Chef Training - 2012

Presented by:

Jason Lancaster

Justin Witrick

This presentation was adapted from Opscode's training materials

Chef Training - 2012

What is Chef?

Chef is a configuration management tool.

- Declarative: What, not how
- Idempotent: Only take action if required
- Convergent: Takes care of itself

What is configuration management?

It is a process for establishing and maintaining consistency of a product's performance and functional and physical attributes with its requirements, design and operational information throughout its life

What Can it do?

This tool can be used to managing and maintaining servers in a repeatable way.

Chef Training - 2012

Where were we before Chef?

Rackspace Email – Used rpms called 'configure' that were built to have all configuration files populated correctly.

Why is that bad?

How do you keep all servers up to-date with the same version of the configuration file.

How can Chef help fix the issue of keeping all servers at the same point?

By having a centralized automated system that handles managing specific versions of packages and configuration values, it allows all the servers to stay at the same point.

Chef in depth

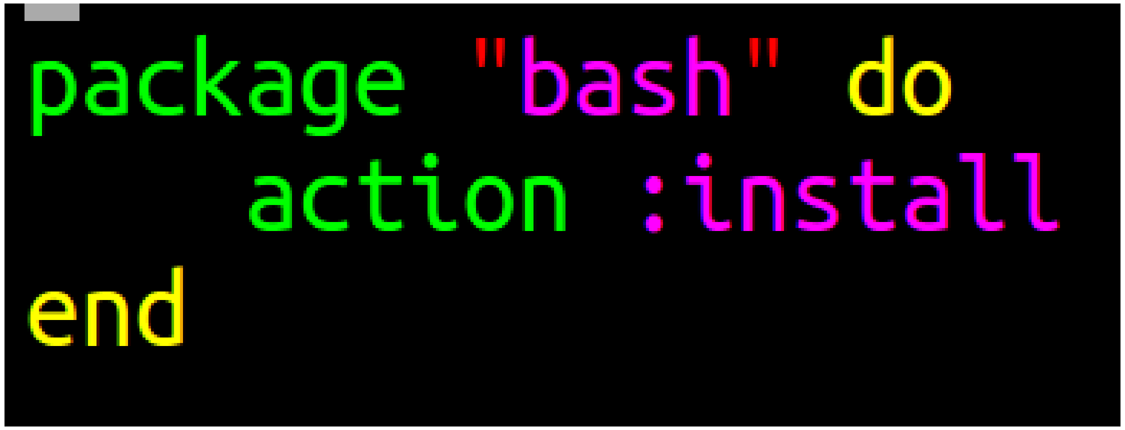
Topics:

- Chef Resources
- Idempotent Actions
- Convergent Nodes
- Recipe Helpers
- Configuration Policy
- Applying the Policy
- Anatomy of chef run
- Chef server components

Chef Resources

You configure systems with Chef by writing self-documenting code. This code is lists of *Resources* that configure the system to do its job.

Chef manages system resources with a declarative interface that abstracts the details.



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

Chef Resource cont.

Defined resources:

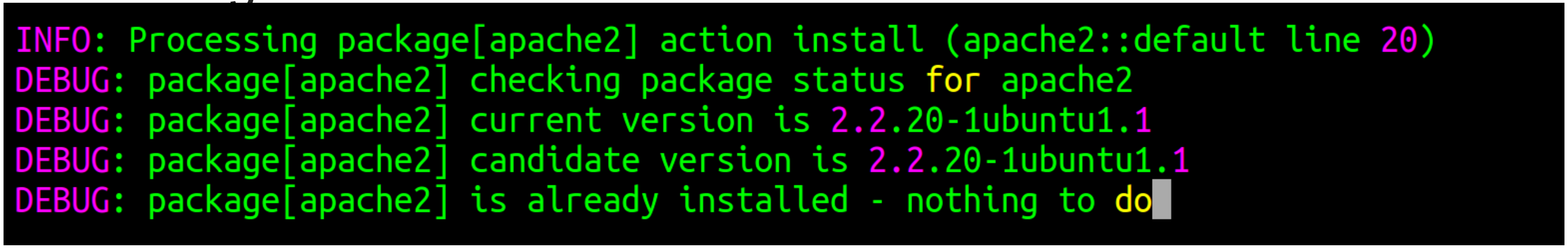
- Cookbook File
- Cron
- Directory
- Execute
- File
- HTTP Request
- Ifconfig
- Log
- Mount
- Package
- PowerShell script
- Ruby block
- Template

A Complete list can be found at:

http://wiki.opscode.com/display/chef/Resources

Idempotent Actions

Chef Resources have *Providers* that take idempotent action to configure the resource, but only if it needs to change.



```
INFO: Processing package[apache2] action install (apache2::default line 20)
DEBUG: package[apache2] checking package status for apache2
DEBUG: package[apache2] current version is 2.2.20-1ubuntu1.1
DEBUG: package[apache2] candidate version is 2.2.20-1ubuntu1.1
DEBUG: package[apache2] is already installed - nothing to do
```

Chef providers handle the details of checking the current state of the resource. Different platforms may have different providers for managing the same type of resource, yum vs apt, init vs upstart.

Convergent Nodes

Chef runs on the system, configuring the Node. The node is the initial unit of authority about itself.

The node is responsible only for itself.

In Chef, a single run should completely configure the system. If it does not, it is a bug (in your code, on the system, or in Chef itself).

When Chef runs, it saves the node to the Chef Server, making that information available through a network-accessible API.

Chef: The Framework

Chef provides a framework for system integration.

- Resources are written in Chef Recipes, a Ruby domain-specific language (DSL).
- Recipe helpers such as search allow dynamic data usage.
- Chef provides a library of primitives that can be used for other purposes.

Recipe Ruby DSL

Ruby is a 3rd generation interpreted programming language. Ruby has features that make it easy to create domain specific languages. This lends itself quite nicely to configuration management.

In Chef, Ruby gets out of the way, but it is still there when you need it.

Chef Recipes are a pure Ruby domain specific language. They are collected in Cookbooks along with associated components like config files or libraries.

Recipe Helpers

Chef provides a number of recipe helpers to obtain and manipulate data to use in Resources.

Search is used to discover information like IP addresses about other systems.

Arbitrary data about the infrastructure can be stored in Data Bags and accessed in recipes.

Because Chef uses Ruby, you can create your own helpers, too.

Configuration Policy

Policy about the nodes is written in recipes, which are stored in Cookbooks.

Cookbooks are uploaded to the Chef Server and distributed to the nodes that should be configured.

Cookbooks have versions and dependencies, both of which affect what code gets executed on particular nodes.

Apply the policy

Tying it all together are roles which are descriptions of the nodes.

A webserver role contains the list of cookbooks and node-specific information required to fulfill serving HTTP traffic.

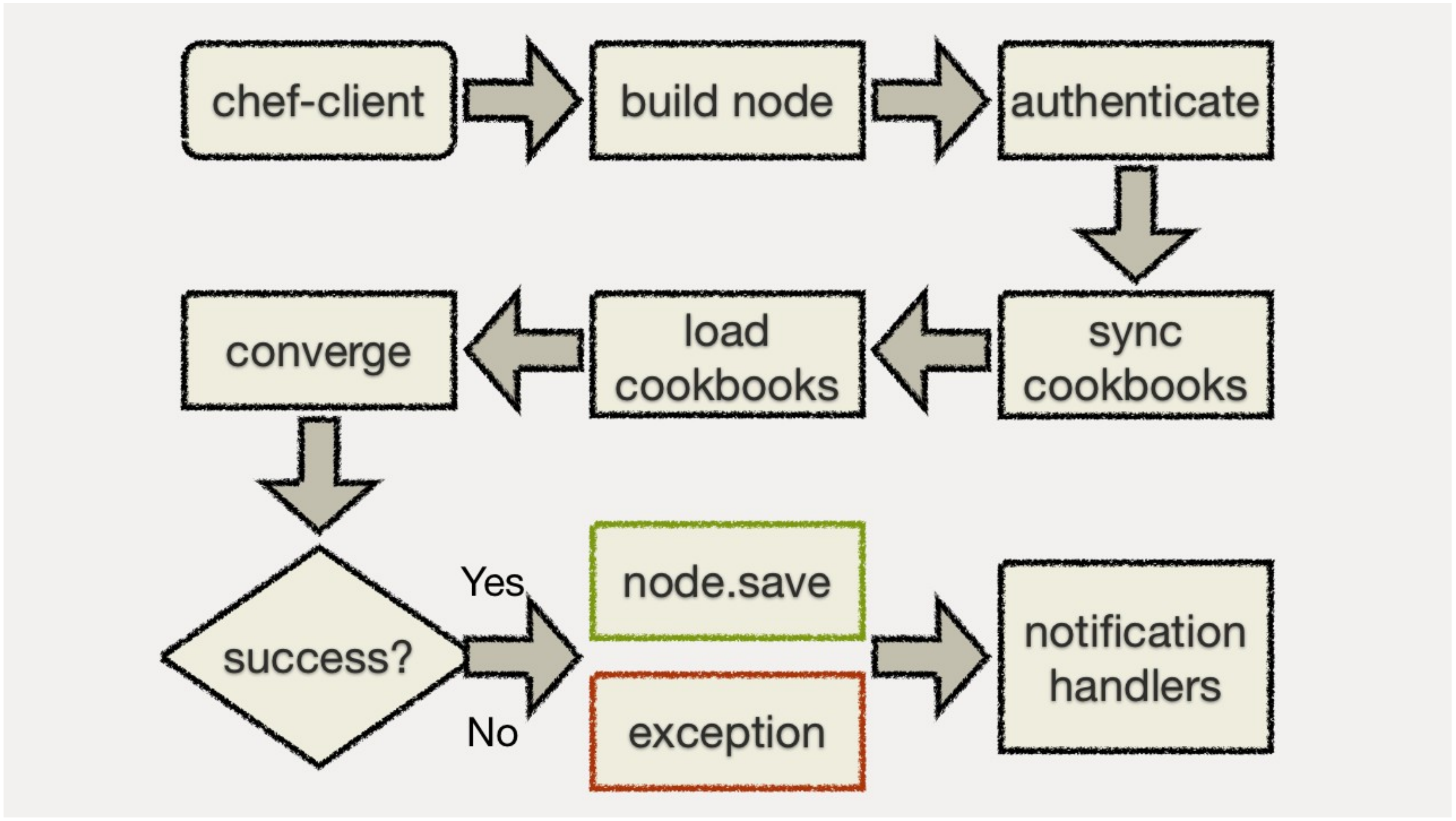
The node has a list of roles and/or recipes that Chef should run to configure the node to do its job. Chef downloads only what it needs from the Chef Server.

Anatomy of Chef Run

Anatomy

- Build the node
- Synchronize cookbooks
- Compile resource collection
- Configure the node
- Run notification handlers

Anatomy of chef run



Build Node

- After the client has authenticated with the Server, Chef retrieves the node object from the server.
- Node objects represent a set of data called attributes and a list of configuration to apply called a run list.

Synchronize Cookbooks

- Chef downloads from the Chef Server all the cookbooks that appear as recipes in the node's expanded run list.
- Chef also downloads all cookbooks that are listed as dependencies which might not appear in the run list.
- If the node's chef_environment specifies cookbook versions, the Chef downloads the version specified. Otherwise the latest available version is downloaded.

Node Convergence

Convergence is when the configuration management system brings the node into compliance with policy.

In other words, the node is configured based on the roles and recipes in its run list.

Convergence in Chef happens in two phases.

- Compile
- Execute

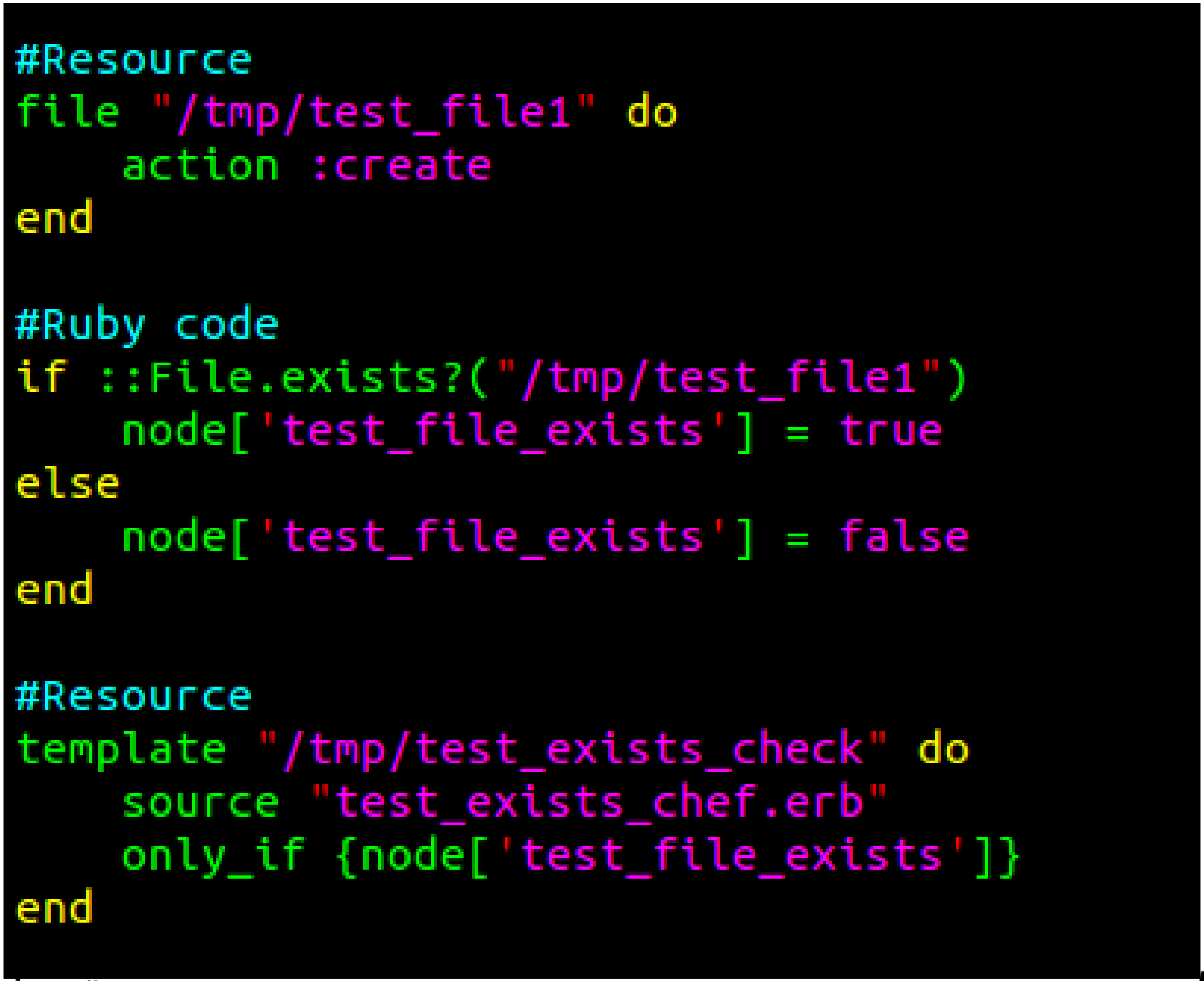
Convergence: Compile

Chef recipes are written in Ruby. During the compile phase, the Chef Recipe DSL is processed for Chef Resources to be configured.

During the processing of recipes:

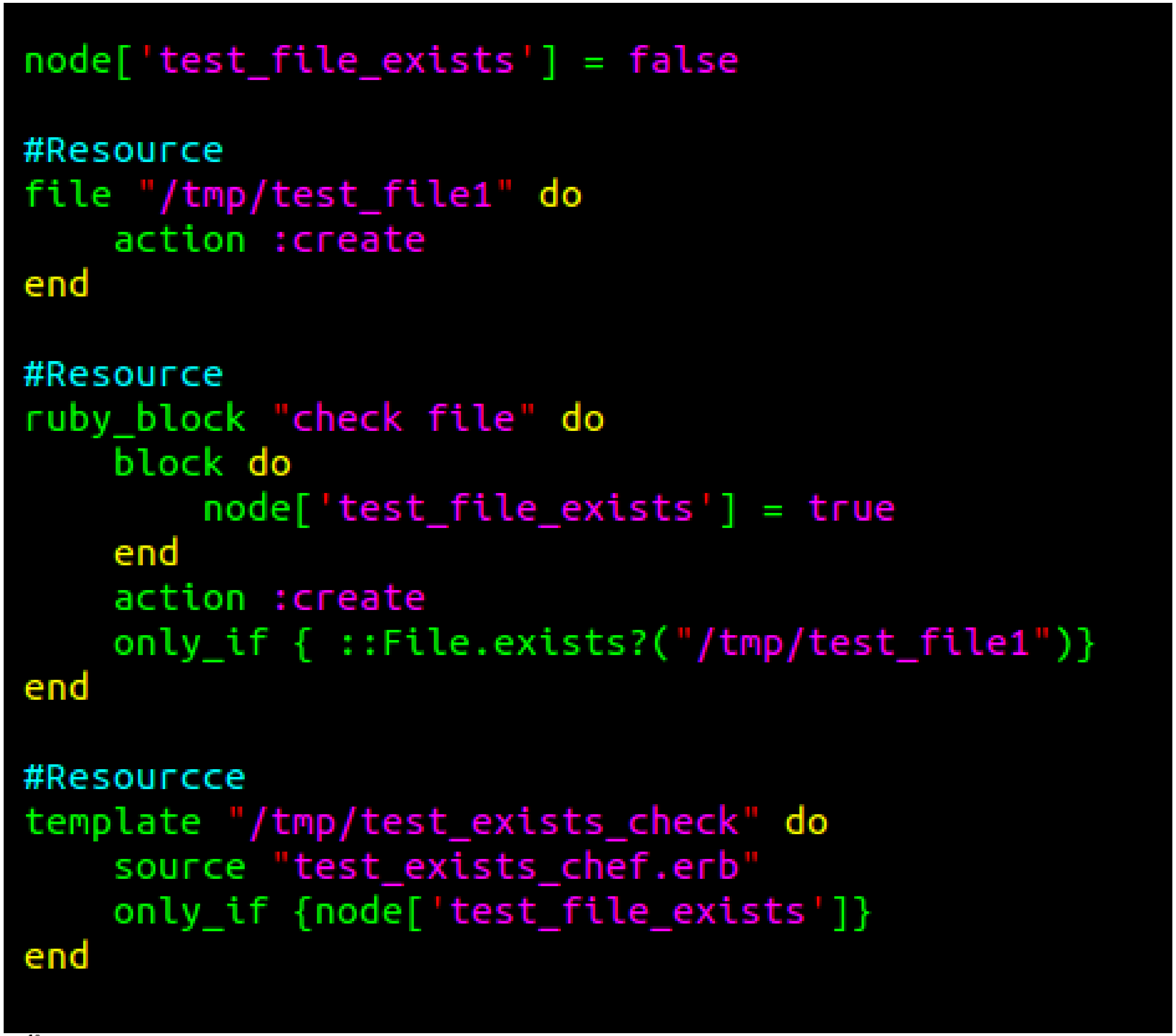
- Ruby code is executed <u>directly</u>
- Recognized resources are added to the Resource Collection

Compile time



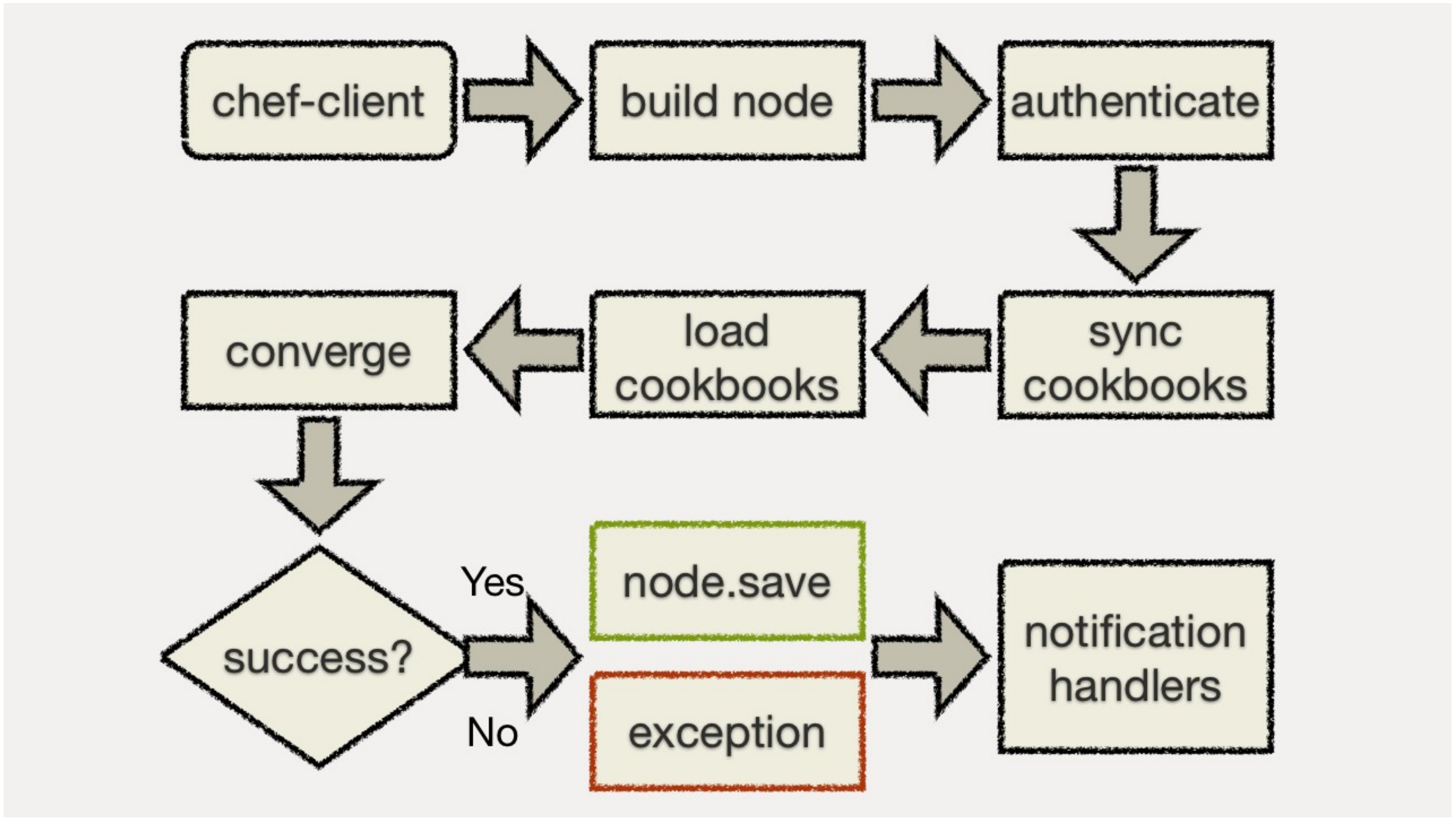
```
#Resource
file "/tmp/test_file1" do
    action :create
end

#Ruby code
if ::File.exists?("/tmp/test_file1")
    node['test_file_exists'] = true
else
    node['test_file_exists'] = false
end

#Resource
template "/tmp/test_exists_check" do
    source "test_exists_chef.erb"
    only_if {node['test_file_exists']}
end
```

Resource execution



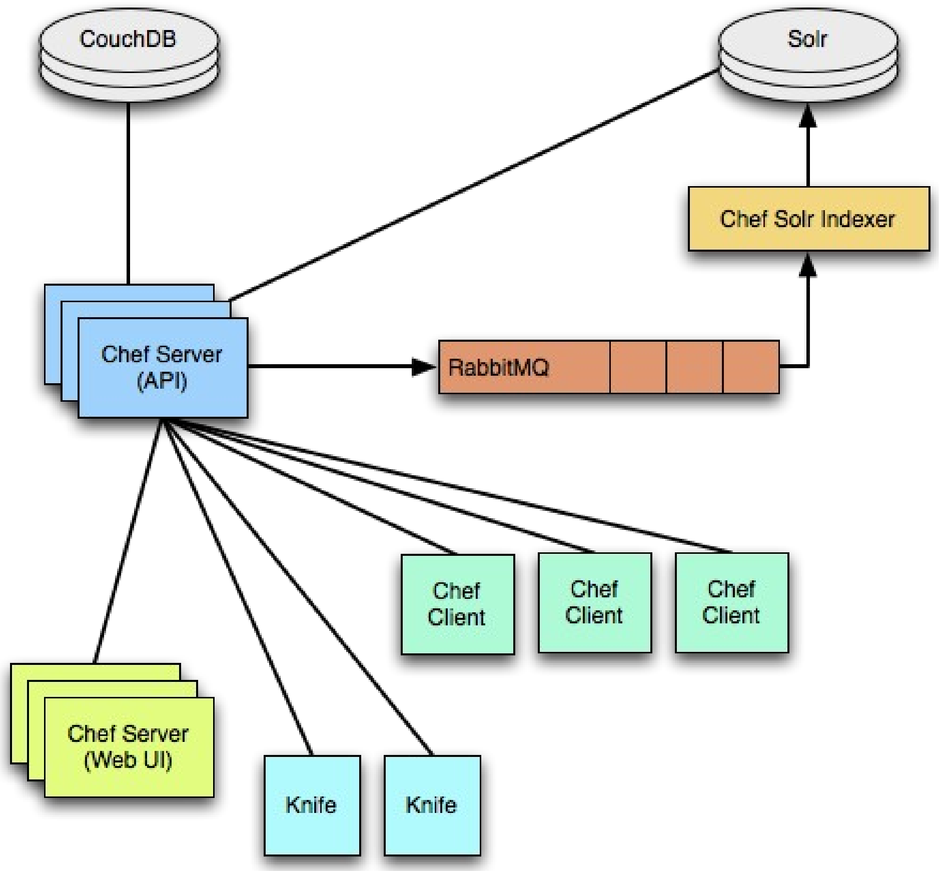
```
node['test_file_exists'] = false

#Resource
file "/tmp/test_file1" do
    action :create
end

#Resource
ruby_block "check file" do
    block do
        node['test_file_exists'] = true
    end
    action :create
    only_if { ::File.exists?("/tmp/test_file1")}
end

#Resourcce
template "/tmp/test_exists_check" do
    source "test_exists_chef.erb"
    only_if {node['test_file_exists']}
end
```

Anatomy of chef run



Chef Server

Chef Server


CouchDB
Solr
Chef Solr Indexer
Chef Server (API)
RabbitMQ
Chef Client
Chef Client
Chef Client
Chef Server (Web UI)
Knife
Knife

Rabbitmq

The Chef Server runs RabbitMQ as an AMQP (Advanced Message Queuing Protocol) server. Whenever data is stored in CouchDB that needs to be indexed by SOLR, the server sends a message and the data payload to the queue, and the indexer picks it up.

CouchDB

Chef Server utilizes CouchDB for storing JSON data about Nodes, Roles, and Data Bags

Tools for Interactions with Chef

Topics:

- Knife
- chef-client
- chef-solo

Knife Utility

Knife is the "swiss army knife" of infrastructure management tools.

- manage the local Chef repository
- interact with the Chef Server API
- interact with cloud computing providers' APIs
- extend with custom plugins/libraries

Knife Commands

knife node show NODENAME

Displays information about a particular node

knife cookbook upload fail2ban

Uploads cookbook 'fail2ban' to chef-server

knife role edit webserver

Opens a role for modification

chef-client & chef-solo

The programs chef-client and chef-solo load the Chef library and make it available to apply configuration management with Chef.

Both programs know how to configure the system given the appropriate recipes found in cookbooks.

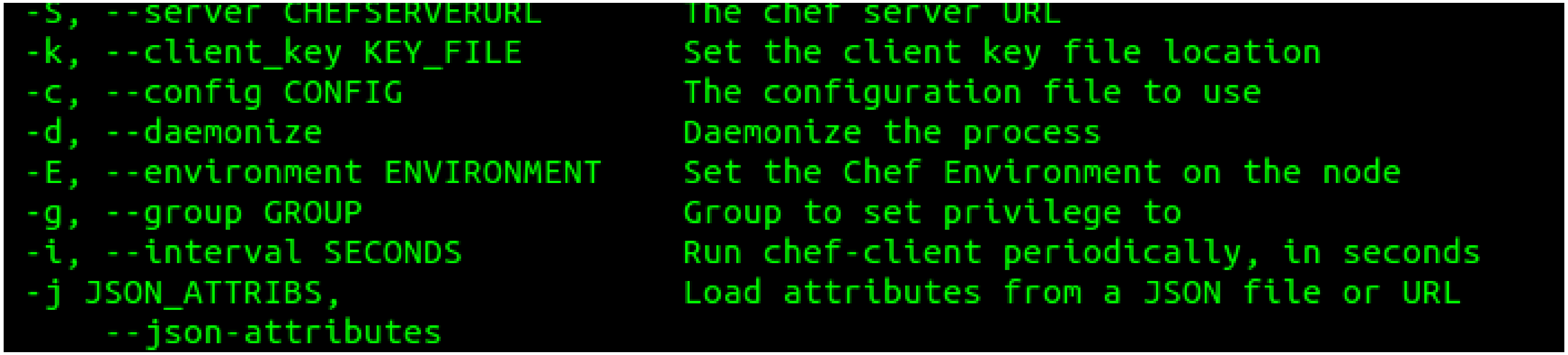
chef-client

chef-client talks to a Chef Server API endpoint, authenticating with an RSA key pair. It retrieves data and code from the server to configure the node per the defined policy.

List of recipes can be predefined, assigned to a node on the Chef Server, and retrieved when chef-client runs.

The default configuration file is /etc/chef/client.rb.

Chef-client commands



```
-S, --server CHEFSERVERURL        The chef server URL
-k, --client_key KEY_FILE         Set the client key file location
-c, --config CONFIG               The configuration file to use
-d, --daemonize                   Daemonize the process
-E, --environment ENVIRONMENT     Set the Chef Environment on the node
-g, --group GROUP                 Group to set privilege to
-i, --interval SECONDS            Run chef-client periodically, in seconds
-j JSON_ATTRIBS,                  Load attributes from a JSON file or URL
    --json-attributes
```

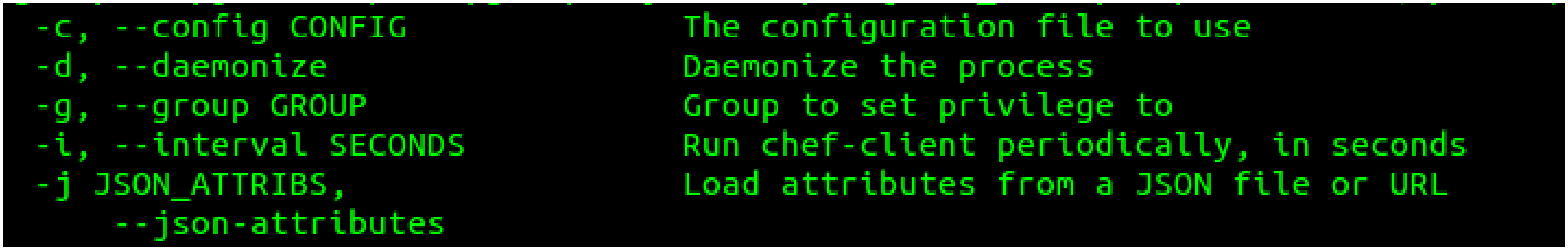
chef-solo

chef-solo operates without a Chef Server. It requires that all the recipes it needs are available, and that it be told what to run on the node.

A JSON file is passed to chef-solo to give it these instructions in a run_list for the node.

The default configuration file is /etc/chef/solo.rb.

Chef-solo commands



```
  -c, --config CONFIG                The configuration file to use
  -d, --daemonize                    Daemonize the process
  -g, --group GROUP                  Group to set privilege to
  -i, --interval SECONDS             Run chef-client periodically, in seconds
  -j JSON_ATTRIBS,                   Load attributes from a JSON file or URL
      --json-attributes
```

Chef Components

- Nodes
- Cookbooks
- Recipe
- Role
- Environments
- Data bags

Nodes

A node is a server that runs your software.

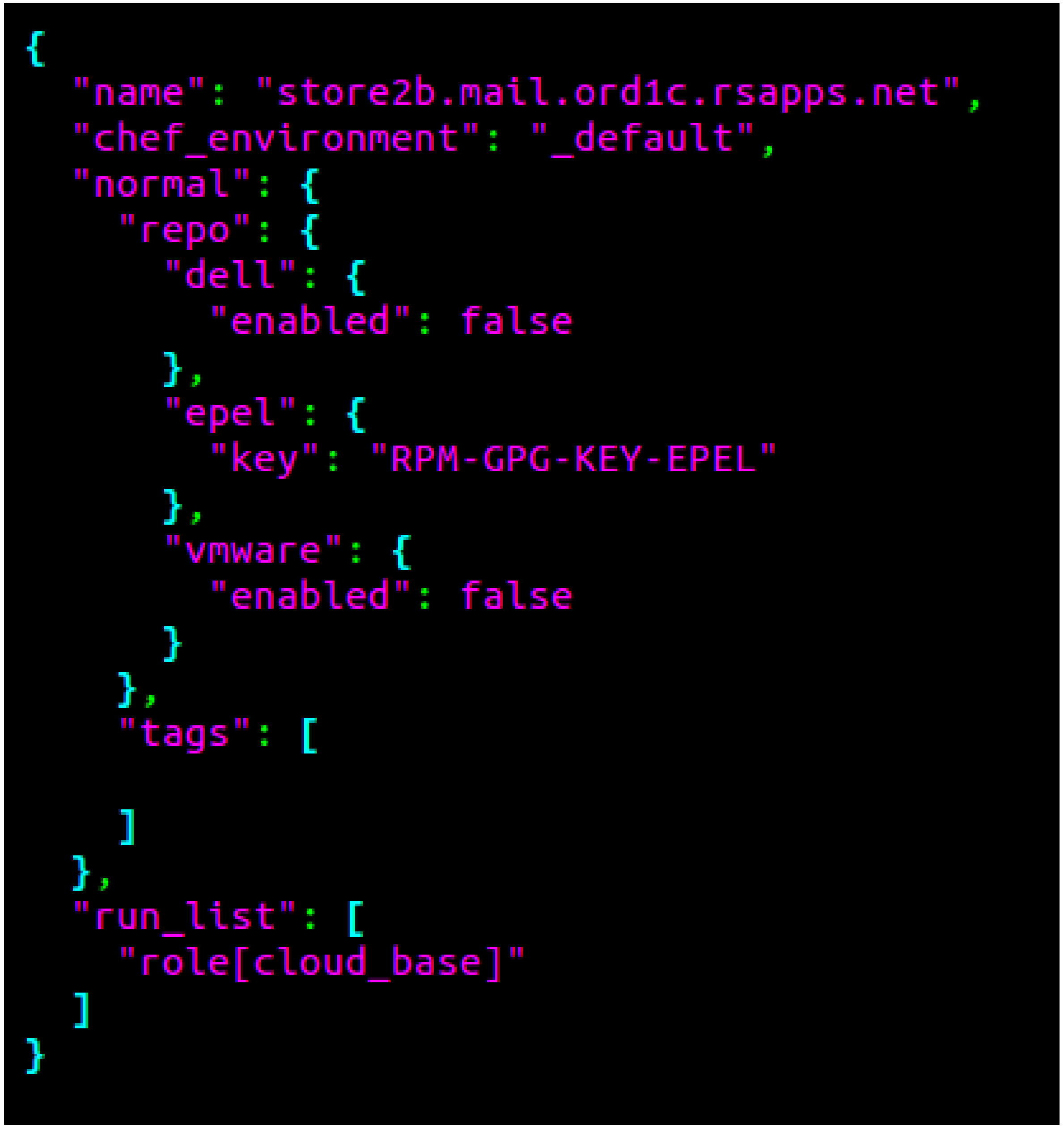
Chef::Node is the node object. It looks and almost behaves like a hash, except when it doesn't.

Nodes have attributes at varying priority levels (automatic, default, normal, override).

Nodes have a run list.

Nodes have an environment.

Nodes example



```
{
  "name": "store2b.mail.ord1c.rsapps.net",
  "chef_environment": "_default",
  "normal": {
    "repo": {
      "dell": {
        "enabled": false
      },
      "epel": {
        "key": "RPM-GPG-KEY-EPEL"
      },
      "vmware": {
        "enabled": false
      }
    },
    "tags": [

    ]
  },
  "run_list": [
    "role[cloud_base]"
  ]
}
```

Node Run lists

The run list can contain recipes and roles. Roles can contain recipes and also other roles.

Chef expands the node's run list down to the recipes. The roles and recipes get set to node attributes.

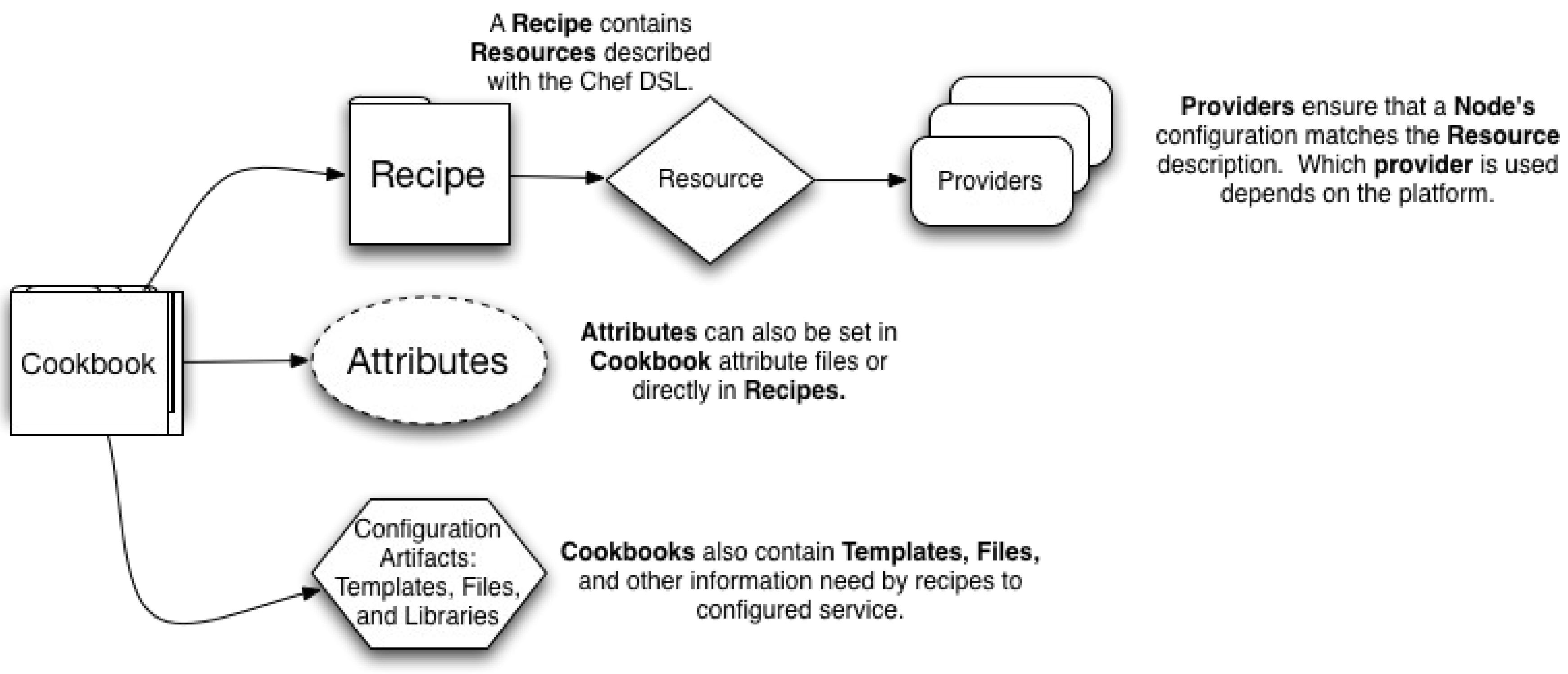
Cookbooks

Cookbooks are the fundamental units of file distribution in Chef. They are "packages" for Chef recipes and other helper components.

They are designed to be sharable packages for managing infrastructure as code. Cookbooks can be shared within an organization, or with the Chef Community.

Nodes managed by Chef download cookbooks from the Chef Server to apply their configuration.

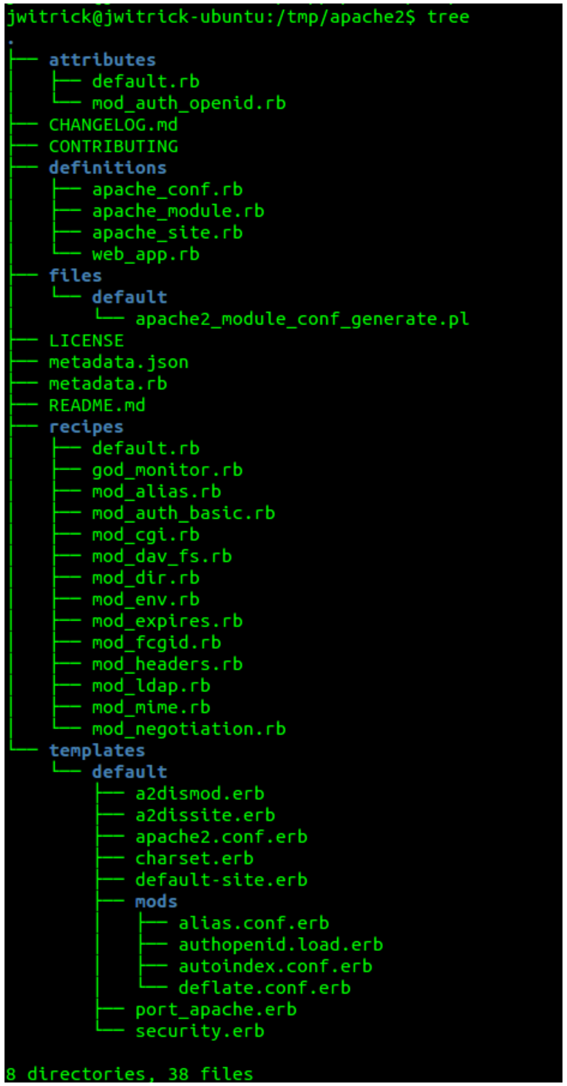
Cookbook cont.



A **Recipe** contains **Resources** described with the Chef DSL.
Recipe
Resource
Providers
Providers ensure that a **Node's** configuration matches the **Resource** description. Which **provider** is used depends on the platform.
Cookbook
Attributes
Attributes can also be set in **Cookbook** attribute files or directly in **Recipes.**
Configuration Artifacts: Templates, Files, and Libraries
Cookbooks also contain **Templates, Files,** and other information need by recipes to configured service.

Cookbooks cont.

Below is an example structure of a cookbook:



```
jwitrick@jwitrick-ubuntu:/tmp/apache2$ tree
.
├── attributes
│   ├── default.rb
│   └── mod_auth_openid.rb
├── CHANGELOG.md
├── CONTRIBUTING
├── definitions
│   ├── apache_conf.rb
│   ├── apache_module.rb
│   ├── apache_site.rb
│   └── web_app.rb
├── files
│   └── default
│       └── apache2_module_conf_generate.pl
├── LICENSE
├── metadata.json
├── metadata.rb
├── README.md
├── recipes
│   ├── default.rb
│   ├── god_monitor.rb
│   ├── mod_alias.rb
│   ├── mod_auth_basic.rb
│   ├── mod_cgi.rb
│   ├── mod_dav_fs.rb
│   ├── mod_dir.rb
│   ├── mod_env.rb
│   ├── mod_expires.rb
│   ├── mod_fcgid.rb
│   ├── mod_headers.rb
│   ├── mod_ldap.rb
│   ├── mod_mime.rb
│   └── mod_negotiation.rb
└── templates
    └── default
        ├── a2dismod.erb
        ├── a2dissite.erb
        ├── apache2.conf.erb
        ├── charset.erb
        ├── default-site.erb
        ├── mods
        │   ├── alias.conf.erb
        │   ├── authopenid.load.erb
        │   ├── autoindex.conf.erb
        │   └── deflate.conf.erb
        ├── port_apache.erb
        └── security.erb

8 directories, 38 files
```

Notice:

- How there are multiple recipes under recipes.
- There are multiple attribute files.

Cookbook cont.

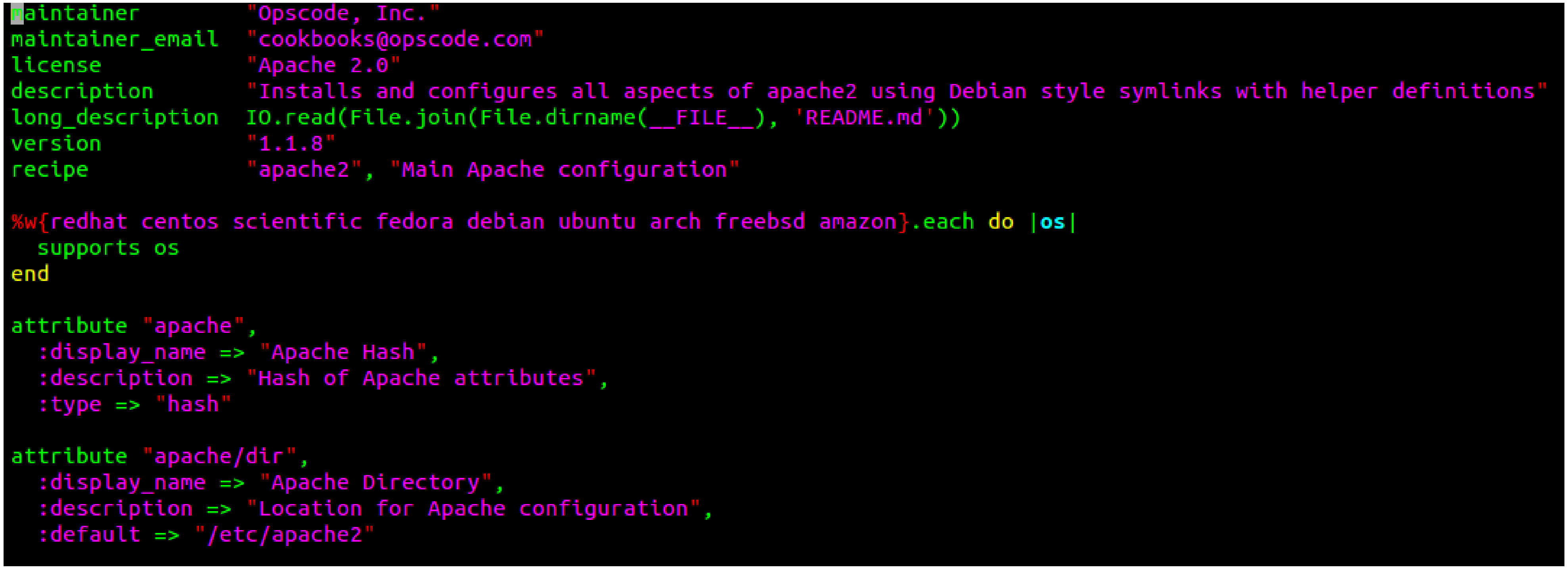
The most commonly used cookbook components are:

- recipes
- metadata
- assets (files and templates)

Cookbook cont. Metadata

Metadata serves two purposes.

– Documentation
– Dependency management



```
maintainer        "Opscode, Inc."
maintainer_email  "cookbooks@opscode.com"
license           "Apache 2.0"
description       "Installs and configures all aspects of apache2 using Debian style symlinks with helper definitions"
long_description  IO.read(File.join(File.dirname(__FILE__), 'README.md'))
version           "1.1.8"
recipe            "apache2", "Main Apache configuration"

%w{redhat centos scientific fedora debian ubuntu arch freebsd amazon}.each do |os|
  supports os
end

attribute "apache",
  :display_name => "Apache Hash",
  :description => "Hash of Apache attributes",
  :type => "hash"

attribute "apache/dir",
  :display_name => "Apache Directory",
  :description => "Location for Apache configuration",
  :default => "/etc/apache2"
```

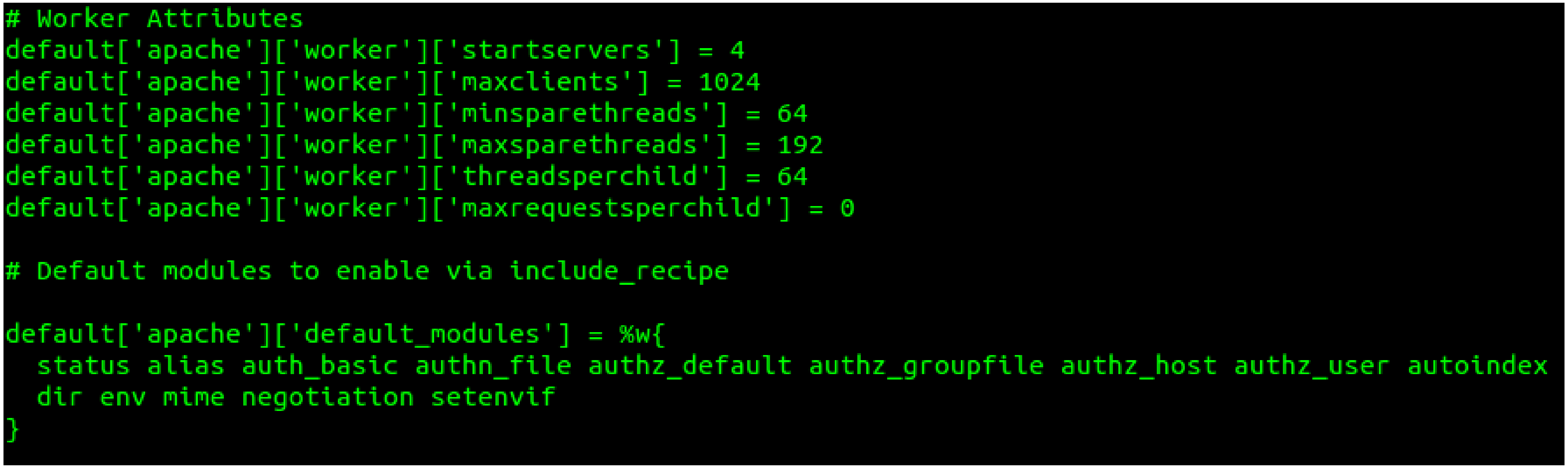
Attributes

An attribute is a way to store data either in cookbook, role, environment or on the node.

There are four levels:

- **Automatic**
- **Override**
- **Normal**
- **Default**

Attributes



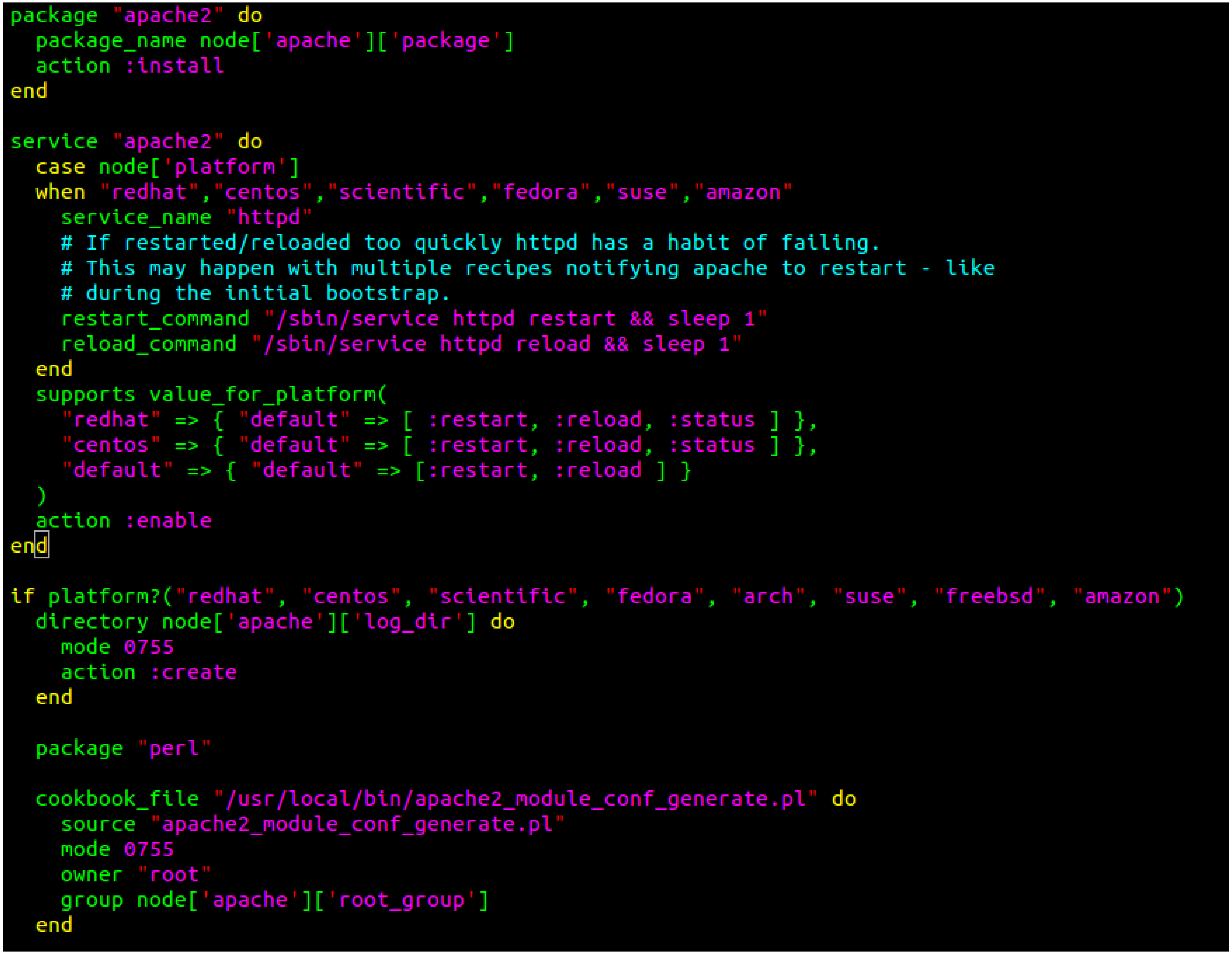
```
# Worker Attributes
default['apache']['worker']['startservers'] = 4
default['apache']['worker']['maxclients'] = 1024
default['apache']['worker']['minsparethreads'] = 64
default['apache']['worker']['maxsparethreads'] = 192
default['apache']['worker']['threadsperchild'] = 64
default['apache']['worker']['maxrequestsperchild'] = 0

# Default modules to enable via include_recipe

default['apache']['default_modules'] = %w{
  status alias auth_basic authn_file authz_default authz_groupfile authz_host authz_user autoindex
  dir env mime negotiation setenvif
}
```

Recipes

Recipes are the work unit in Chef. They contain lists of resources that should be configured on the node to put it in the desired state to fulfill its job.

Nodes have a run list, which is simply a list of the recipes that should be applied when Chef runs.

The node will download all the cookbooks that appear in its run list.

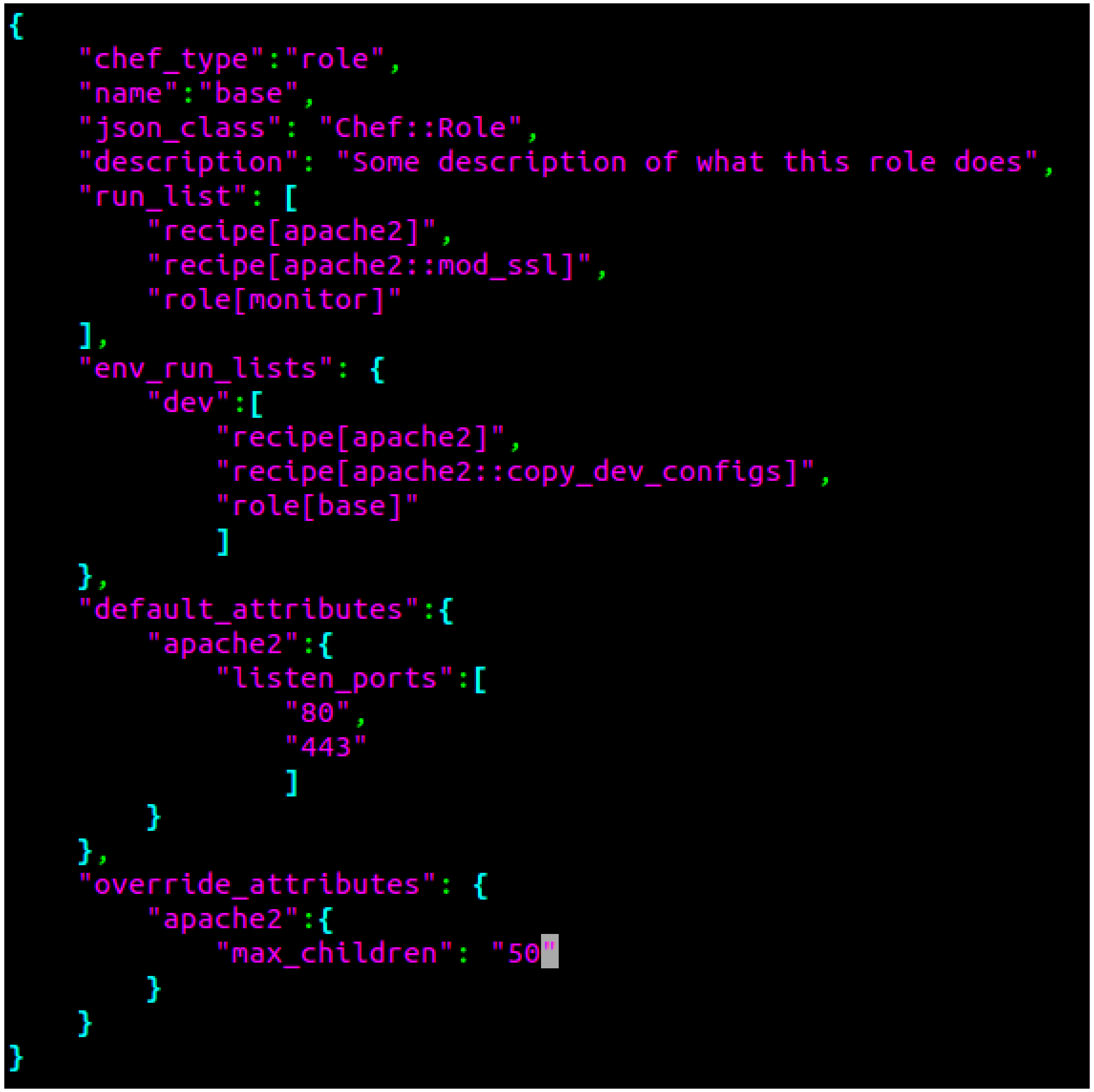


```ruby
package "apache2" do
  package_name node['apache']['package']
  action :install
end

service "apache2" do
  case node['platform']
  when "redhat","centos","scientific","fedora","suse","amazon"
    service_name "httpd"
    # If restarted/reloaded too quickly httpd has a habit of failing.
    # This may happen with multiple recipes notifying apache to restart - like
    # during the initial bootstrap.
    restart_command "/sbin/service httpd restart && sleep 1"
    reload_command "/sbin/service httpd reload && sleep 1"
  end
  supports value_for_platform(
    "redhat" => { "default" => [ :restart, :reload, :status ] },
    "centos" => { "default" => [ :restart, :reload, :status ] },
    "default" => { "default" => [:restart, :reload ] }
  )
  action :enable
end

if platform?("redhat", "centos", "scientific", "fedora", "arch", "suse", "freebsd", "amazon")
  directory node['apache']['log_dir'] do
    mode 0755
    action :create
  end

  package "perl"

  cookbook_file "/usr/local/bin/apache2_module_conf_generate.pl" do
    source "apache2_module_conf_generate.pl"
    mode 0755
    owner "root"
    group node['apache']['root_group']
  end
```

Roles

A role can be used to group a set of cookbooks together for a specific purpose in a repeatable way.

Roles. Cont.



```
{
    "chef_type":"role",
    "name":"base",
    "json_class": "Chef::Role",
    "description": "Some description of what this role does",
    "run_list": [
        "recipe[apache2]",
        "recipe[apache2::mod_ssl]",
        "role[monitor]"
    ],
    "env_run_lists": {
        "dev":[
            "recipe[apache2]",
            "recipe[apache2::copy_dev_configs]",
            "role[base]"
            ]
    },
    "default_attributes":{
        "apache2":{
            "listen_ports":[
                "80",
                "443"
                ]
        }
    },
    "override_attributes": {
        "apache2":{
            "max_children": "50"
        }
    }
}
```

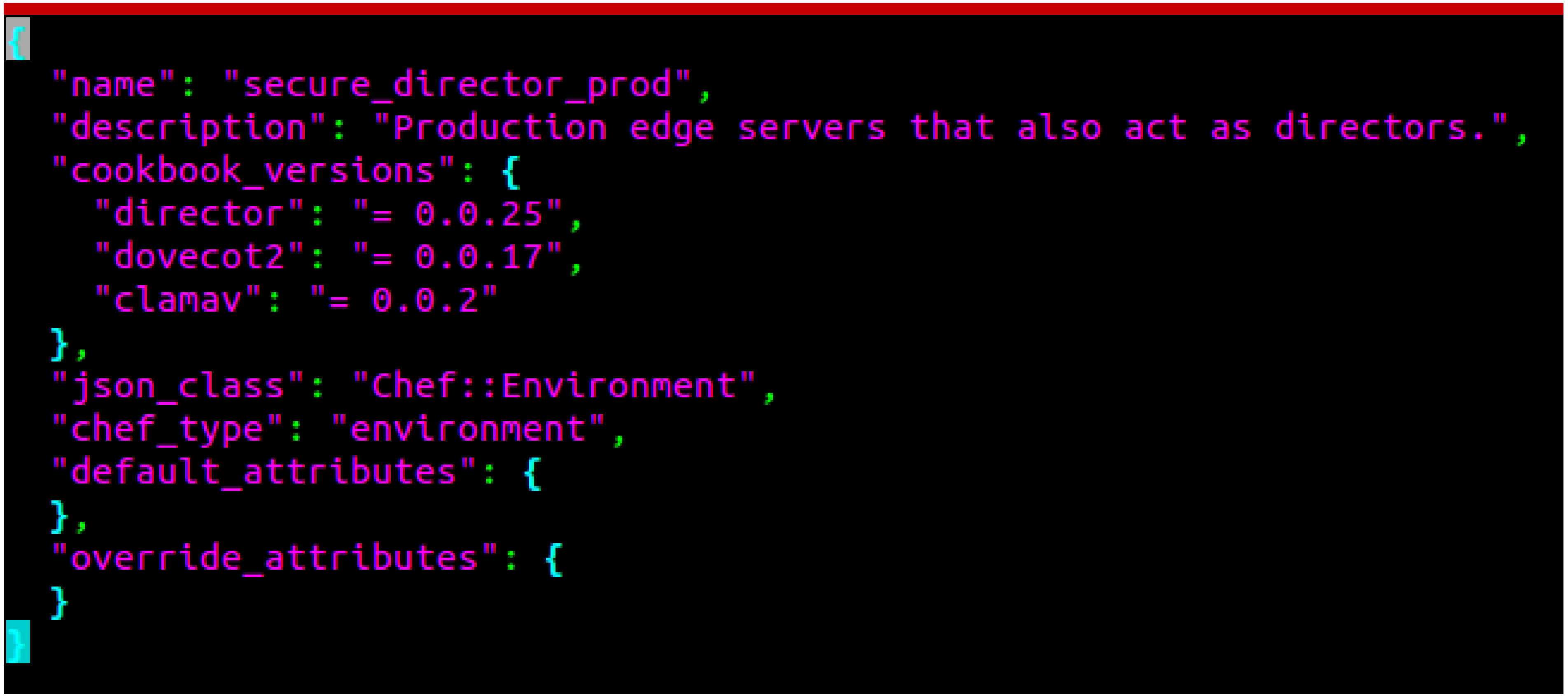
Environments

Every node has an environment. If you do not explicitly set one, _default is used.

Chef environments allow you to set cookbook version constraints to nodes in a particular environment, e.g. "production".

Retrieve the value of the node's environment with the Chef::Node#chef_environment method.

Environments. Cont.



```
{
  "name": "secure_director_prod",
  "description": "Production edge servers that also act as directors.",
  "cookbook_versions": {
    "director": "= 0.0.25",
    "dovecot2": "= 0.0.17",
    "clamav": "= 0.0.2"
  },
  "json_class": "Chef::Environment",
  "chef_type": "environment",
  "default_attributes": {
  },
  "override_attributes": {
  }
}
```

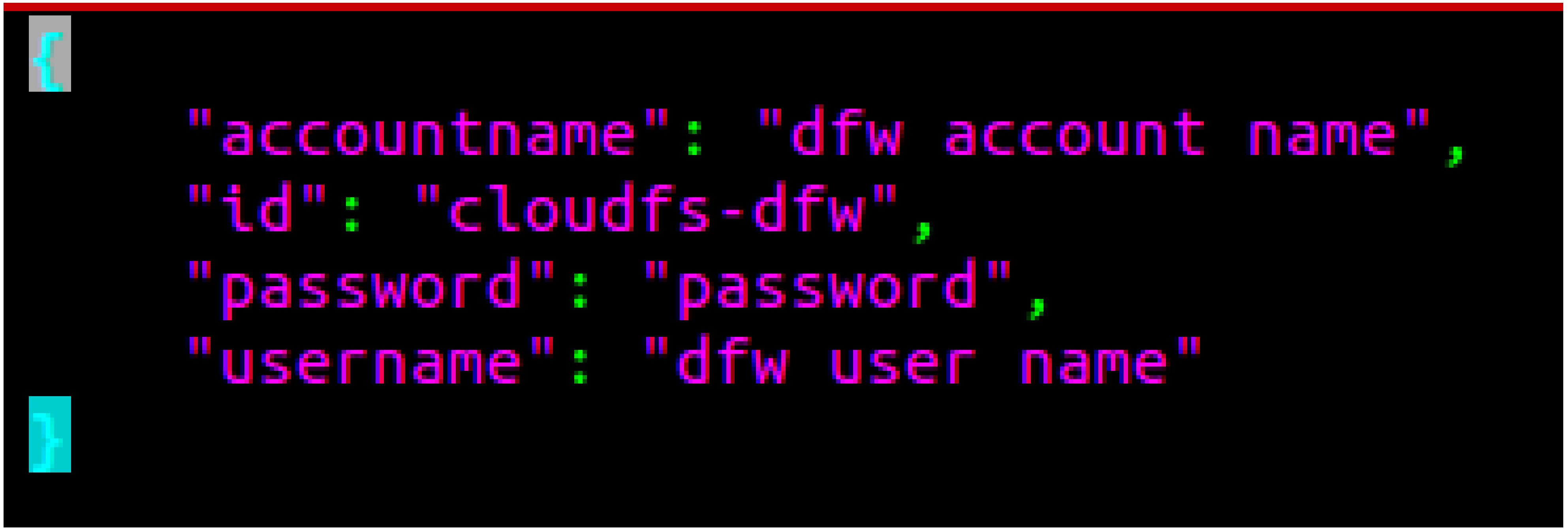
Data Bags

Data bags are containers of "items" that are plain JSON data.

Items can contain any arbitrary key/value pairs, such as user information, application setup parameters or DNS entries.

They are centrally available to recipes for processing.

Data Bags – example file



```
{
        "accountname": "dfw account name",
        "id": "cloudfs-dfw",
        "password": "password",
        "username": "dfw user name"
}
```

Data Bags - searching

There are 2 ways to access a data bag:

1. Using chef built in search:

   search(:<data_bag_name>, “id:<data_bag_item_id>”)

2. Using chef data_bag function:

   data_bag(“<data_bag_name>”) and
   data_bag_item(“<data_bag_name>”,
   “<data_bag_item_id>”)

QUESTIONS