Chef Introduction

Infrastructure Automation

- *Infrastructure automation* is the process of scripting environments — from installing an operating system,

  to installing and configuring servers on instances,

  to configuring how the instances and

  software communicate with one another,

  and much more.

By scripting environments, you can apply the same configuration to a single node or to thousands.

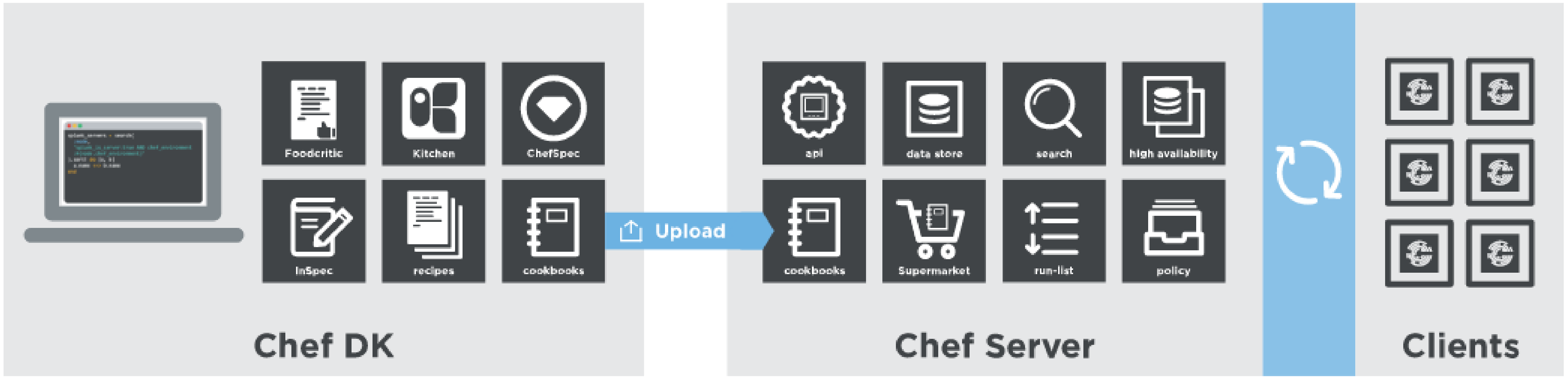
Infrastructure Automation

- Infrastructure automation also goes by other names:
  - configuration management,
  - IT management,
  - provisioning, scripted infrastructures,
  - system configuration management, and
  - many other overlapping terms.
- The point is the same: you are describing your infrastructure and its configuration as a script or set of scripts so that environments can be replicated in a much less error-prone manner.
- Infrastructure automation brings agility to both development and operations because any authorized team member can modify the scripts while applying good *development* practices — such as automated testing and versioning — to your infrastructure.

What is Chef?

- Chef 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 Components



Chef Workstation

- The workstation is the location from which all of Chef is managed,
  - including installing the Chef DK,
  - authoring cookbooks, and
  - using tools like
    - Kitchen,
    - chef-zero (a command-line tool that runs locally as if it were connected to a real Chef server),
    - Knife (Command line for interacting with the Chef server)
    - chef (for interacting with your local chef-repo),
    - core Chef resources (for building recipes) and
    - InSpec (for building security and compliance checks into your workflow).

Workstations

- A workstation is a computer that is configured to run various Chef command-line tools that synchronize with a chef-repo, author cookbooks, interact with the Chef server, and interact with nodes.
- The workstation is the location from which most users do most of their work, including:
  - Developing cookbooks and recipes (and authoring them using Ruby syntax and patterns)
  - Keeping the chef-repo synchronized with version source control
  - Using command-line tools
  - 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

components of workstations

Chef development kit

- The Chef development kit is a package that contains everything that is needed to start using Chef:
  - chef-client
  - chef
  - Ohai
  - chef-zero
  - Testing tools like Kitchen, ChefSpec, Cookstyle, and Foodcritic
  - Chef provisioning
  - Everything else needed to author cookbooks and upload them to the Chef server

command-line tools

- 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 server
- Use the knife command-line tool to interact with nodes or work with objects on the Chef server

Chef-repo

- The chef-repo is the repository structure in which cookbooks are authored, tested, and maintained:
- Cookbooks contain recipes, attributes, custom resources, libraries, definitions, 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.

Kitchen

- Use Kitchen to automatically test cookbook data across any combination of platforms and test suites:
  - Defined in a .kitchen.yml file
  - 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

ChefSpec

- Use ChefSpec to simulate the convergence of resources on a node:
  - Runs the chef-client on a local machine
  - Is an extension of RSpec, a behavior-driven development (BDD) framework for Ruby
  - Is the fastest way to test resources and recipes

Chef Node

Chef Node

- Nodes are the machines—
  - physical,
  - virtual,
  - cloud, and so on—
  - that are under management by Chef.
- The chef-client is installed on each node and is what performs the automation on that machine.

Physical Node

- 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-client and also allow that chef-client to communicate with a Chef server.

Containers

- 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.

Cloud Node

- 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-client can be used to deploy, configure, and maintain those instances.

Networking Device Node

- A network node is any networking device—a switch, a router—that is being managed by a chef-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.

Components of Chef Node

Chef Client

- A chef-client is an agent that runs locally on every node that is under management by Chef.
- When a chef-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 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-client with the Chef server every time a chef-client needs access to data that is stored on the Chef server.

Ohai

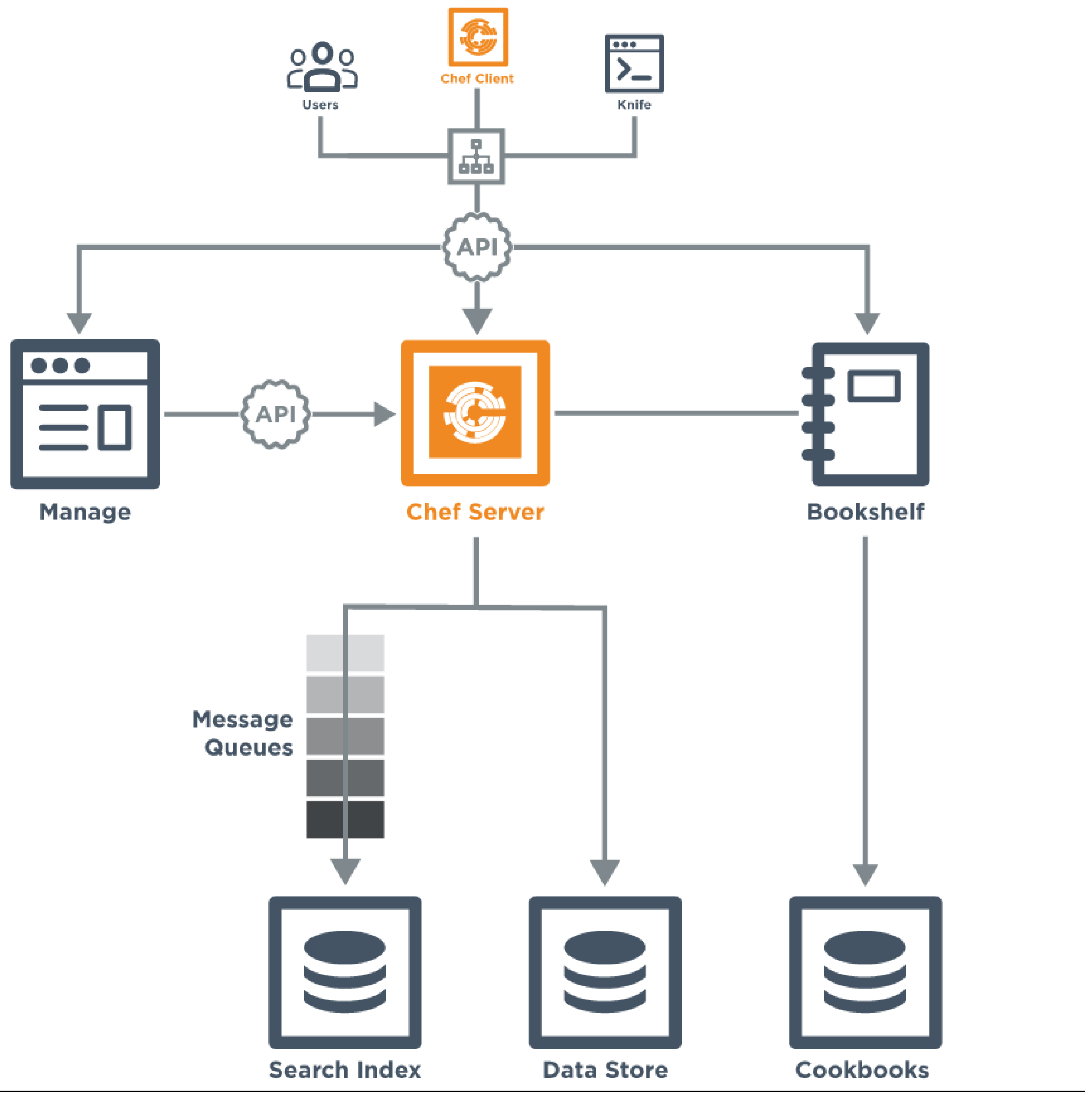
- Ohai is a tool that is used to detect attributes on a node, and then provide these attributes to the chef-client at the start of every chef-client run.
- Ohai is required by the chef-client and must be present on a node.
- The types of attributes Ohai collects include (but are not limited to):
  - Platform details
  - Network usage
  - Memory usage
  - CPU data
  - Kernel data
  - Host names
  - Fully qualified domain names
  - Other configuration details

Chef Server

Chef Server

- Use the Chef server as your foundation to create and manage flexible, dynamic infrastructure whether you manage 50 or 500,000 nodes, across multiple datacenters, public and private clouds, and in heterogeneous environments.
- The Chef server acts as a hub for configuration data.
- The Chef server stores cookbooks, the policies that are applied to nodes, and metadata that describes each registered node that is being managed by the chef-client.
- Nodes use the chef-client to ask the Chef server for configuration details, such as recipes, templates, and file distributions.
- The chef-client then does as much of the configuration work as possible on the nodes themselves (and not on the Chef server).
- This scalable approach distributes the configuration effort throughout the organization.

Chef Server Architecture



Chef Manage

- chef-server-webui is a Ruby on Rails 3.0 application that hosts the web interface for the Chef server.
- The Chef management console uses the Chef server API for all communication to the Chef server.

Chef Server

- chef is a complete written on Erlang.
- It exposes the API t communicate with client and Web UI
- Which Manages Bookshelf, Data Store, Search Indexes.

Bookshelf

- Bookshelf is used to store cookbook content—files, templates, and so on—that have been uploaded to the Chef server as part of a cookbook version.
- Cookbook content is stored by content checksum.
- If two different cookbooks or different versions of the same cookbook include the same file or template, Bookshelf will store that file only once. T
- he cookbook content managed by Bookshelf is stored in flat files and is separated from the Chef server and search index repositories.
- All cookbooks are stored in a dedicated repository.

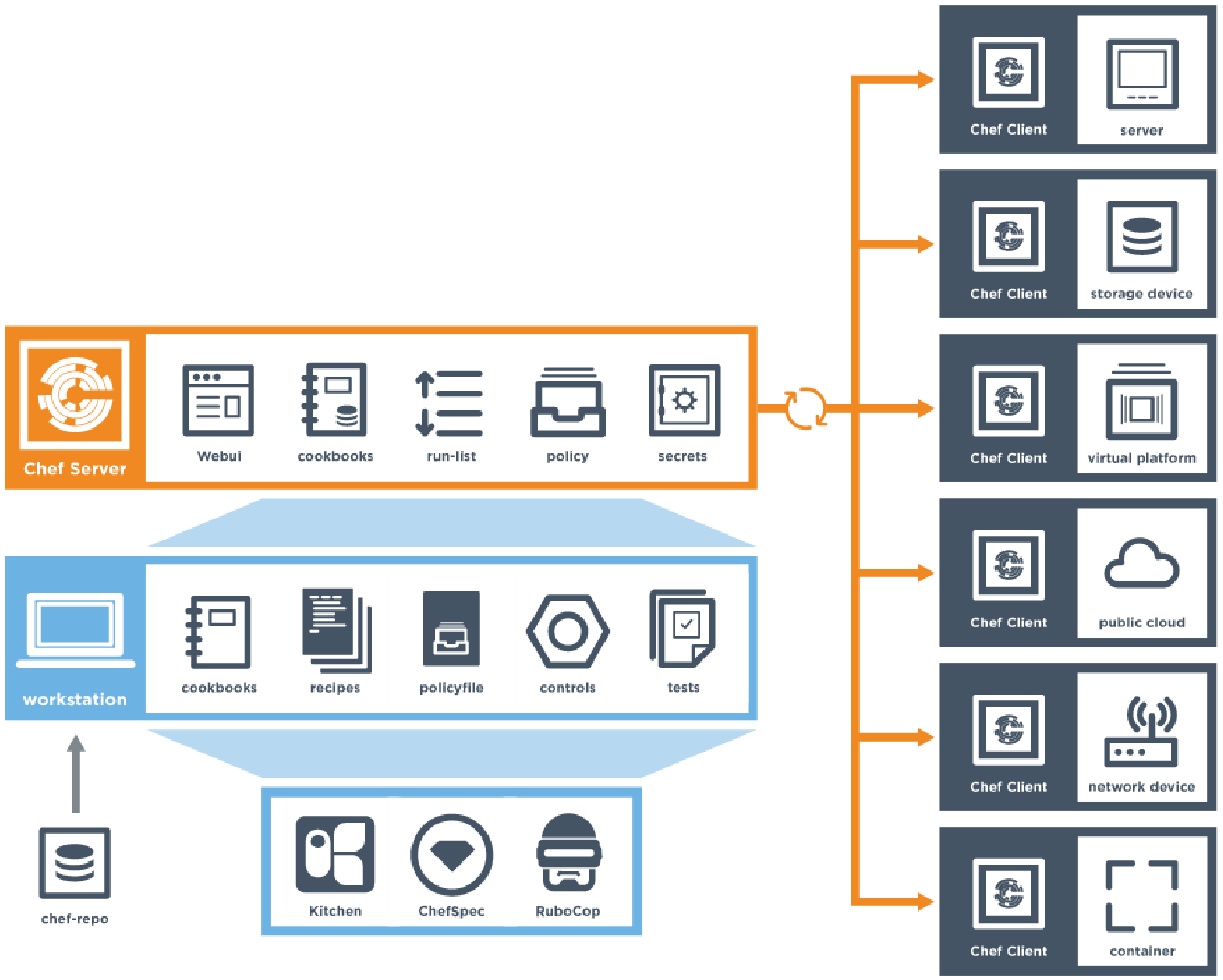
Message Queues

- Messages are sent to the search index using the following components:
- RabbitMQ is used as the message queue for the Chef server. All items that will be added to the search index repository are first added to a queue.
- chef-expander is used to pull messages from the RabbitMQ queue, process them into the required format, and then post them to chef-solr for indexing.
- chef-solr wraps Apache Solr and exposes its REST API for indexing and search.
- All messages are added to a dedicated search index repository.

PostgreSQL

- PostgreSQL is the data storage repository for the Chef server.
- This represents the independently configured set of servers that are running PostgreSQL and are configured to act as the data store for the Chef server

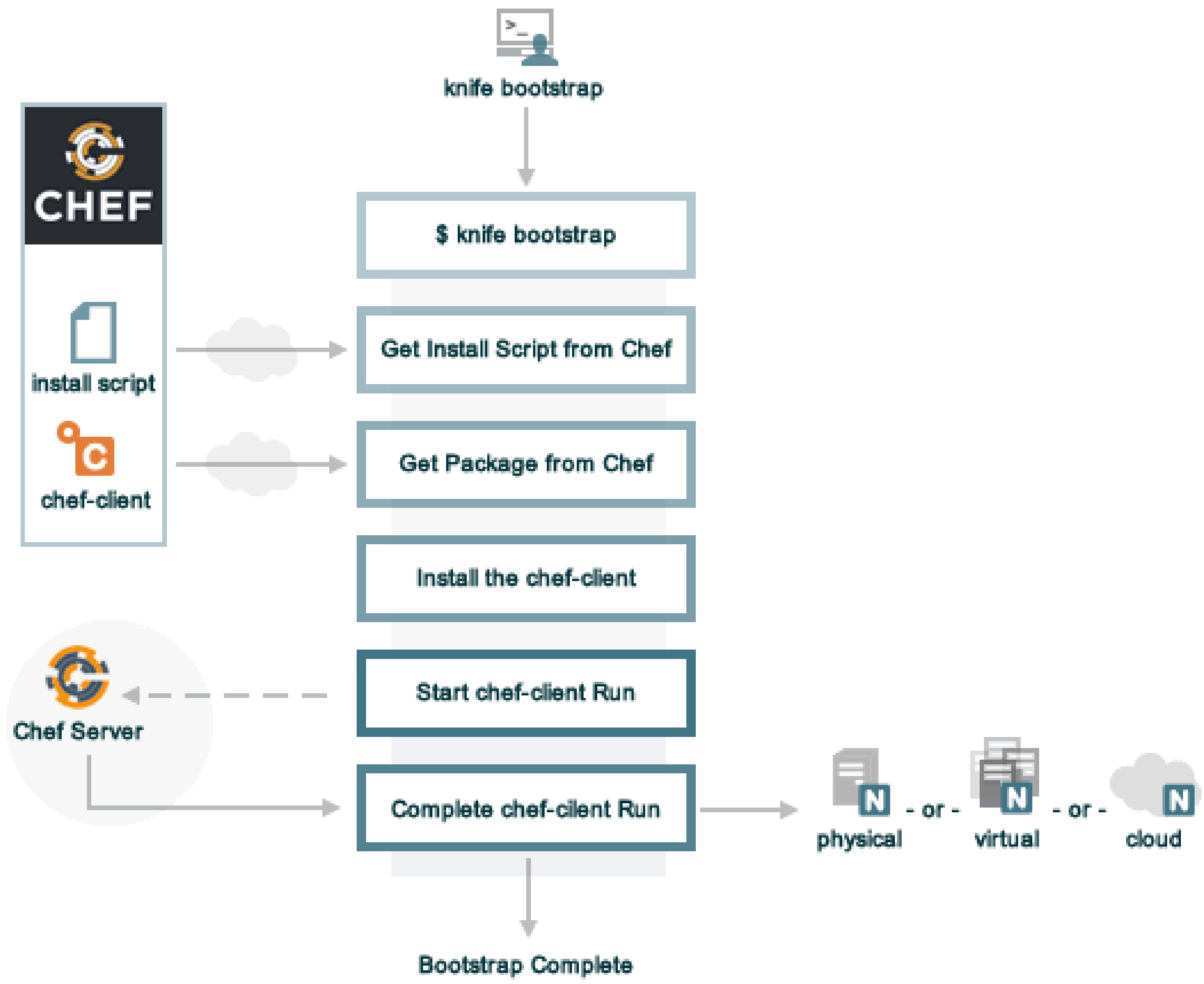
Chef components interaction



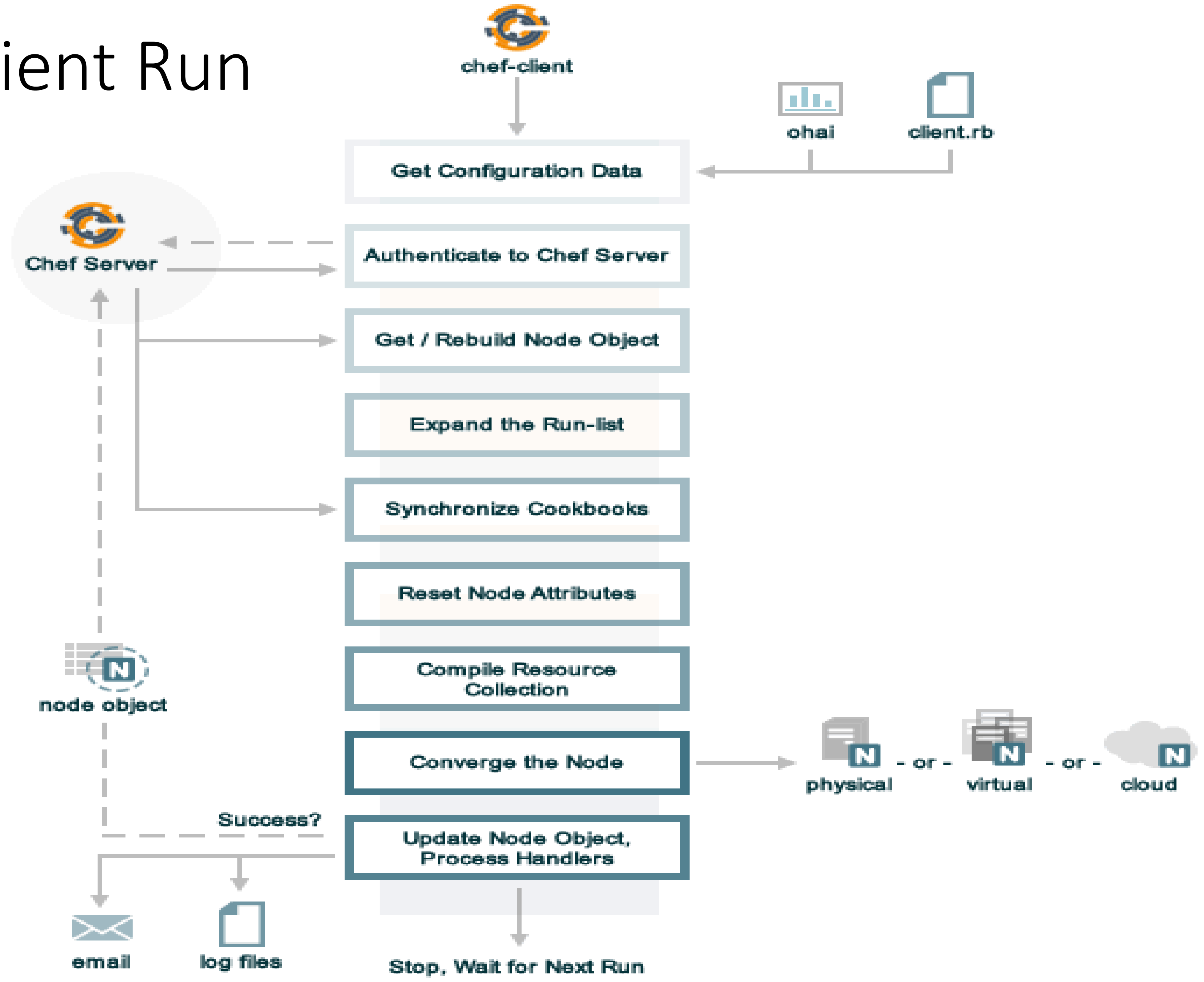
Knife Boot Strap

Knife Boot Strap

- Installs Chef Client
- Copies Chef Server URL
- Copies Validation_Client Name
- Copies Validation_client_key
- Client Register node to Server
- Save node detail in server



Chef Client Run



Get configuration data

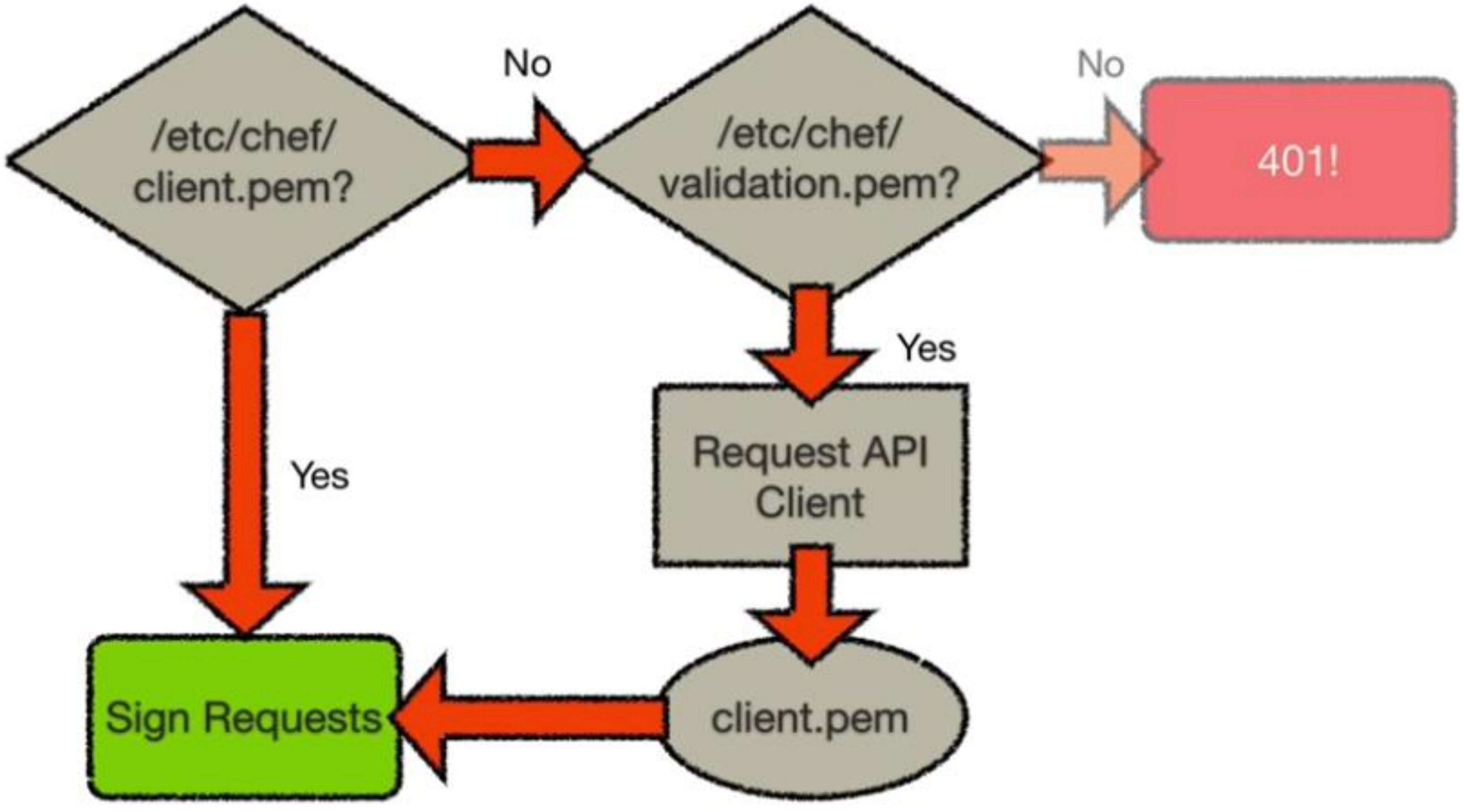
- The chef-client gets process configuration data from the client.rb file on the node, and then gets node configuration data from Ohai.
- One important piece of configuration data is the name of the node, which is found in the node_name attribute in the client.rb file or is provided by Ohai.
- If Ohai provides the name of a node, it is typically the FQDN for the node, which is always unique within an organization.

```
# cat /etc/chef/client.rb
log_location     STDOUT
chef_server_url  "https://chef-server.candl.com/organizations/candl"
validation_client_name "chef-validator"
node_name "node1"
trusted_certs_dir "/etc/chef/trusted_certs"
```

Authenticate to the Chef Server

- The chef-client authenticates to the Chef server using an RSA private key and the Chef server API.
- The name of the node is required as part of the authentication process to the Chef server.
- If this is the first chef-client run for a node, the chef-validator will be used to generate the RSA private key.

Authentication Process



Get, rebuild the node object

- The chef-client pulls down the node object from the Chef server.
- If this is the first chef-client run for the node, there will not be a node object to pull down from the Chef server.
- After the node object is pulled down from the Chef server, the chef-client rebuilds the node object.
- If this is the first chef-client run for the node, the rebuilt node object will contain only the default run-list.
- For any subsequent chef-client run, the rebuilt node object will also contain the run-list from the previous chef-client run.

Expand the run-list

- The chef-client expands the run-list from the rebuilt node object,
- Compiling a full and complete list of roles and recipes that will be applied to the node,
- Placing the roles and recipes in the same exact order they will be applied. (The run-list is stored in each node object's JSON file, grouped under run_list.)

Synchronize cookbooks

- The chef-client asks the Chef server for a list of all cookbook files (including recipes, templates, resources, providers, attributes, libraries, and definitions) that will be required to do every action identified in the run-list for the rebuilt node object.
- The Chef server provides to the chef-client a list of all of those files.
- The chef-client compares this list to the cookbook files cached on the node (from previous chef-client runs), and
- Then downloads a copy of every file that has changed since the previous chef-client run, along with any new files.

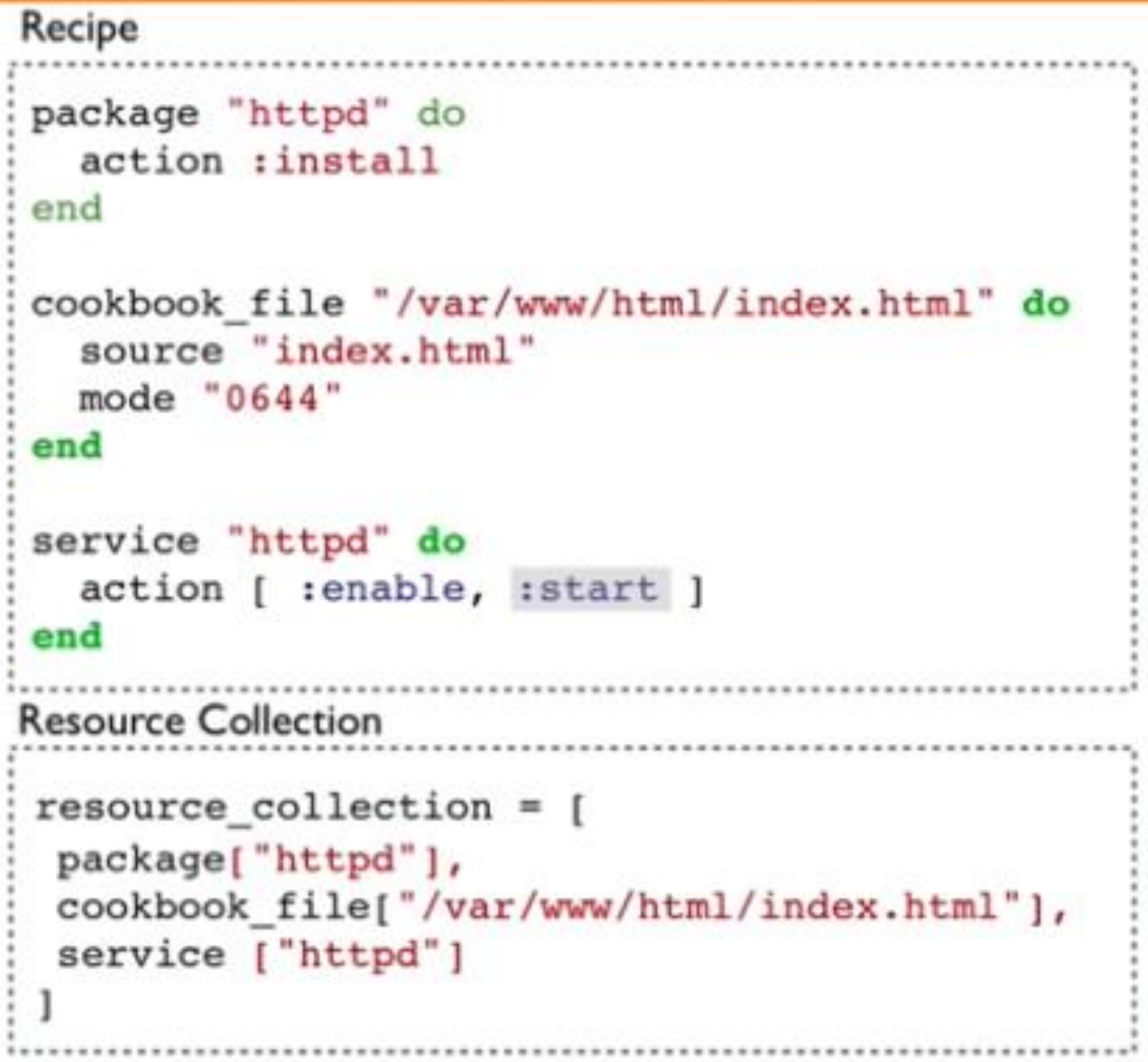
Reset node attributes

- All attributes in the rebuilt node object are reset.
- All attributes from attribute files, environments, roles, and Ohai are loaded.
- Attributes that are defined in attribute files are first loaded according to cookbook order.
- For each cookbook, attributes in the default.rbfile are loaded first, and then additional attribute files (if present) are loaded in lexical sort order. All attributes in the rebuilt node object are updated with the attribute data according to attribute precedence.
- When all of the attributes are updated, the rebuilt node object is complete.

Compile the resource collection

- The chef-client identifies each resource in the node object and builds the resource collection.
- Libraries are loaded first to ensure that all language extensions and Ruby classes are available to all resources.
- Next, attributes are loaded, followed by lightweight resources, and then all definitions (to ensure that any pseudo-resources within definitions are available).
- Finally, all recipes are loaded in the order specified by the expanded run-list. This is also referred to as the “compile phase”.

Compile the resource collection



Recipe

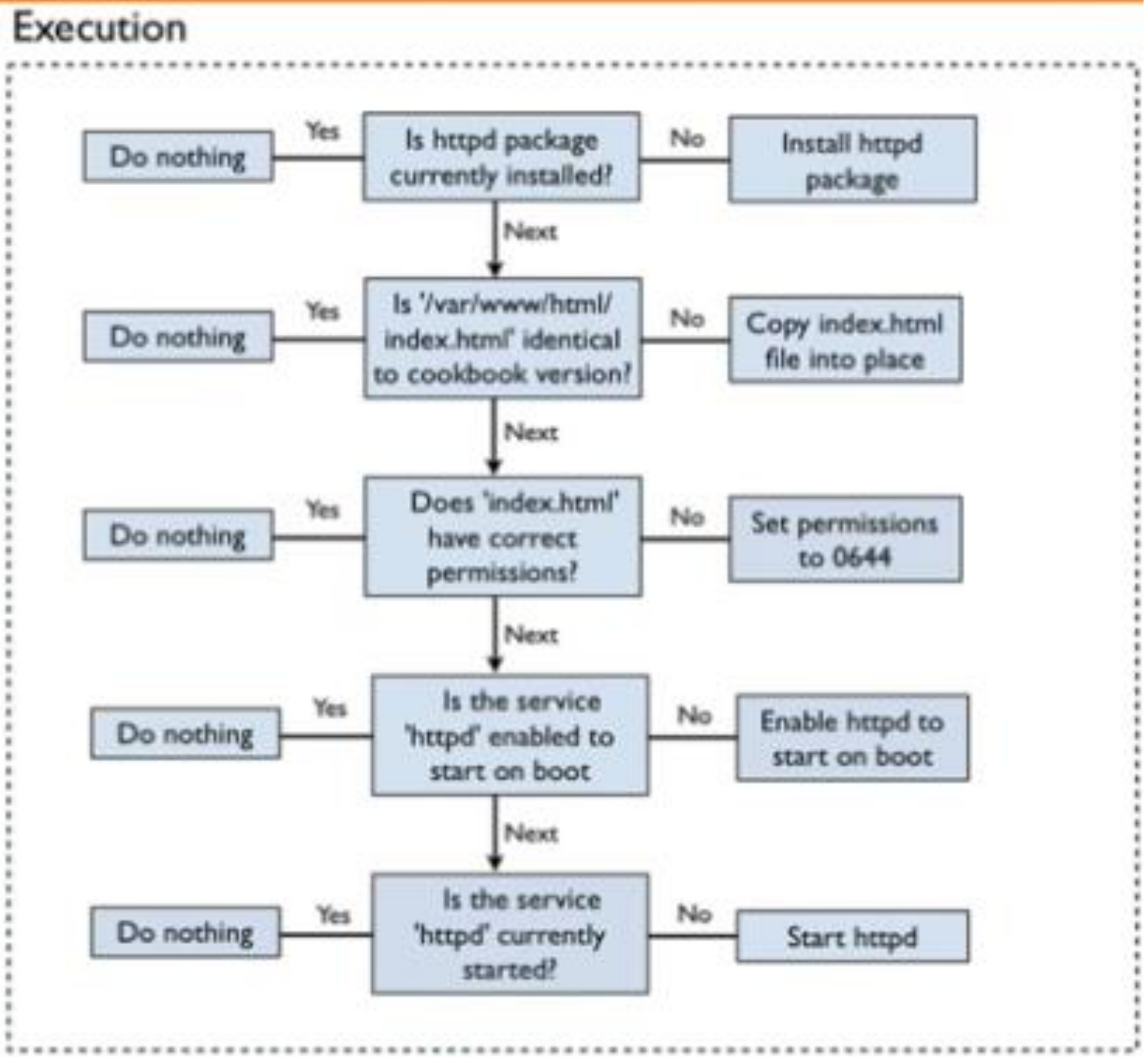
```
package "httpd" do
  action :install
end

cookbook_file "/var/www/html/index.html" do
  source "index.html"
  mode "0644"
end

service "httpd" do
  action [ :enable, :start ]
end
```

Resource Collection

```
resource_collection = [
 package["httpd"],
 cookbook_file["/var/www/html/index.html"],
 service ["httpd"]
]
```


Execution
Do nothing
Yes
Is httpd package currently installed?
No
Install httpd package
Next
Do nothing
Yes
Is '/var/www/html/index.html' identical to cookbook version?
No
Copy index.html file into place
Next
Do nothing
Yes
Does 'index.html' have correct permissions?
No
Set permissions to 0644
Next
Do nothing
Yes
Is the service 'httpd' enabled to start on boot
No
Enable httpd to start on boot
Next
Do nothing
Yes
Is the service 'httpd' currently started?
No
Start httpd

Converge the node

- The chef-client configures the system based on the information that has been collected.
- Each resource is executed in the order identified by the run-list, and then by the order in which each resource is listed in each recipe.
- Each resource in the resource collection is mapped to a provider.
- The provider examines the node, and then does the steps necessary to complete the action.
- And then the next resource is processed.
- Each action configures a specific part of the system.
- This process is also referred to as convergence. This is also referred to as the “execution phase”.

Update the node object, process exception and report handlers

- When all of the actions identified by resources in the resource collection have been done, and when the chef-client run finished successfully,
- the chef-client updates the node object on the Chef server with the node object that was built during this chef-client run. (This node object will be pulled down by the chef-client during the next chef-client run.)
- This makes the node object (and the data in the node object) available for search.
- The chef-client always checks the resource collection for the presence of exception and report handlers.
- If any are present, each one is processed appropriately.

Stop, wait for the next run

- When everything is configured and the chef-client run is complete,
- the chef-client stops and waits until the next time it is asked to run.