4. Reclass Models

Reclass is a Recursive External node Classifier. In other words, it is a tool for hierarchical inventory management of server nodes used in conjunction with configuration management tools to enable Infrastructure as Code.

Chapter Details	
Chapter Goal	Use Reclass models to deploy a simple environment
Chapter Sections	*3.1. Reclass Model* *3.2. SaltStack And Reclass Model*

4.1. Reclass Model

Reclass allows us to define node information such as its variables, roles, and cluster groups in a central and hierarchical fashion. Salt can leverage Reclass to define its static files such as pillars and grains in this central location using Reclass models. The following keywords appear in Reclass YAML files:

- class (*category, tag, or feature that applies to node*)
- environment (*salt environment*)
- node (*salt minion*)
- application (*salt state*)
- parameters (*salt pillars*)

This model enables class inheritance which we will explore by restructuring our previously deployed application using Reclass model. As a reference, the following is the top file `/srv/salt/top.sls` we created previously:

```
base:
  '*':
    - edit
    - ntp
  'node1':
    - apache2
    - php
  'node2':
    - mysql
```

References:

Visit the Reclass documentation: http://reclass.pantsfullofunix.net

Step 1 Log in to your **master** environment and switch to root user privilege:

```
stack@lab:~$ ssh master
stack@master's password: b00tcamp
stack@master:~$ sudo su
[sudo] password for stack: b00tcamp
root@master:/home/stack#
```

Similar to when directory structure was made for Salt files, we must create Reclass directories for two items:

- `/srv/salt/reclass/nodes`
- `/srv/salt/reclass/classes`

Step 2 Create a new directory `/srv/salt/reclass` for Reclass inventory:

```
root@master:/home/stack# mkdir /srv/salt/reclass
```

Step 3 Create a new directory for nodes definition:

```
root@master:/home/stack# mkdir /srv/salt/reclass/nodes
```

Step 4 Create a new directory for classes definition:

```
root@master:/home/stack# mkdir /srv/salt/reclass/classes
```

Step 5 Create a new class file `common.yml` in the directory `/srv/salt/reclass/classes/` with the following content:

```
applications:
  - edit
  - ntp
```

Creating the `common.yml` file in the `/srv/salt/reclass/classes` directory allows us to reference this file as a parent class later on to apply its applications to various nodes.

Step 6 To define the `master` node using Reclass model create a new file `master.yml` in the `/srv/salt/reclass/nodes/` directory with the following content:

```
environment: base
classes:
  - common
```

This model will inherit all applications and parameters from the class *common*.

Step 7 To define the `node1` node using the Reclass model create a new file `node1.yml` in the `/srv/salt/reclass/nodes/` directory with the following content:

```
applications:
  - apache2
  - php
classes:
  - common
parameters:
  keep_alive_timeout: 20
```

This model declares two **applications** to be installed: *apache2* and *php*. Applications are mapped to the states. **Parameter** is equivalent to Pillars - we will use the Reclass adapter to map the Pillars to Parameters later on. *node1* also inherits the *common* class.

Step 8 To define the `node2` node using the Reclass model create a new file `node2.yml` in the `/srv/salt/reclass/nodes/` directory with the following content:

```
classes:
  - common
applications:
  - mysql
```

Step 9 Install Reclass:

```
root@master:/home/stack# apt-get install reclass reclass-doc
```

Before applying the model Reclass will merge all inherited classes. During this process all list will be appended, dictionaries will be merged deeply and single parameters will be replaced. Execute following command to see merged model for *master* node

Step 10 View the merged Reclass model for *master* node:

```
root@master:/home/stack# reclass -b /srv/salt/reclass/ -n master
__reclass__:
  environment: base
  name: master
  node: ./master
  timestamp: Mon Apr 10 17:09:43 2017
  uri: yaml_fs:///srv/salt/reclass/nodes/./master.yml
applications:
- edit
- ntp
classes:
- common
environment: base
parameters: {}
```

Reclass first retrieves information about the specified node. Then, it iterates the list of classes defined in the node recursively, descending each class until a leaf node is reached. Then all information regarding the node is merged as displayed by the command.

Step 11 Run the same command for *node1* node:

```
root@master:/home/stack# reclass -b /srv/salt/reclass -n node1
__reclass__:
  environment: base
  name: node1
  node: ./node1
  timestamp: Mon Apr 10 17:10:36 2017
  uri: yaml_fs:///srv/salt/reclass/nodes/./node1.yml
applications:
- edit
- ntp
- apache2
- php
classes:
- common
environment: base
parameters:
  keep_alive_timeout: 20
```

Notice that the list of applications contains those merged from the *common* class.

Step 12 View the entire inventory merged:

```
root@master:/home/stack# reclass -b /srv/salt/reclass/ -i
...
```

4.2. SaltStack And Reclass Model

Reclass can be intergrated with several configuration management tools using a plugin model, called adapter. Using the Reclass salt.py adapter Salt can reference the Reclass inventory for use with Top files and Pillars.

Step 1 Create a symlink to the Salt adapter Python module for easy debugging and to see the adapter in action:

```
root@master:~# ln -s /usr/lib/python2.7/dist-packages/reclass/adapters/salt.py /usr/loc
root@master:~# chmod +x /usr/lib/python2.7/dist-packages/reclass/adapters/salt.py
```

Step 2 Check out the `top.sls` generated by the Reclass adapter:

```
root@master:~# salt-reclass -b /srv/salt/reclass/ --top
base:
  master:
  - edit
  - ntp
  node1:
  - edit
  - ntp
  - apache2
  - php
  node2:
  - edit
  - ntp
  - mysql
```

Step 3 Get the pillars from *node1* from Reclass model using adapter:

```
root@master:~# salt-reclass -b /srv/salt/reclass/ --pillar node1
__reclass__:
  applications:
  - edit
  - ntp
  - apache2
  - php
  classes:
  - common
  environment: base
  nodename: node1
keep_alive_timeout: 20
```

This output is in the format that is ready to be used by *Salt*.

Step 4 To configure the master to use Reclass models edit the file `/etc/salt/master` and append the following content to the end of file:

```
reclass: &reclass
  storage_type: yaml_fs
  inventory_base_uri: /srv/salt/reclass

ext_pillar:
  - reclass: *reclass

master_tops:
  reclass: *reclass
```

Step 5 Remove the `top.sls` file(s) in your `file_roots` directory. This is necessary due to edge cases with Reclass and the Top file.

```
root@master:~# rm /srv/salt/top.sls
```

Step 6 Last but not least, restart the Salt master

```
root@master:~# service salt-master restart
```

Now your Salt client is able to reference Reclass models.

Step 7 Show the top data that minions will use for highstate:

```
root@master:~# salt \* state.show_top

master:
    ----------
    base:
        - edit
        - ntp
node2:
    ----------
    base:
        - edit
        - ntp
        - mysql
node1:
    ----------
    base:
        - edit
        - ntp
        - apache2
        - php
```

Step 8 List pillar items using the Salt client

```
root@master:~# salt \* pillar.items
```

Step 9 Apply the highstate using Salt client

```
root@master:~# salt \* state.apply

...
          ID: /etc/apache2/apache2.conf
Function: file.managed
  Result: True
 Comment: File /etc/apache2/apache2.conf updated
 Started: 16:57:35.737497
Duration: 31.081 ms
 Changes:
          ----------
          diff:
              ---
              +++
              @@ -102,7 +102,7 @@
               # KeepAliveTimeout: Number of seconds to wait for the next request from
               # same client on the same connection.
               #
              -KeepAliveTimeout 60
              +KeepAliveTimeout 20

               # These need to be set in /etc/apache2/envvars
```

```
                User ${APACHE_RUN_USER}
...
```

In `/srv/salt/reclass/nodes/node1.yml` file, we changed the `KeepAliveTimeout` value to **20** as shown by the *diff* in the output.

All summaries should return with Success.

Congrautlations, you have finished the *3. Reclass Models* chapter!

Checkpoint:

- Create Reclass models for existing applications
- Setup Salt master to reference Reclass models
- Utilize Salt client to apply highstate