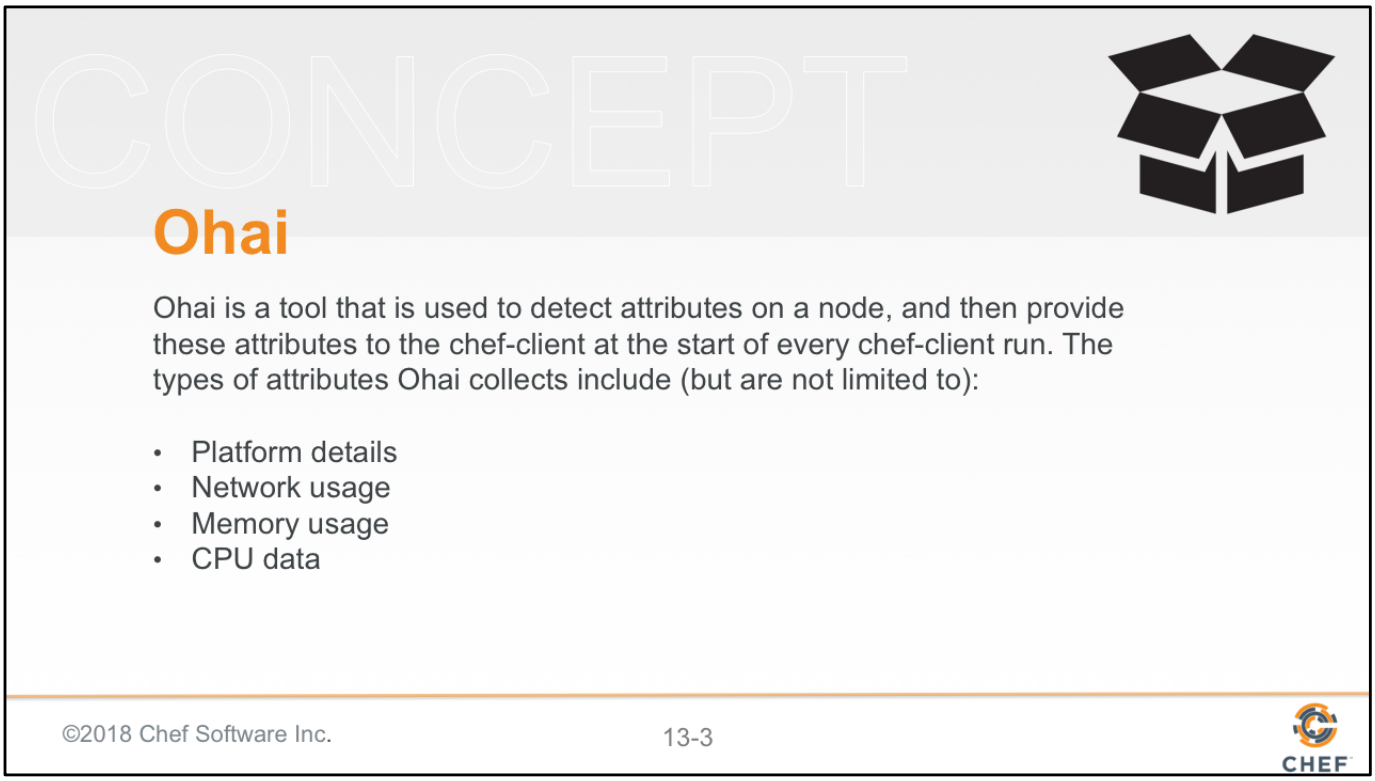
Before you set out to start managing your nodes it is important to understand the current state of your nodes. As Chef, a platform agnostic tool, is written in Ruby, a platform agnostic language, it is useful to understand what is or is not installed on the system. This information is helpful in helping a resource select the correct provider or for that provider to determine which version of the tool or language is at it disposal.

Objectives

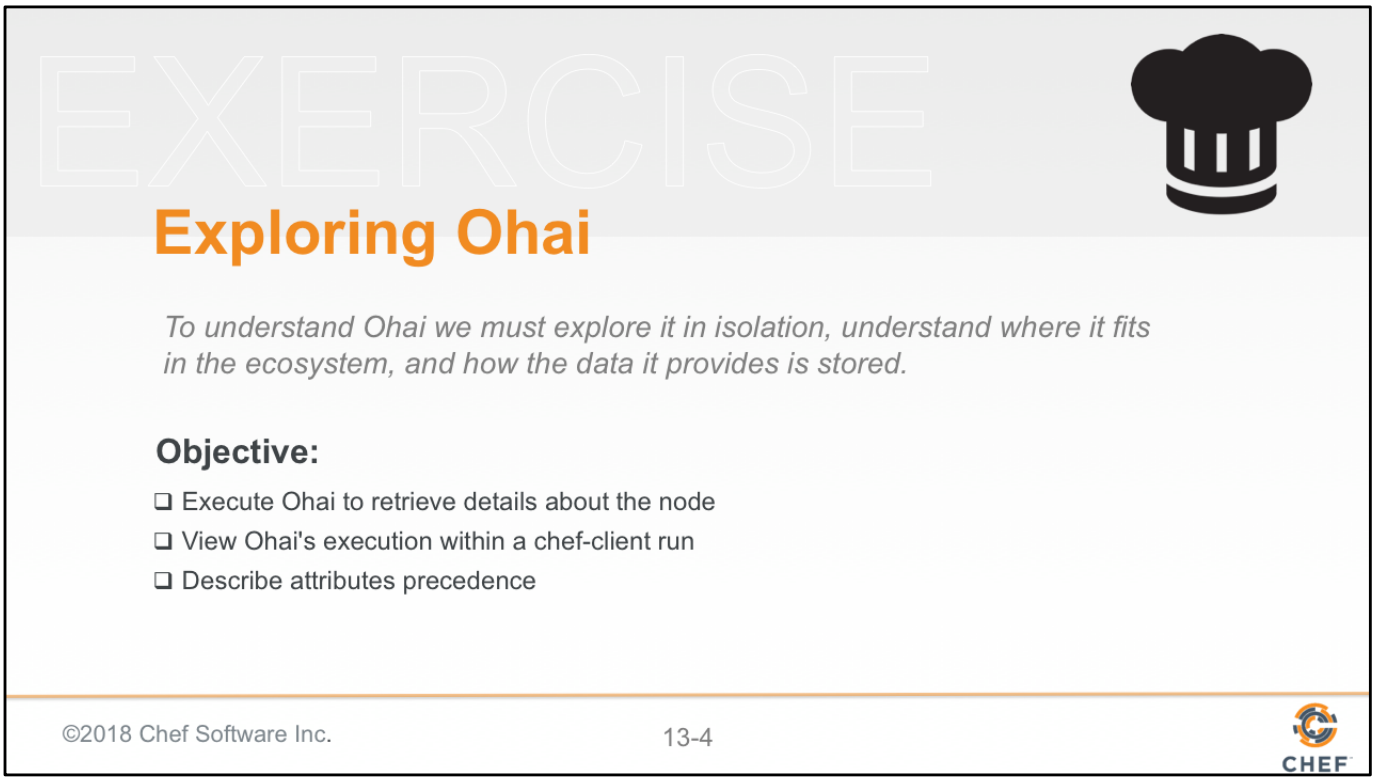
After completing this module, you should be able to:

- Execute the Ohai command-line tool to return an attribute
- Describe when Ohai is loaded in the chef-client run
- Describe when new attributes for the node are stored
- Describe precedence of attributes collected by Ohai

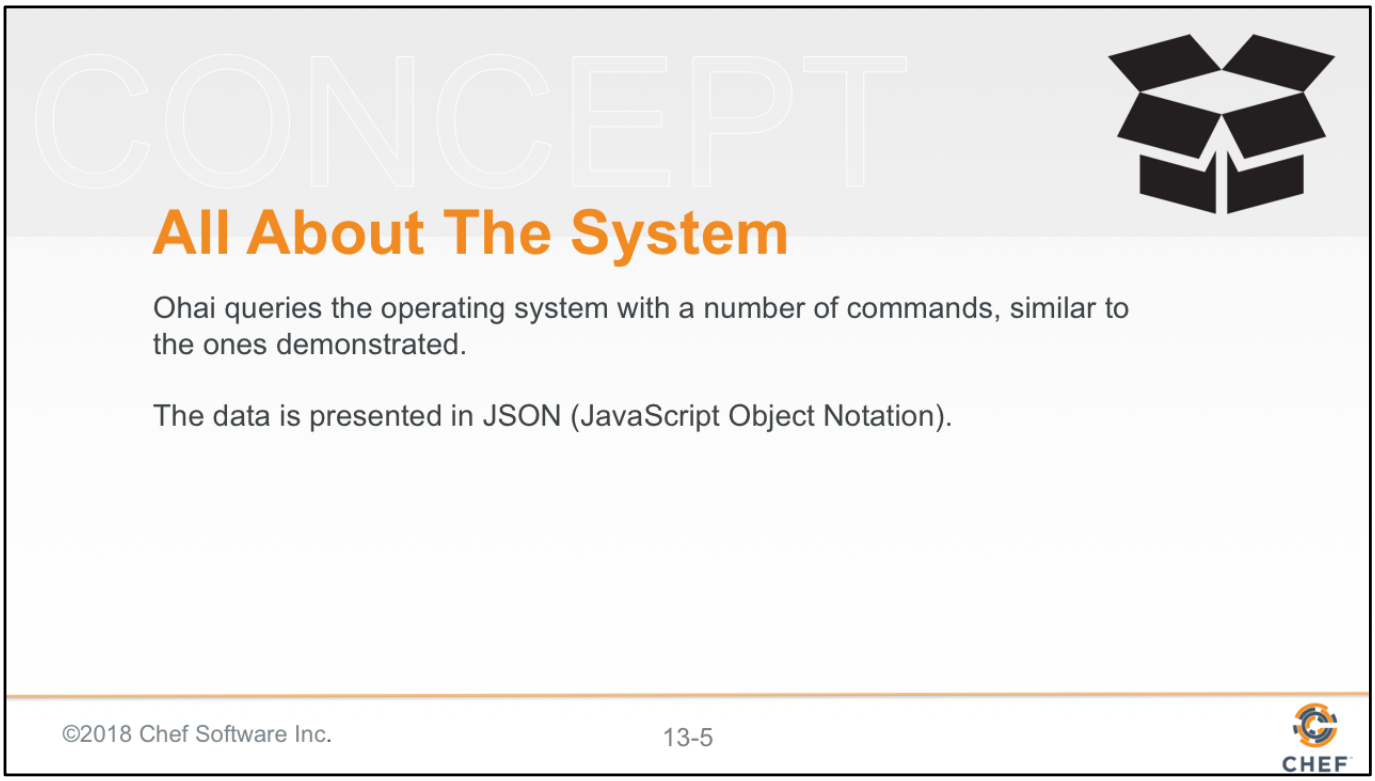
After completing this module you will be able to execute the Ohai command-line tool to return an attribute, describe when Ohai is loaded in the chef-client run, when new attributes for the node are stored in that chef-client run, and be able to describe the attribute precedence for attributes collect by 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. (Ohai is installed on a node as part of the chef-client install process.)



As a group we will explore using Ohai from the command-line then view how it is executed within a chef-client run and then talk about the attributes that it collects. We'll start with ohai the command-line tool.



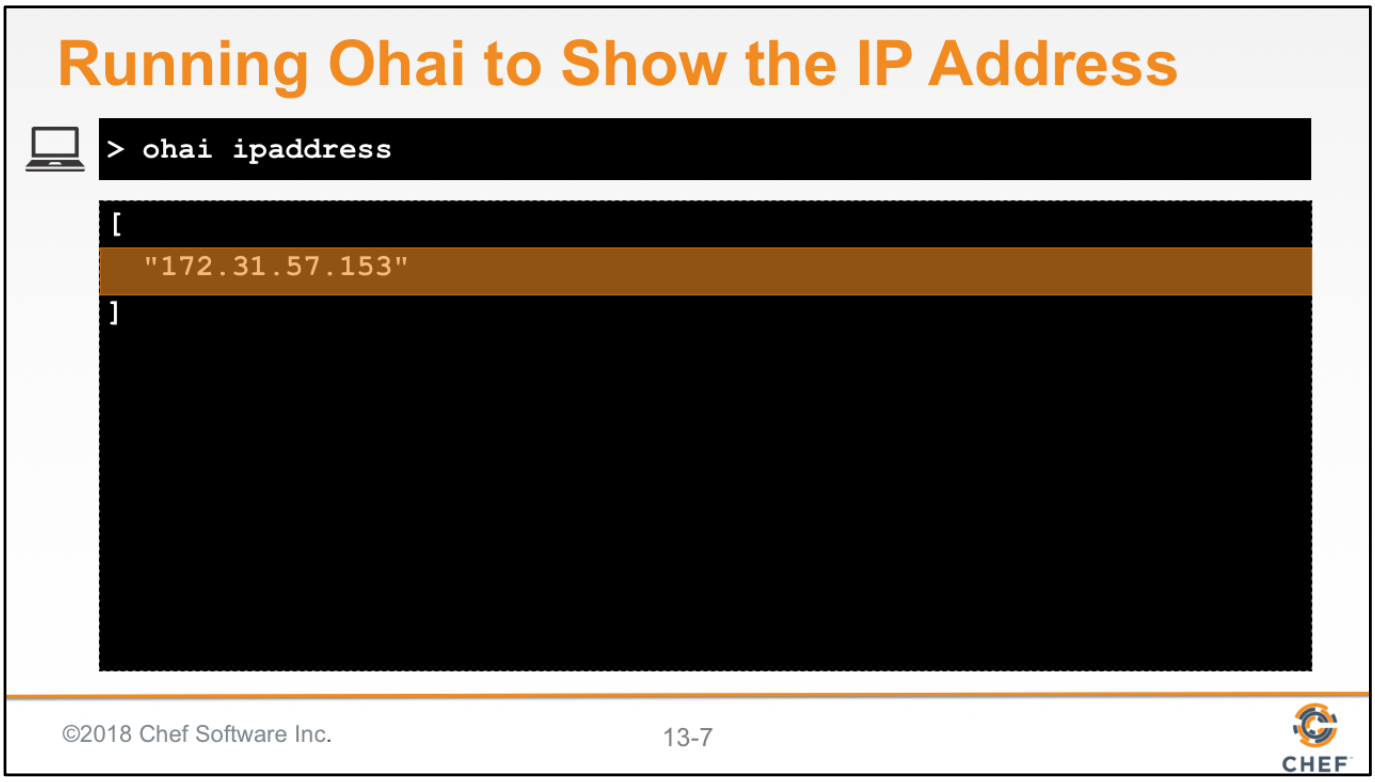
Ohai, the command-line application, will output all the system details represented in JavaScript Object Notation (JSON).

Running Ohai to Show All Attributes

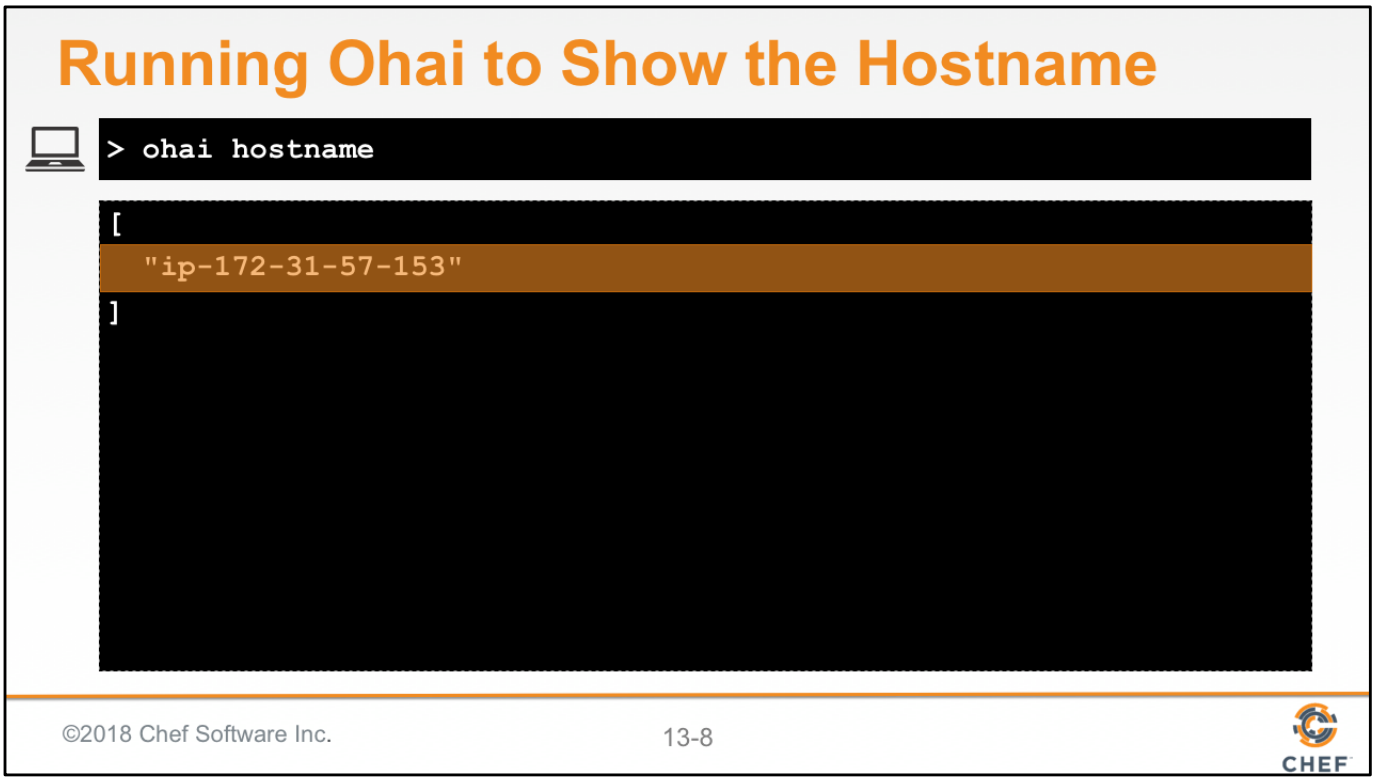
```
> ohai
```

```
{
  "kernel": {
    "name": "Linux",
    "release": "2.6.32-431.1.2.0.1.el6.x86_64",
    "version": "#1 SMP Fri Dec 13 13:06:13 UTC 2013",
    "machine": "x86_64",
    "os": "GNU/Linux",
    "modules": {
      "veth": {
        "size": "5040",
        "refcount": "0"
      },
      "ipt_addrtype": {
```


Ohai is also a command-line application that is part of the ChefDK. When you run it you will see the entire JSON representation of the system.



You can also run ohai with a parameter. In this case when we want only the ipaddress from the entire body of information we can provide it as a parameter.



Similar, we can specify the hostname to return only the hostname of the system.

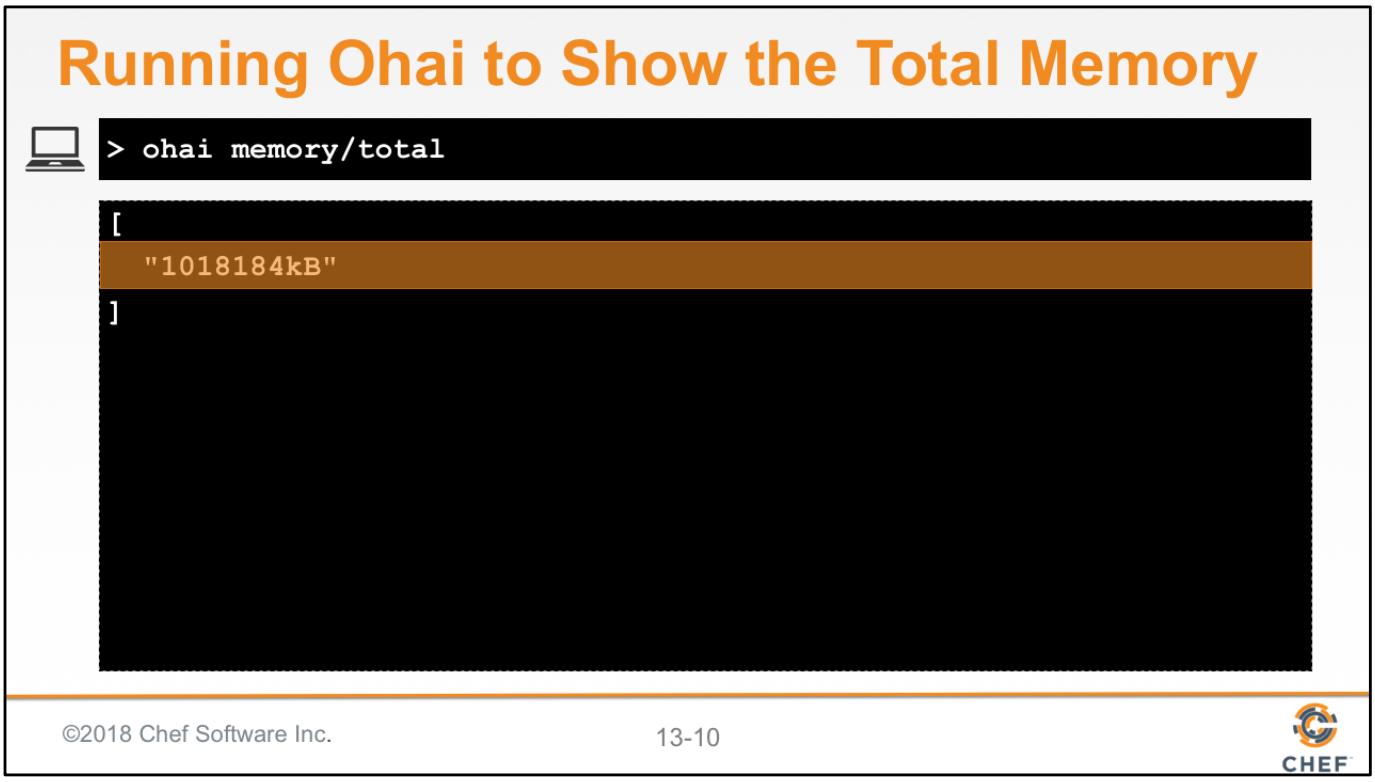
Running Ohai to Show the Memory

```
> ohai memory
```


```
{
  "swap": {
    "cached": "0kB",
    "total": "0kB",
    "free": "0kB"
  },
  "hugepages": {
    "total": "0",
    "free": "0",
    "reserved": "0",
    "surplus": "0"
  },
  "total": "1018184kB",
  "free": "280972kB",
  "buffers": "54340kB",
```


CHEF

When we ask for the memory of the system we receive a hash that contains a number of keys and values.



We can grab a single value, like the total memory, by specifying a slash between the top-level key and the next level key underneath it. This command will return the total memory of the system.

Running Ohai to Show the CPU

```
> ohai cpu
```

```
{
  "0": {
    "vendor_id": "GenuineIntel",
    "family": "6",
    "model": "63",
    "model_name": "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz",
    "stepping": "2",
    "mhz": "2400.078",
    "cache_size": "30720 KB",
    "physical_id": "0",
```


CHEF

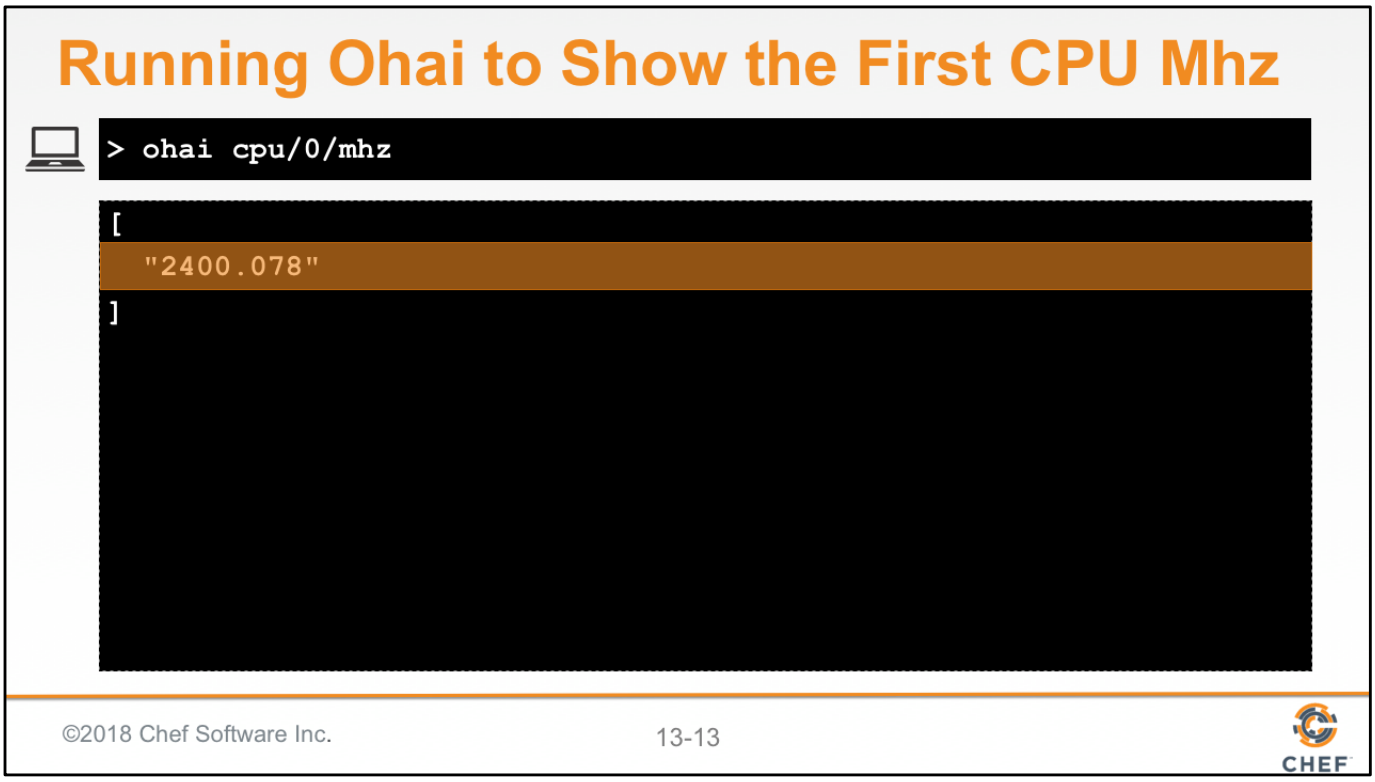
We can return all the details about the cpu. We see that there is one cpu, named '0', that contains more information.

Running Ohai to Show the First CPU

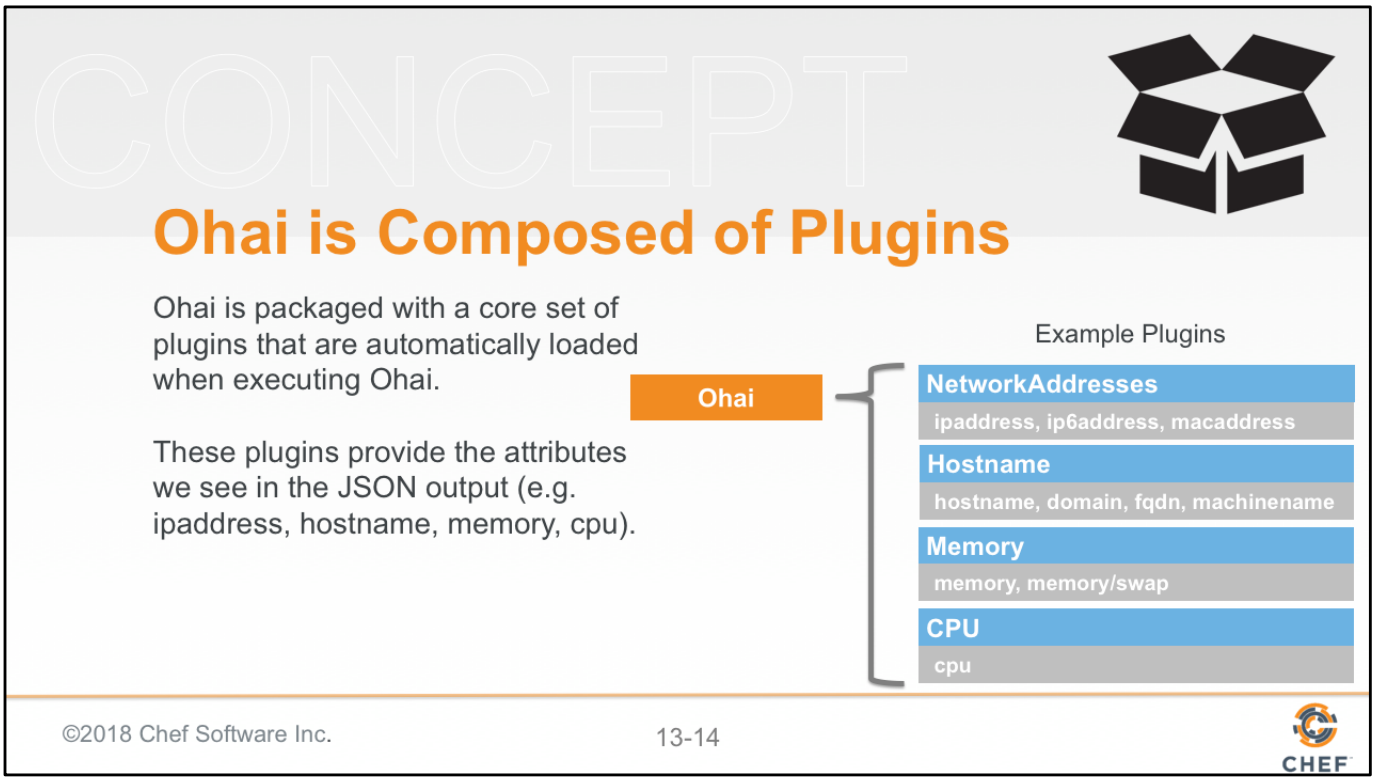
```
> ohai cpu/0
```

```
{
  "vendor_id": "GenuineIntel",
  "family": "6",
  "model": "63",
  "model_name": "Intel(R) Xeon(R) CPU E5-2676 v3 @ 2.40GHz",
  "stepping": "2",
  "mhz": "2400.078",
  "cache_size": "30720 KB",
  "physical_id": "0",
  "core_id": "0",
```

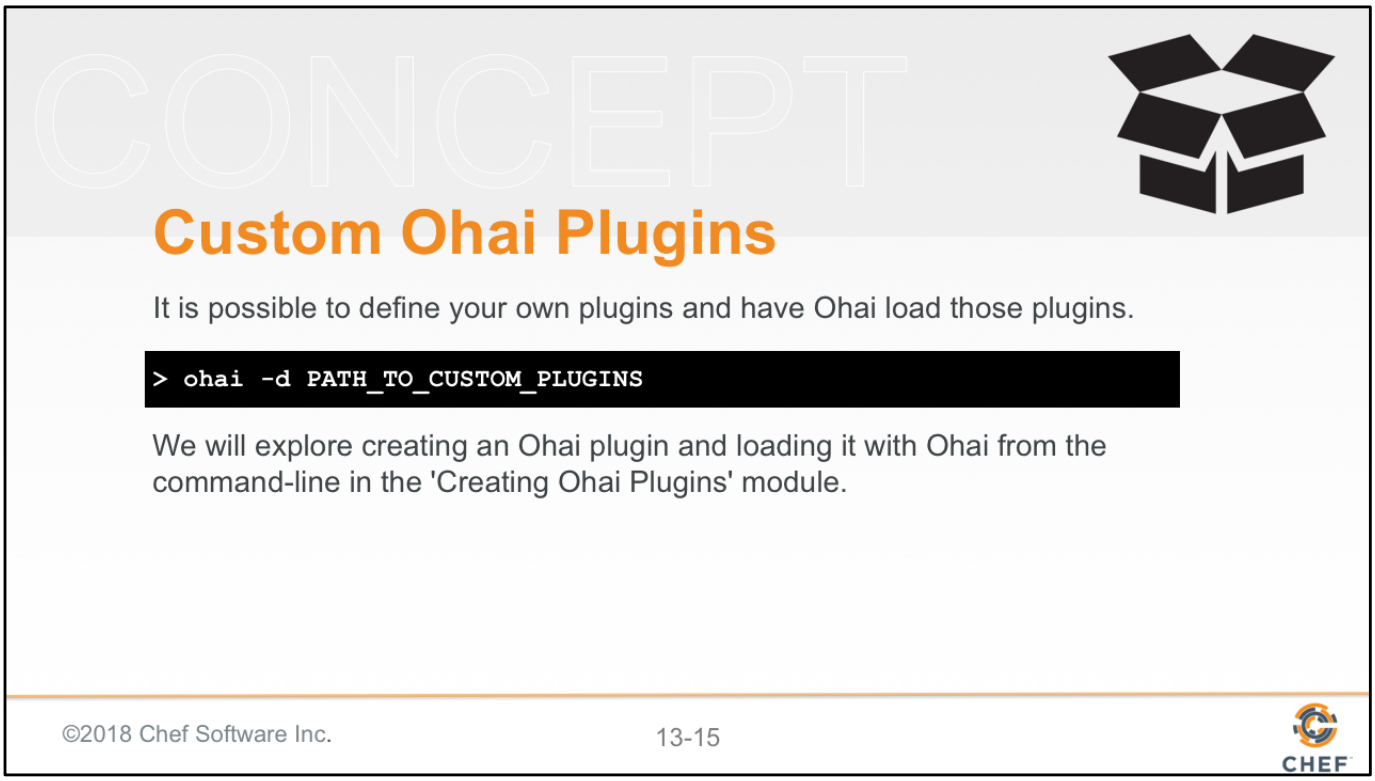

Here we are asking for all the details about the cpu named '0'.



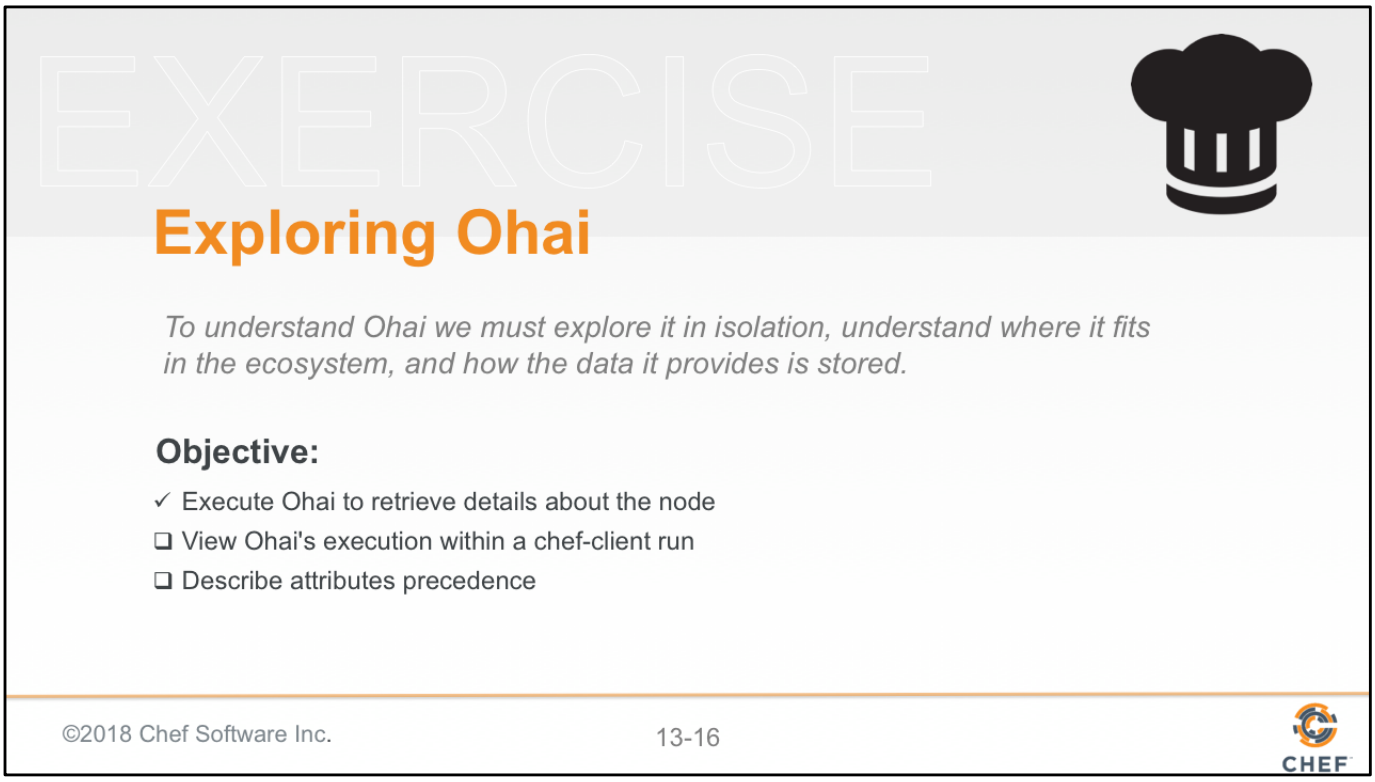
And finally if we wanted to display the Megahertz of that specific cpu we can append an additional key to the parameter.



Ohai is composed of plugins that collect these different attributes. When you execute Ohai it will load the core plugins that are packaged with it.



Ohai is composed of plugins that collect these different attributes. When you execute Ohai it will load the core plugins that are packaged with it.



Executing ohai from the terminal gives you an idea about all the data that Ohai can provide for a system. Now it is important to see where this data is captured in the chef-client run.

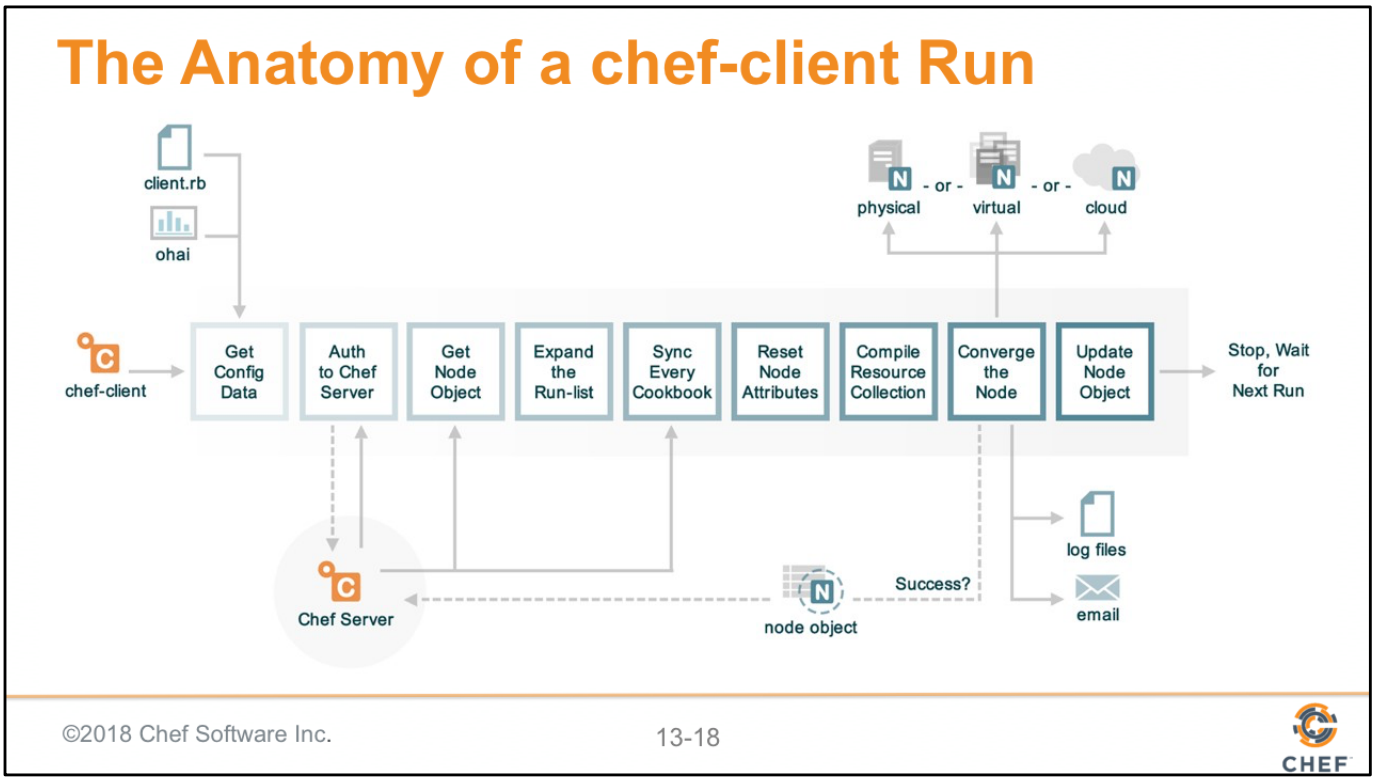
CONCEPT

chef-client

chef-client automatically executes ohai and stores the data about the node in an object we can use within the recipes. When the chef-client run completes successfully the details about the node are sent to the Chef Server.

http://docs.chef.io/ohai.html

CHEF



chef-client automatically loads the Ohai libraries and executes them to capture details about the system. These details are stored in the node object which is available in the recipes that we define. At the end of a successful chef-client run this node object is sent to the Chef Server.

Running Ohai in Code

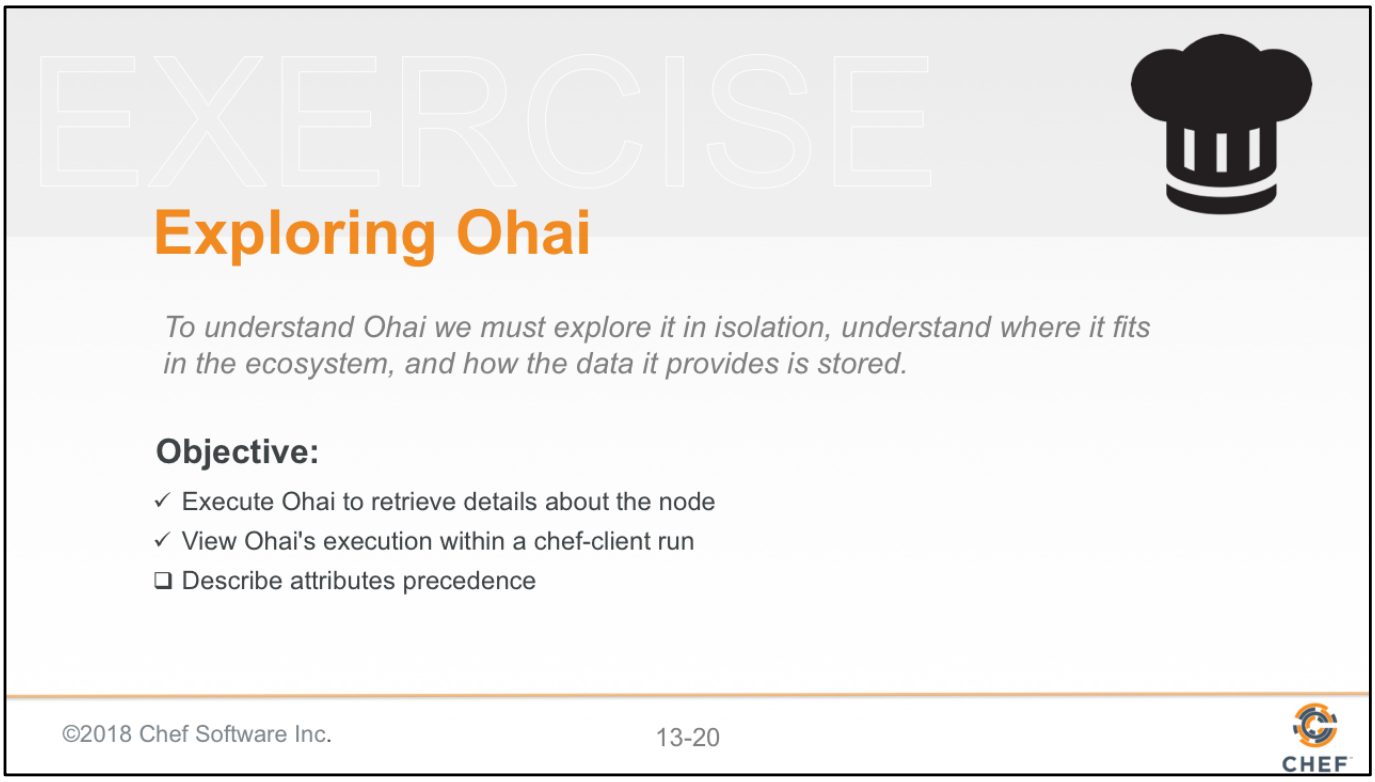
```
> chef exec pry
```

```
[1] pry(main)> require 'ohai'
=> true
[2] pry(main)> ohai = Ohai::System.new
=> #<Ohai::System:0x007fc62fadc490 @plugin_pat...@safe_run=true>>
[3] pry(main)> ohai.all_plugins('ipaddress')
[4] pry(main)> ohai.all_plugins('hostname')
[5] pry(main)> ohai.all_plugins('memory')
[6] pry(main)> ohai.all_plugins('cpu')
[7] pry(main)> ohai.all_plugins
[8] pry(main)> exit
```

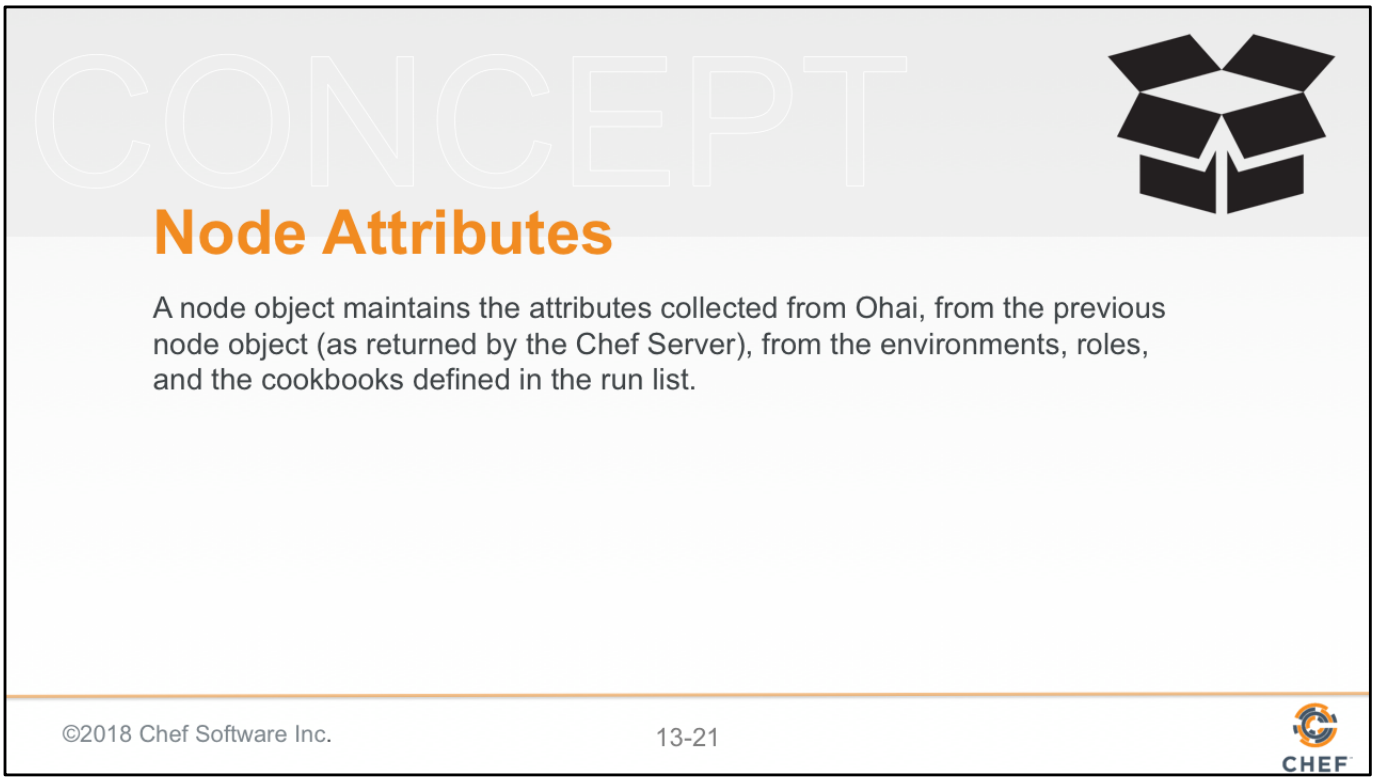

chef-client run ohai in code as one of it's first steps. We can examine how that is done with Pry. Pry can be used as a debugger and as a REPL (Read-Evaluate-Print-Loop) tool. We can run this to allow to explore how Ohai is executed by the chef-client application.

First launch the session by running the specified command. Within this interactive session you can load the Ohai gem with the require command, create a new Ohai System object, and then ask the ohai object to load specific plugins or all plugins through the 'all_plugins' method. When you are done you can exit by entering the command 'exit’.

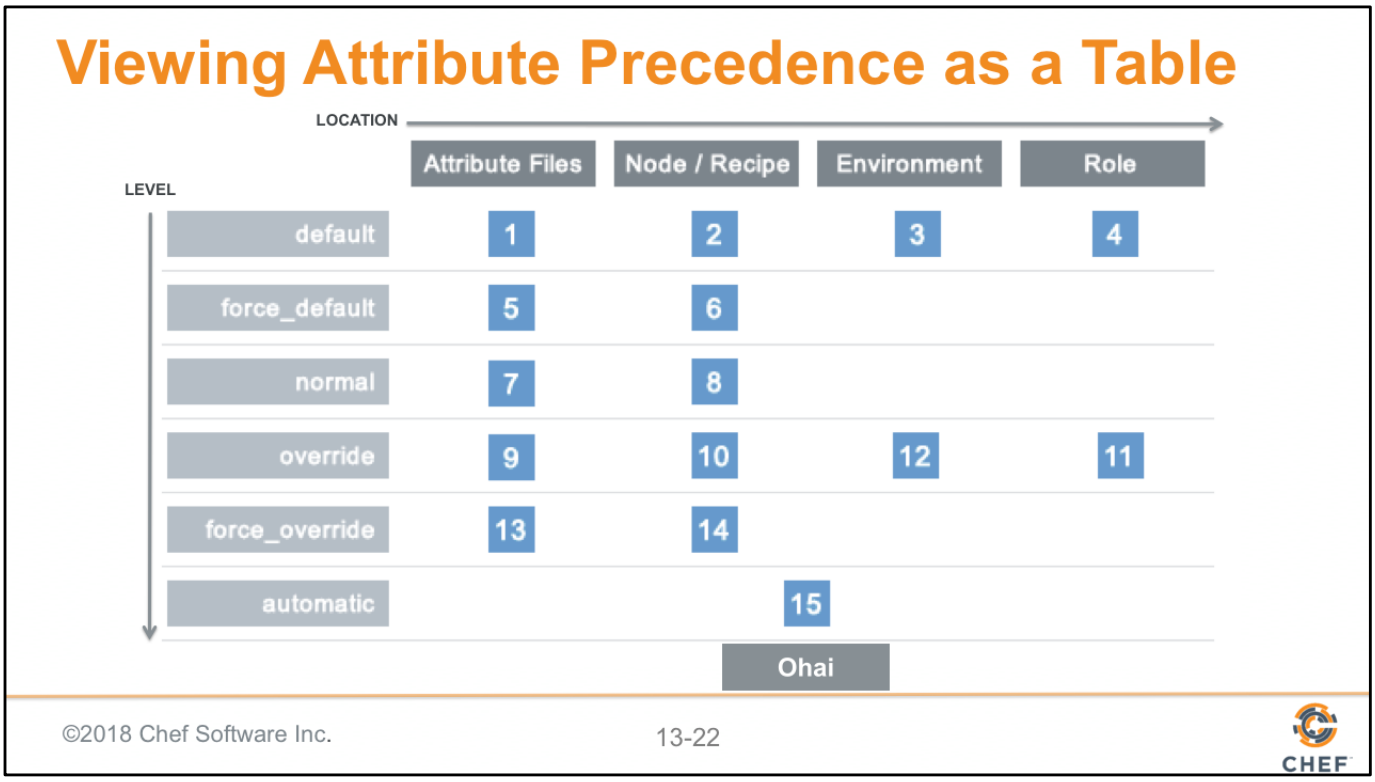
Type ‘q’ to exit large outputs



chef-client loads and executes Ohai within Ruby. Ohai returns Ruby object representations of the data that chef-client is able to evaluate and store within the node object. These attributes discovered by Ohai become attributes of the node object and it is important to take a quick moment to discuss how these attributes compare to the other attributes that may be defined in other locations within cookbooks, roles, and environments.



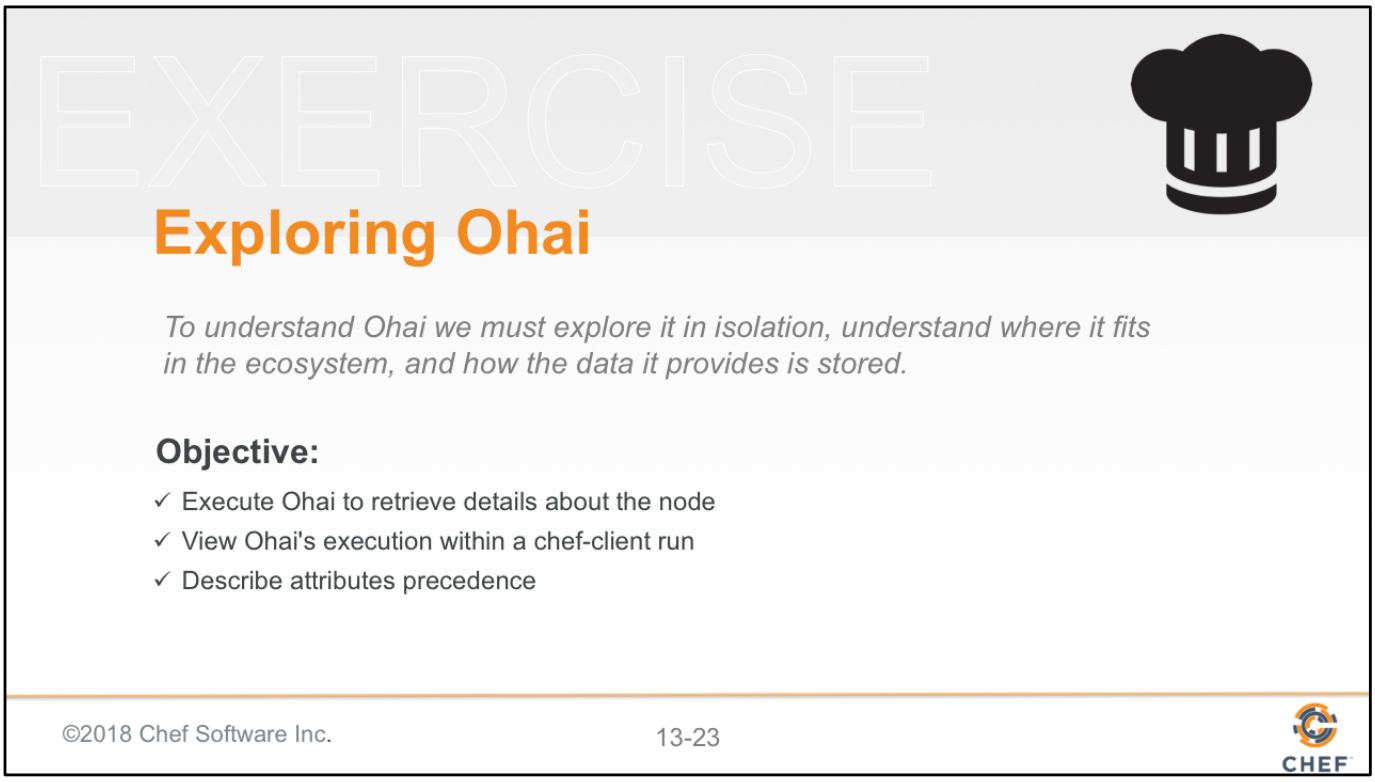
Later within the chef-client a node object is created with the attributes collected from Ohai, the values previously stored on the Chef Server, and then the attributes defined in the environments, roles and cookbooks described in the node's run list. The node prioritizes and gives precedence to the attributes collected by Ohai.



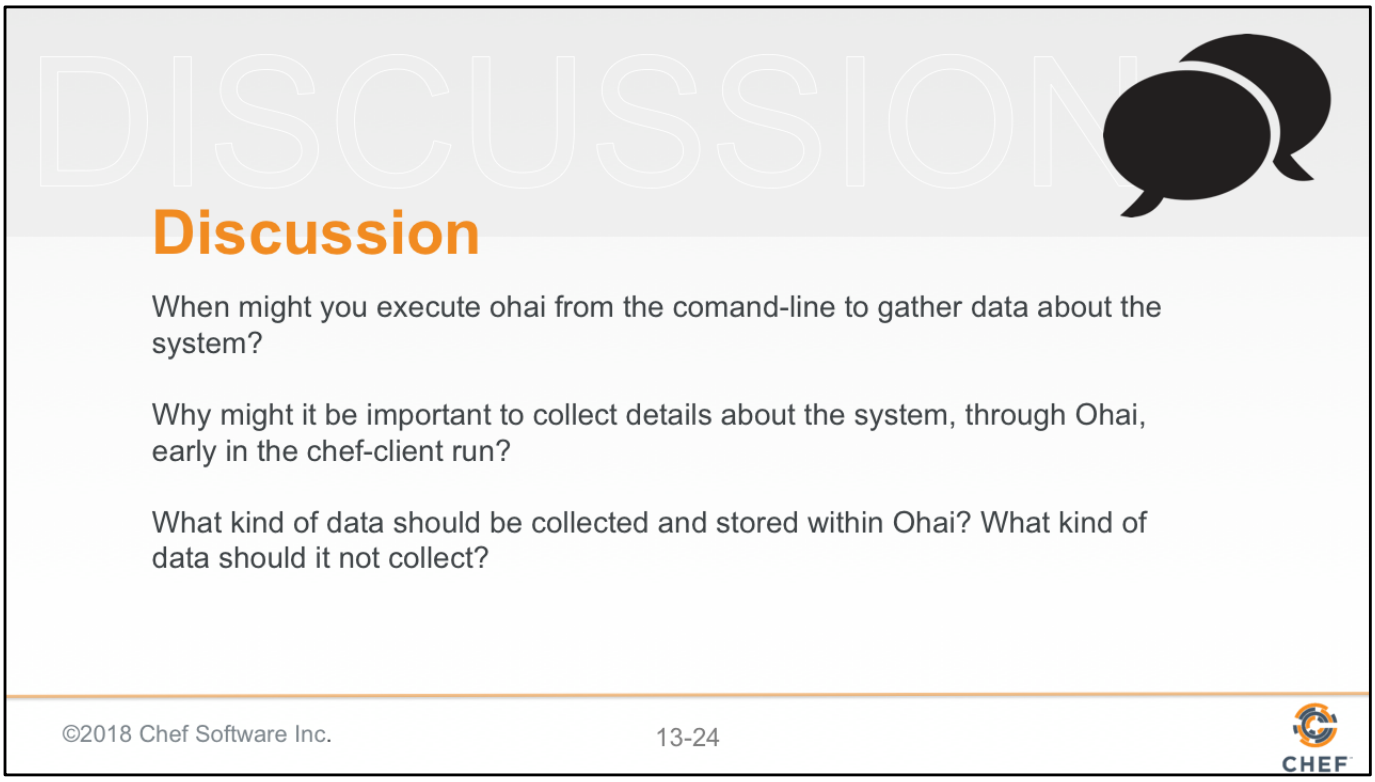
This is a table representation of the various levels of precedence that can be specified with the location in which it can be specified. The lower the value, the lower the precedence. The higher the value, the higher the precedence.

The attributes collected from Ohai are considered automatic attributes granting them the value of 15. This means all data collected through Ohai attributes cannot ever be overridden.

That should make sense based on the data that we have queried so far in this module (e.g. CPU, memory). Never would we want to have an attribute defined in a cookbook or environment override this data collected about our system. This is also important when considering whether you want to create an Ohai plugin. The kind of data that you want to collect should not be data that you will want to override as it is data that describes the system and not data that you want to configure the system.

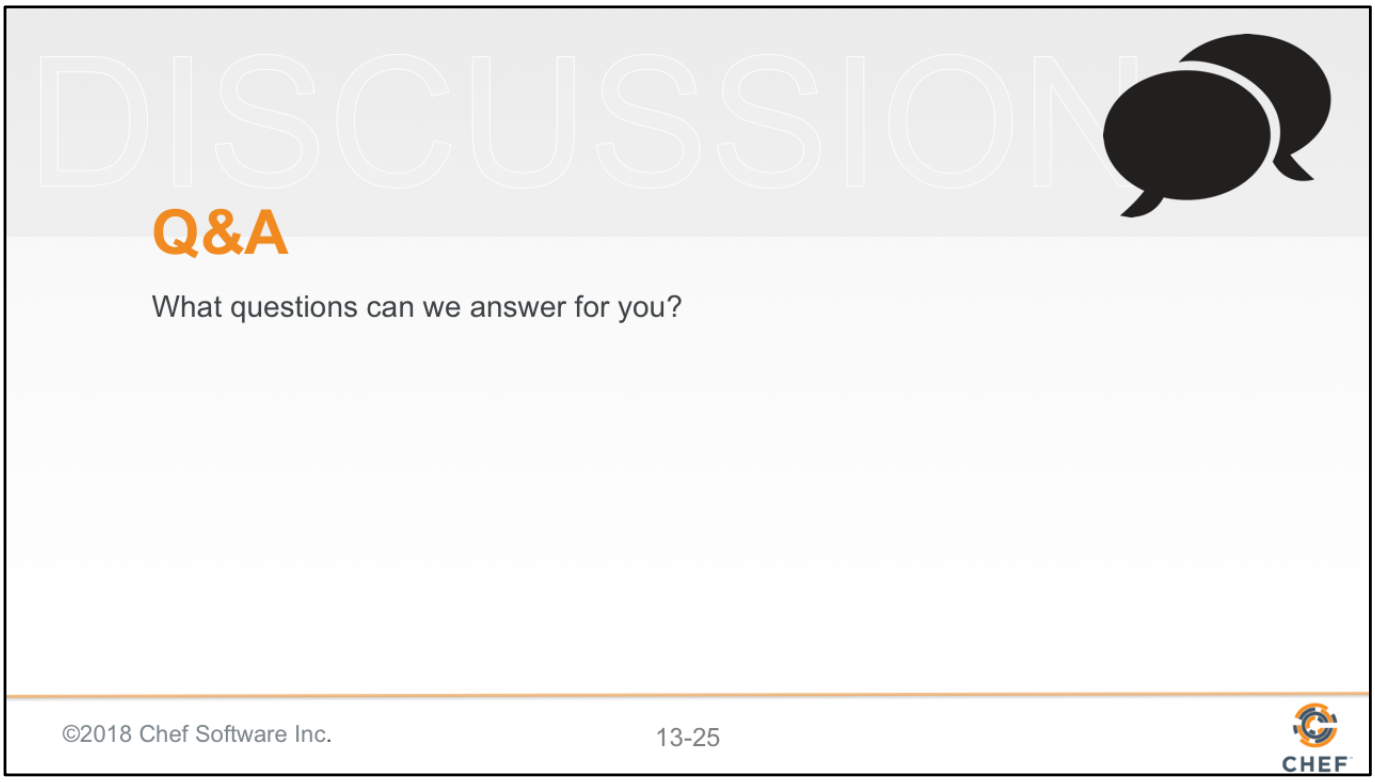


We have seen how to use Ohai as a command-line tool, explored how chef-client uses it, and seen the precedence level at which this data is stored. In the next module we will discuss Ohai's plugin history, its plugin structure, and the DSL (Domain Specific Language) it provides to express these plugins.



Let's finish with a discussion.

Instructor Note: With large groups I often find it better to have individuals turn to the individuals around them, form groups of whatever size they feel comfortable, and have them take turns asking and answering the questions. When all the groups are done I then open the discussion up to the entire group allowing each group or individuals to share their answers.



What questions can we answer for you?


CHEF™