Cisco's OpenStack Installer (COI) is almost entirely composed of Puppet modules with a small number of bash scripts. At its core, puppet is a system that enables users to describe the state of the system they are managing, and the puppet runtime will ensure that the system matches that state. To do this, puppet manages a number of what are termed "Resources". For example, puppet can ensure that a package is installed, that a file has certain contents, or that a service is running. In fact, for many simple deployments, these three resources will cover all of what is required. These resources can be grouped into classes and modules, to allow developers to separate concerns.

Before attempting to modify COI, it is recommended that developers are comfortable writing their own puppet modules. There is a wealth of information on the Puppetlabs site including a tutorial on how to work with Puppet.

COI Structure

COI currently uses ~30 Puppet modules, each managing a particular part of the deployment, plus a "manifests" package that ties together the modules. For example, the naginator module is used to install the nagios monitoring system on all nodes. In the manifests package there are puppet class and define elements that enable this module for each node. There are also two higher order classes called control and compute within the manifests, which are currently used to configure what is on openstack compute nodes and what is on the control nodes, like so:

/etc/puppet/manifests/core.pp

```
class control() {
  #the things we want on each openstack controller
}

class compute() {
  #the things we want on each openstack compute node
}
```

So to have a compute node, in `site.pp` we could have a line such as:

/etc/puppet/manifests/site.pp

```
node my_compute_node {
  class { compute: }
}
```

This allows us to easily configure the compute and control nodes separately. To have something on every node, such as nagios, we create a node type and inherit our nodes from that, like so:

/etc/puppet/manifests/core.pp

```
node base {
  class { "naginator::base_target":
  }
}
```

/etc/puppet/manifests/site.pp

```
node my_compute_node inherits base {
  class { compute: }
}
```

These are the two main mechanisms in place for controlling what is installed on the nodes managed by puppet. If you look through the folsom-manifests repository, you'll see that there are a number of differences to what is shown above:

1. There are a large number of arguments to each of the classes. This allows customisation without having to edit anything more than site.pp.
2. There is more than one base class. Because the build node sits outside the openstack cluster but is still managed by puppet, there is a `base` node type, then inheriting from that that there is `os_base` for openstack nodes and `build-node` for the build node.
3. Nodes seem to be defined twice in `site.pp`. This is because they are defined once for the build node via their out of band address so that it can PXE boot and install ubuntu, and once so that puppet can manage them.

Machine Provisioning

In addition to the management of all nodes (including itself) using puppet, the build node is responsible for booting and provisioning the base OS of the control and compute nodes. This is done using Cobbler, which is itself installed and configured using puppet. The first commands to install COI are to download the manifests folder and install the needed puppet modules to the build node, and after this a `puppet apply` is run locally on the build node. This `puppet apply` will install Cobbler, the Ubuntu 12.04 install image, and the puppetmaster along with some ancillary services such as apt-cache and DNS. The compute and control nodes will be populated into the Cobbler database and can then be turned on and provisioned. A quick look at `cobbler_node.pp` shows this basic structure:

```
define cobbler_node($node_type, $mac, $ip, $power_address, $power_id = undef,
  $power_user = 'admin', $power_password = 'password', $power_type = 'ipmitool' ) {
  cobbler::node {
  ...
  # Arguments for the node class defined in the cobbler module
  ...
  }
}

node /cobbler-node/ inherits "base" {
  ...
  # Things that will be installed on any node that inherits "cobbler-node"
  ...
}
```

It is important to draw the distinction between these two sections. The first section is used to inform the build node that it needs to put something into Cobbler to manage a compute or control node. Both types of nodes are provisioned in the same way, so they are simply called `"cobbler_node"` at this point. In `site.pp`, they are seen like this:

```
node /build-node/ inherits master-node {
  cobbler_node { "control-server": node_type => "control", mac => "00:11:22:33:44:55", ip => "192.
                           power_address  => "192.168.242.110", power_user => "admin", power_passw
                           power_type => "ipmitool" }
  ...
  # Other compute / control nodes
  ...
}

node build-server inherits build-node { }
```

So we can see that the build server now knows which machines to provision via these `cobbler_node` definitions, and that is the reason for #3 in the previous section: puppet needs to know where a machine is to

manage it, and Cobbler needs to know the out of band access details in order to provision it. The reason for inheriting `master-node` is that `master-node` is in `core.pp` with all the ancillary classes for the build node, and `master-node` inherits from `cobbler-node` which is the second section in `cobbler_node.pp` and contains only the Cobbler module definition to install Cobbler on the build node.

More Detail

If you are planning on making serious modifications to COI, more detail on each of the modules is available here