-specific configuration policies are maintained

Some important aspects of policy include:

| **Feature** | **Description** |
| --- | --- |
|  | A role is a way to define certain patterns and processes that exist across nodes in an organization as belonging to a single job function. Each role consists of zero (or more) attributes and a run-list. Each node can have zero (or more) roles assigned to it. When a role is run against a node, the configuration details of that node are compared against the attributes of the role, and then the contents of that role’s run-list are applied to the node’s configuration details. When a Chef Infra Client runs, it merges its own attributes and run-lists with those contained within each assigned role. |
|  | An environment is a way to map an organization’s real-life workflow to what can be configured and managed when using Chef Infra Server. Every organization begins with a single environment called the \_default environment, which cannot be modified (or deleted). Additional environments can be created to reflect each organization’s patterns and workflow. For example, creating production, staging, testing, and developmentenvironments. Generally, an environment is also associated with one (or more) cookbook versions. |
|  | A cookbook version represents a set of functionality that is different from the cookbook on which it is based. A version may exist for many reasons, such as ensuring the correct use of a third-party component, updating a bug fix, or adding an improvement. A cookbook version is defined using syntax and operators, may be associated with environments, cookbook metadata, and/or run-lists, and may be frozen (to prevent unwanted updates from being made).  A cookbook version is maintained just like a cookbook, with regard to source control, uploading it to the Chef Infra Server, and how the Chef Infra Client applies that cookbook when configuring nodes. |
|  | A run-list defines all of the information necessary for Chef to configure a node into the desired state. A run-list is:   * An ordered list of roles and/or recipes that are run in the exact order defined in the run-list; if a recipe appears more than once in the run-list, the Chef Infra Client will not run it twice * Always specific to the node on which it runs; nodes may have a run-list that is identical to the run-list used by other nodes * Stored as part of the node object on the Chef server * Maintained using knife and then uploaded from the workstation to the Chef Infra Server, or maintained using Chef Automate |

### Linux¶

1. Visit the ChefDK downloads page and download the appropriate package for your distribution:
2. wget https://packages.chef.io/files/stable/chefdk/3.2.30/ubuntu/18.04/chefdk\_3.2.30-1\_amd64.deb
3. Use your distribution’s package manager to install ChefDK:
   * Red Hat Enterprise Linux:
   * rpm -Uvh chefdk-3.2.30-1.el7.x86\_64.rpm
   * Debian/Ubuntu:

dpkg -i chefdk\_3.2.30-1\_amd64.deb

## Configure Ruby Environment¶

For many users of Chef, the version of Ruby that is included in ChefDK should be configured as the default version of Ruby.

**Note**

These instructions are intended for macOS and Linux users. For instructions on setting up your Ruby environment on Windows, see ChefDK on Windows.

1. Open a command window and enter the following:
2. $ which ruby

which will return something like /usr/bin/ruby.

1. To use ChefDK version of Ruby as the default Ruby, edit the $PATH and GEM environment variables to include paths to ChefDK. For example, on a machine that runs Bash, run:
2. echo 'eval "$(chef shell-init bash)"' >> ~/.bash\_profile

where bash and ~/.bash\_profile represents the name of the shell.

If zsh is your preferred shell then run the following:

echo 'eval "$(chef shell-init zsh)"' >> ~/.zshrc

1. Run which ruby again. It should return /opt/chefdk/embedded/bin/ruby.

**Note**

Using ChefDK-provided Ruby as your system Ruby is optional. For many users, Ruby is primarily used for authoring Chef cookbooks and recipes. If that’s true for you, then using the ChefDK-provided Ruby is recommended.

## Add Ruby to $PATH¶

The Chef Infra Client includes a stable version of Ruby as part of its installer. The path to this version of Ruby must be added to the $PATH environment variable and saved in the configuration file for the command shell (Bash, csh, and so on) that is used on the machine running ChefDK. In a command window, type the following:

echo 'export PATH="/opt/chefdk/embedded/bin:$PATH"' >> ~/.configuration\_file && source ~/.configuration\_file

where configuration\_file is the name of the configuration file for the specific command shell. For example, if Bash were the command shell and the configuration file were named bash\_profile, the command would look something like the following:

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

**Warning**

On Microsoft Windows, C:/opscode/chefdk/bin must be before C:/opscode/chefdk/embedded/bin in the PATH.

## Install Git¶

An open source distributed version control system called Git must be installed before the chef-repo can be cloned to ChefDK machine from GitHub.

To install Git:

1. Go to the following URL: https://help.github.com/articles/set-up-git.
2. Follow the directions, install Git (https://git-scm.com/downloads), and then complete the remaining configuration steps on that page.

**Note**

It is not necessary to create or fork a repository in order to clone the chef-repo from GitHub.

## Create the Chef repository¶

Use the chef generate repo to create the Chef repository. For example, to create a repository called chef-repo:

chef generate repo chef-repo

### Create .chef Directory¶

The .chef directory is used to store three files:

* config.rb
* ORGANIZATION-validator.pem
* USER.pem

Where ORGANIZATION and USER represent strings that are unique to each organization. These files must be present in the .chef directory in order for ChefDK to be able to connect to a Chef Infra Server.

To create the .chef directory:

1. In a command window, enter the following:
2. mkdir -p ~/chef-repo/.chef

Note that you’ll need to replace chef-repo with the name of the repository you created previously.

1. After the .chef directory has been created, the following folder structure will be present on the local machine:
2. chef-repo/
3. .chef/ << the hidden directory
4. certificates/
5. config/
6. cookbooks/
7. data\_bags
8. environments/
9. roles/
10. Add .chef to the .gitignore file to prevent uploading the contents of the .chef folder to GitHub. For example:
11. $ echo '.chef' >> ~/chef-repo/.gitignore

### Starter Kit¶

If you have access to Chef Infra Server through Automate or Chef Manage, you can download the starter kit. The starter kit will create the necessary configuration files: the .chef directory, config.rb, ORGANIZATION-validator.pem, and USER.pem. Simply download the starter kit and then move it to the desired location on your ChefDK machine.

## Configure the Chef Repository¶

### With WebUI¶

Use the following steps to manually set up the chef-repo and to use the Chef management console to get the .pemand config.rb files.

#### Get Config Files¶

For a ChefDK installation that will interact with the Chef Infra Server (including the hosted Chef Infra Server), log on and download the following files:

* config.rb. This configuration file can be downloaded from the **Organizations** page.
* ORGANIZATION-validator.pem. This private key can be downloaded from the **Organizations** page.
* USER.pem. This private key can be downloaded from the **Change Password** section of the **Account Management** page.

#### Move Config Files¶

The config.rb, ORGANIZATION-validator.pem, and USER.pem files must be moved to the .chef directory after they are downloaded from the Chef Infra Server.

To move files to the .chef directory:

1. In a command window, enter each of the following:
2. cp /path/to/config.rb ~/chef-repo/.chef

and:

cp /path/to/ORGANIZATION-validator.pem ~/chef-repo/.chef

and:

cp /path/to/USERNAME.pem ~/chef-repo/.chef

where /path/to/ represents the path to the location in which these three files were placed after they were downloaded.

1. Verify that the files are in the .chef folder.

### Without WebUI¶

Use the following steps to manually set up the Chef repository: On your Chef Infra Server, create the ORGANIZATION-validator.pem and USER.pem files with the chef-server-ctl command line tool. Then, on your workstation create the config.rb file with the knife tool.

#### Create an Organization¶

On the Chef Infra Server machine create the ORGANIZATION-validator.pem from the command line using chef-server-ctl. Run the following command:

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

where

* The name must begin with a lower-case letter or digit, may only contain lower-case letters, digits, hyphens, and underscores, and must be between 1 and 255 characters. For example: chef
* The full name must begin with a non-white space character and must be between 1 and 1023 characters. For example: "Chef Software, Inc."
* -f FILE\_NAME: Write the ORGANIZATION-validator.pem to FILE\_NAME instead of printing it to STDOUT. For example: /tmp/chef.key.

For example, an organization named chef, with a full name of Chef Software, Inc., and with the ORGANIZATION-validator.pem file saved to /tmp/chef.key:

$ chef-server-ctl org-create chef "Chef Software, Inc." -f /tmp/chef.key

#### Create a User¶

On the Chef Infra Server machine create the USER.pem from the command line using chef-server-ctl. Run the following command:

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

where

* -f FILE\_NAME writes the USER.pem to a file instead of STDOUT. For example: /tmp/grantmc.key.

For example: a user named grantmc, with a first and last name of Grant McLennan, an email address of grantmc@chef.io, a poorly-chosen password, and a USER.pem file saved to /tmp/grantmc.key:

$ chef-server-ctl user-create grantmc Grant McLennan grantmc@chef.io p@s5w0rD! -f /tmp/grantmc.key

#### Move .pem Files¶

Download the ORGANIZATION-validator.pem and USER.pem files from the Chef Infra Server and move them to the .chef directory.

To move files to the .chef directory:

1. In a command window, enter each of the following:
2. cp /path/to/ORGANIZATION-validator.pem ~/chef-repo/.chef

and:

cp /path/to/USERNAME.pem ~/chef-repo/.chef

where /path/to/ represents the path to the location in which these three files were placed after they were downloaded.

1. Verify that the files are in the .chef folder.

#### Create the config.rb File¶

Navigate to the ~/chef-repo/.chef directory and create the config.rb using the knife configure tool. The file must be created in the .chef folder. It should look similar to:

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

log\_level :info

log\_location STDOUT

node\_name 'node\_name'

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

validation\_client\_name 'ORG\_NAME-validator'

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

chef\_server\_url 'https://api.chef.io/organizations/ORG\_NAME'

cache\_type 'BasicFile'

cache\_options( :path => "**#{**ENV['HOME']**}**/.chef/checksums" )

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

At a minimum, you must update the following settings with the appropriate values:

* client\_key should point to the location of the Chef Infra Server user’s .pem file on your ChefDK machine.
* validation\_client\_name should be updated with the name of the desired organization that was created on the Chef Infra Server.
* validation\_key should point to the location of your organization’s .pem file on your ChefDK machine.
* chef\_server\_url must be updated with the domain or IP address used to access the Chef Infra Server.

See the knife config.rb documentation for more details.

## Get SSL Certificates¶

Chef Server 12 enables SSL verification by default for all requests made to the server, such as those made by knife and the Chef Infra Client. The certificate that is generated during the installation of the Chef Infra Server is self-signed, which means there isn’t a signing certificate authority (CA) to verify. In addition, this certificate must be downloaded to any machine from which knife and/or the Chef Infra Client will make requests to the Chef Infra Server.

Use the knife ssl fetch subcommand to pull the SSL certificate down from the Chef Infra Server:

knife ssl fetch

See SSL Certificates for more information about how knife and the Chef Infra Client use SSL certificates generated by the Chef Infra Server.

## Verify Install¶

The ChefDK is installed correctly when it is able to use knife to communicate with the Chef Infra Server.

To verify that ChefDK can connect to the Chef Infra Server:

1. In a command window, navigate to the Chef repository:
2. cd ~/chef-repo
3. In a command window, enter the following:
4. knife client list

to return a list of clients (registered nodes and ChefDK installations) that have access to the Chef Infra Server. For example:

chefdk\_machine

registered\_node

Let us write a simple program in ruby. All ruby files will have extension **.rb**. So, put the following source code in a test.rb file.

 Live Demo

#!/usr/bin/ruby -w

puts "Hello, Ruby!";

Here, we assumed that you have Ruby interpreter available in /usr/bin directory. Now, try to run this program as follows −

$ ruby test.rb

This will produce the following result −

Hello, Ruby!