9. DriveTrain Operations

DriveTrain is the heart of Mirantis Cloud Platform that gives you the flexibility to tune your services, versions, configuration, and topology of your clouds to suit the changing needs of your business.

In this chapter, we will learn how to setup and utilize the most important components of DriveTrain: Git, Gerrit, and Jenkins

Chapter Details	
Chapter Goal	Use MCP DriveTrain components to manage MCP infrastructure
Chapter Sections	*9.1. Setting up Git & Gerrit* *9.2. Understanding Pipelines* *9.4. Chapter Summary and Useful Notes* *9.3. Update OpenStack Configuration*

9.1. Setting up Git & Gerrit

The power of Infrastructure as Code cannot be fully realized until we have a proper version control system and process in place. In this section, we will begin by setting up Git and Gerrit using Reclass model. Particularly, we want to create a new project in Gerrit and setup a local Git repository; this will allow us to commit, push, and pull changes related to our Reclass model for the cluster.

Step 1 Log-in to your lab environment through SSH:

```
user@laptop:~$ ssh stack@e2.edu.mirantis.com -p <port>
password: <password>
stack@kvm01:~$
```

Step 2 Become sudo, then log-in to the **cfg01** (salt-master) node:

```
stack@kvm01:~$ sudo -i
root@kvm01:~# ssh cfg01
root@cfg01:~#
```

Step 3 Navigate to `/srv/salt/reclass/classes/cluster/lab28/cicd` directory, which is the path for the model related to DriveTrain cluster:

```
root@cfg01:~# cd /srv/salt/reclass/classes/cluster/lab28/cicd
```

Step 4 Open the `control/leader.yml` file in this directory and add a new class under the following line:

```
root@cfg01:/srv/salt/../cicd/# vim control/leader.yml

classes:
...
  - system.gerrit.client
  - system.gerrit.client.project.ci
```

```
  - cluster.lab28.infra
  - cluster.lab28.cicd.control
# Add the following line
  - cluster.lab28.cicd.control.reclass-gerrit
```

Of course, we will need to define this file as `control/reclass-gerrit.yml` to utilize it. If you need a refresher on Reclass models, visit *4.3.2. Configure OpenStack using Salt and Reclass*

Step 5 Create the `control/reclass-gerrit.yml` file:

```
root@cfg01:/srv/salt/../cicd/# touch control/reclass-gerrit.yml
```

So what should go in this *reclass-gerrit.yml* file? Our goal is the following:

- Create a new Gerrit project
- Clone the existing Reclass model provided by Mirantis Training (specific to this lab)

As we did in the previous chapter, we can reference github and or other levels of our model to understand the formatting of what we need. As you can see under *classes*, in the *control/leader.yml* model is *system.gerrit.client.project.ci*. Let's take a look at the file in the system model, because represents the Gerrit *project* we want to create.

Step 6 Open the following file and view its contents `/srv/salt/reclass/classes/system/gerrit/client/project/ci.yml`:

```
root@cfg01:/srv/salt/../cicd/# cat /srv/salt/reclass/classes/system/gerrit/client/proje
parameters:
  _param:
    gerrit_pipeline_library_repo: https://github.com/Mirantis/pipeline-library
    gerrit_mk_pipelines_repo: https://github.com/Mirantis/mk-pipelines
    gerrit_decapod_pipelines_repo: https://github.com/mateuszlos/decapod-pipelines
  gerrit:
    client:
      project:
        mcp-ci/pipeline-library:
          enabled: true
          description: Jenkins pipeline libraries
          upstream: ${_param:gerrit_pipeline_library_repo}
          access: ${gerrit:client:default_access}
          require_change_id: true
          require_agreement: false
          merge_content: true
        mk/mk-pipelines:
          enabled: true
          description: Jenkins pipelines
          upstream: ${_param:gerrit_mk_pipelines_repo}
          access: ${gerrit:client:default_access}
          require_change_id: true
          require_agreement: false
          merge_content: true
    ...
```

There are various Gerrit projects configured in this system model - which are created by default when this lab was deployed. We will leverage the same format in the cluster model to create the *reclass-gerrit* project.

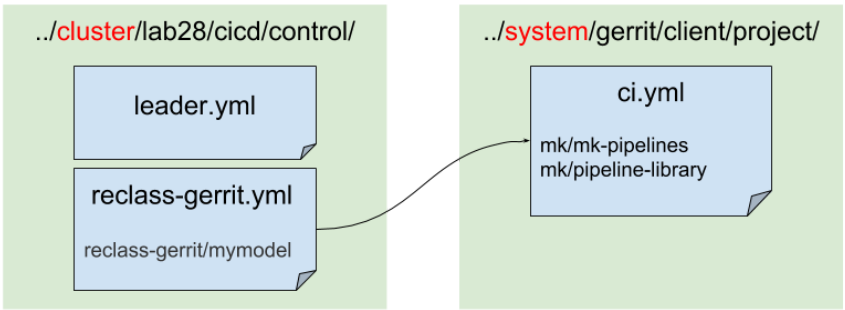
Step 7 Open `control/reclass-gerrit.yml` for editing and populate the file with the following content:

```
root@cfg01:/srv/salt/../cicd/# vim control/reclass-gerrit.yml

parameters:
  _param:
    gerrit_mcp_reclass: https://bitbucket.org/mirantis-training/mcpops-reclass/
  gerrit:
    client:
      project:
        reclass-gerrit/mymodel:
          enabled: true
          description: MCP Ops Reclass Model
          upstream: ${_param:gerrit_mcp_reclass}
          access: ${gerrit:client:default_access}
          require_change_id: true
          require_agreement: false
          merge_content: true
```

Save and exit the file when you are done.

- *gerrit_mcp_reclass*: This parameter is the upstream repository of reclass model used in this lab
- *reclass-gerrit/mymodel*: This will be the project name in Gerrit to be created

> **Notes:**
>
> https://bitbucket.org/mirantis-training/mcpops-reclass/ is where Mirantis Training is hosting the Reclass model used for your lab environment. If you already have a MCP deployment, you may have an upstream repository of your own containing the model relevant to the deployment.



As the diagram indicates, so far we have created a new file *reclass-gerrit.yml* in the cluster level with definition to create a new Gerrit project cloning the initial model from the training repository. Reclass will render pillar data by combining the different layers of the model. When Salt executes the *gerrit* state, it will create the Gerrit project we defined in *reclass-gerrit.yml*

Step 8 Sync all Salt resources then apply the **gerrit.client** state on the **cid01** node:

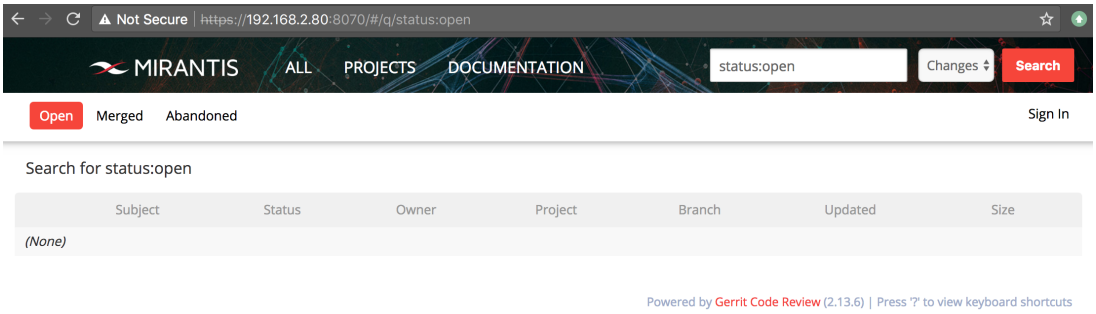
```
root@cfg01:~# salt \* saltutil.sync_all
```

```
root@cfg01:~# salt 'cid01*' state.apply gerrit.client
...
Summary for cid01.trainings.local
-------------
Succeeded: 23 (changed=5)
```

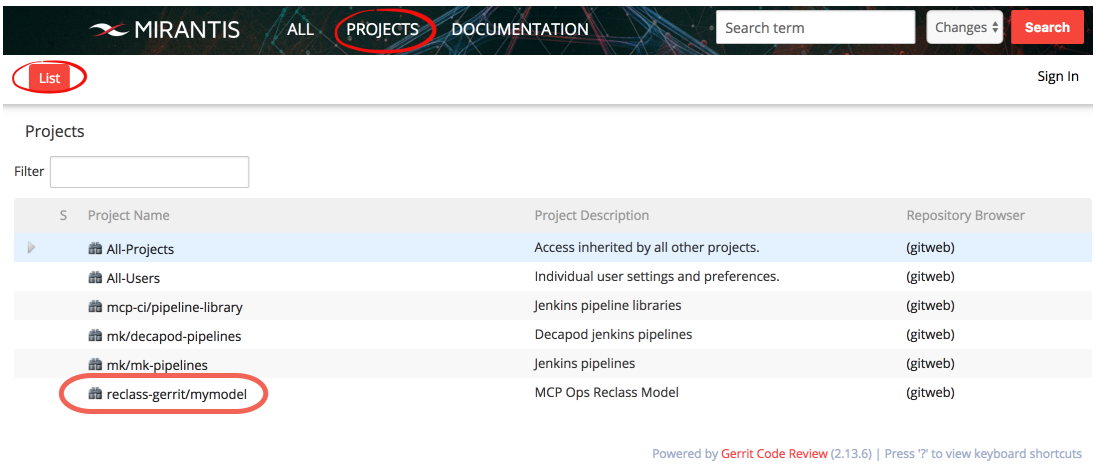
```
Failed:     0
-------------
Total states run:     23
Total run time:   12.889 s
```

After the magical journey is finished with success, verify that your new project was created through the following steps.

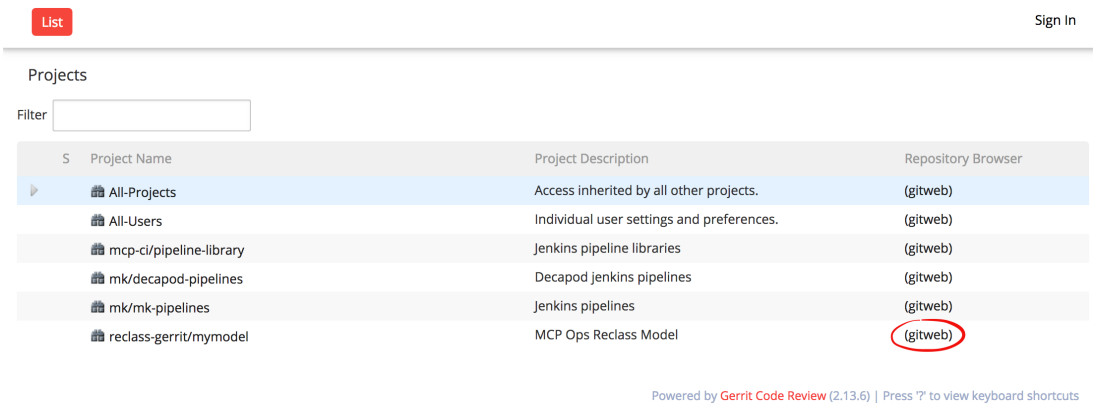
Step 9 Open a **VNC** connection to your lab and open a browser to view the Gerrit user interface at `https://192.168.2.80:8070` ; When you land on this page, you will be viewing as guest. *If your browser returns Your connection is not private message, click Advanced > Proceed to 192.168.2.80 (unsafe)*:



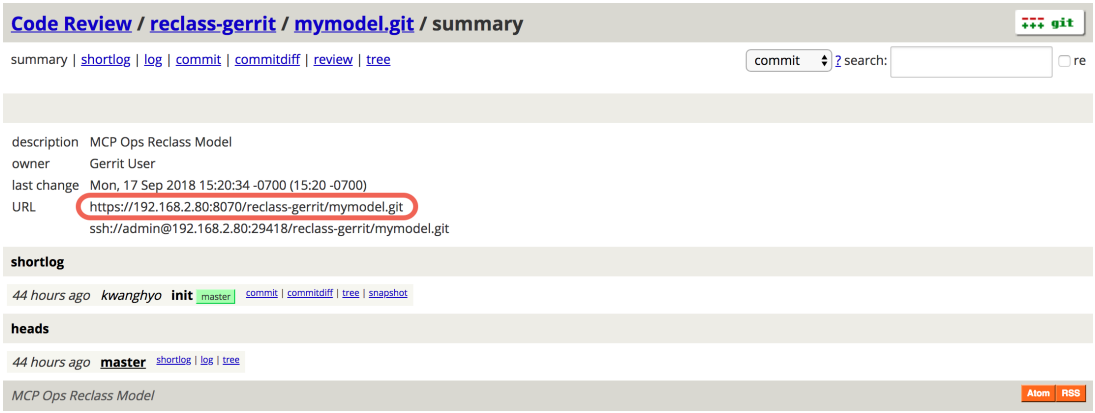
Step 10 Click on **PROJECTS** > **List** from the navigation menu. You should see your newly created Gerrit project from the list!:



Step 11 Click on **(gitweb)** button next to the *reclass-gerrit/mymodel* project and sign-in using credentials **admin** / **r00tme**:



Step 12 This is the code review page for your project. Copy the git URL https link:




```
https://192.168.2.80:8070/reclass-gerrit/mymodel.git
```

This is your internal git repository containing your latest model cloned from upstream (public) repository. In the upcoming steps, you will configure your infrastructure to reference this internal repository - for committing, pushing, and pulling changes.

First, consider the scenario where multiple engineers (including yourself) are modifying service configurations or infrastructure topology via Reclass model; you will need your own local copy of the repository. To simulate this scenario, you will first clone from the local repo and consider it your “development” location.

Step 13 Clone a copy of the local repository in the `/root/` directory of the **cfg01** node. Disable SSL verify to ignore expired certificates:

```
root@cfg01:~# export GIT_SSL_NO_VERIFY=true
```

Clone repo:

```
root@cfg01:~# git clone https://192.168.2.80:8070/reclass-gerrit/mymodel.git
Cloning into 'mymodel'...
remote: Counting objects: 74, done
remote: Finding sources: 100% (74/74)
remote: Total 74 (delta 10), reused 74 (delta 10)
Unpacking objects: 100% (74/74), done.
Checking connectivity... done.
```

Lastly, let’s change the remote origin of `/srv/salt/reclass` directory to the local repository we just setup - right now its origin is a different repository which was used when configuring your lab.

Step 14 Navigate to `/srv/salt/reclass` and change its origin to *https://192.168.2.80:8070/reclass-gerrit/mymodel.git*:

```
root@cfg01:~# cd /srv/salt/reclass
```

```
root@cfg01:/srv/salt/reclass# git remote set-url origin https://192.168.2.80:8070/recla
```

Step 15 Before merging with the new remote, commit the changes you made before to the cluster model:

```
root@cfg01:/srv/salt/reclass# git add .
root@cfg01:/srv/salt/reclass# git commit -m "New gerrit project"
```

Step 16 Pull the model from the new remote origin:

```
root@cfg01:/srv/salt/reclass# git pull
```

You may have a conflict in the `classes/cluster/lab28/cicd/control/leader.yml` file; Open this file and remove any conflicts then commit your change before proceeding to the next steps.

> **Notes:**
>
> Merge conflicts begin with chevrons such as "<<<<<<<< HEAD" and includes "=========" and ">>>>>>>>" to denote where the conflict exists. Remove these markings and add these files.

Step 17 Push your merged changes to the new remote. Use **admin** / **r00tme** as your credentials:

```
root@cfg01:/srv/salt/reclass# git push
...
Username for 'https://192.168.2.80:8070': admin
Password for 'https://admin@192.168.2.80:8070': r00tme
...
To https://192.168.2.80:8070/reclass-gerrit/mymodel.git
01e3f0e..6bd765c  master -> master
```

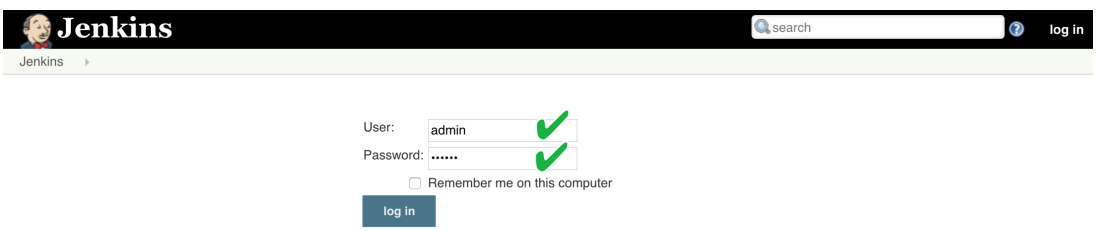
9.2. Understanding Pipelines

Jenkins pipelines is a way to organize a procedure in a repeatable and reliable templated fashion. For example, if you want to change a cloud configuration, you would make corresponding commits in your model repository, commit the code, and an automated trigger will test and deploy this change to the infrastructure.

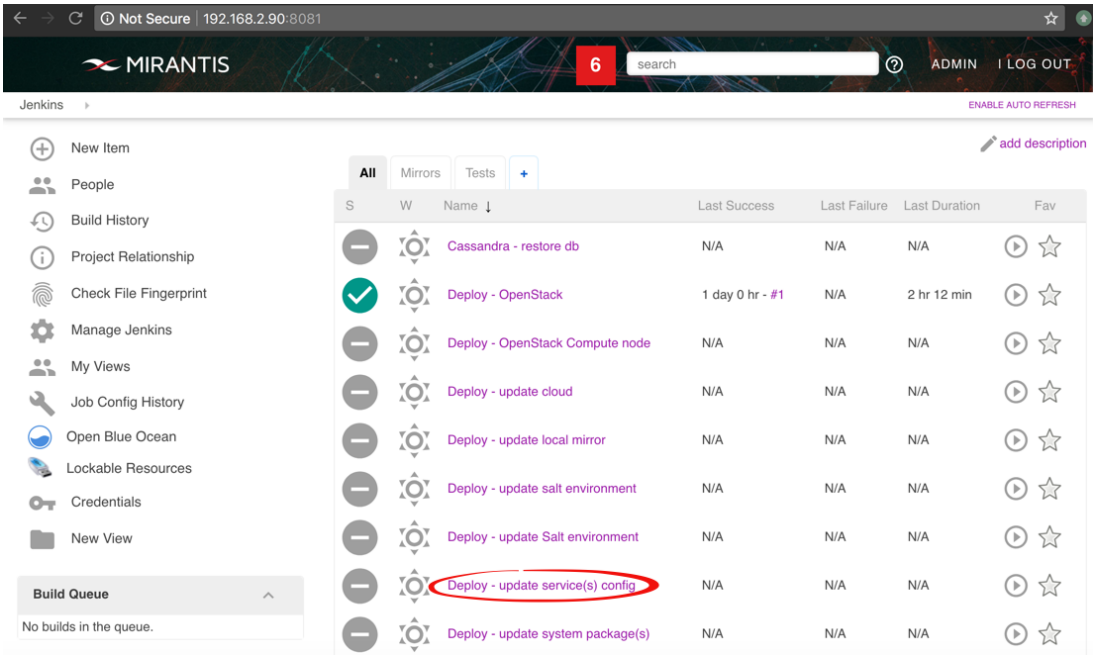
In this lab, you will be introduced to the details of how pipelines work, configure a cloud resource in your model, approve your own changes in gerrit, then manually run a pipeline called *Deploy - update service(s) config*.

Let's get started!

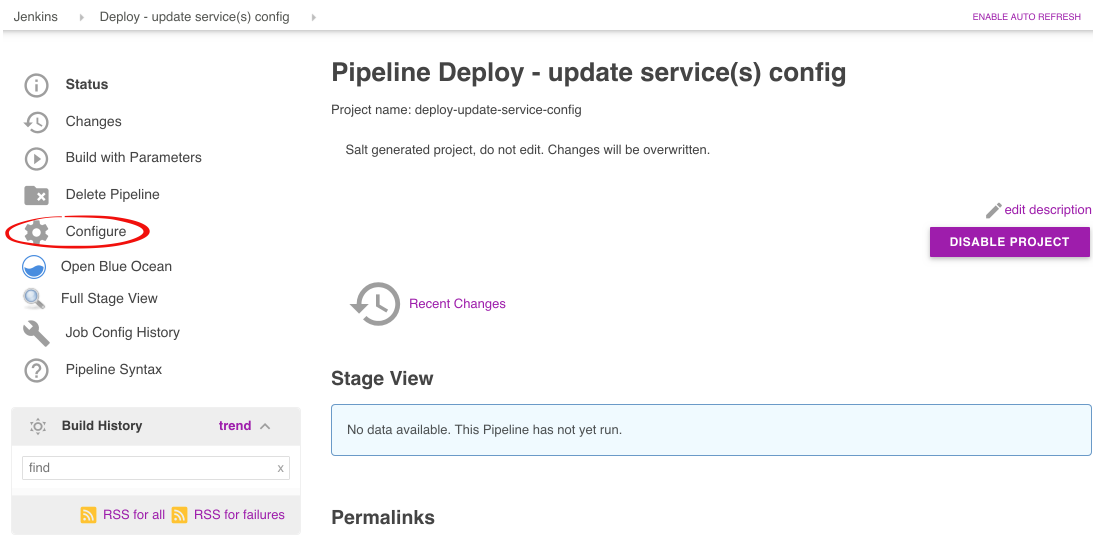
Step 1 Open a VNC connection with your lab and open the browser. Navigate to `https://192.168.2.80:8081` to open Jenkins UI. Log-in usinge *admin* / *r00tme* credentials.



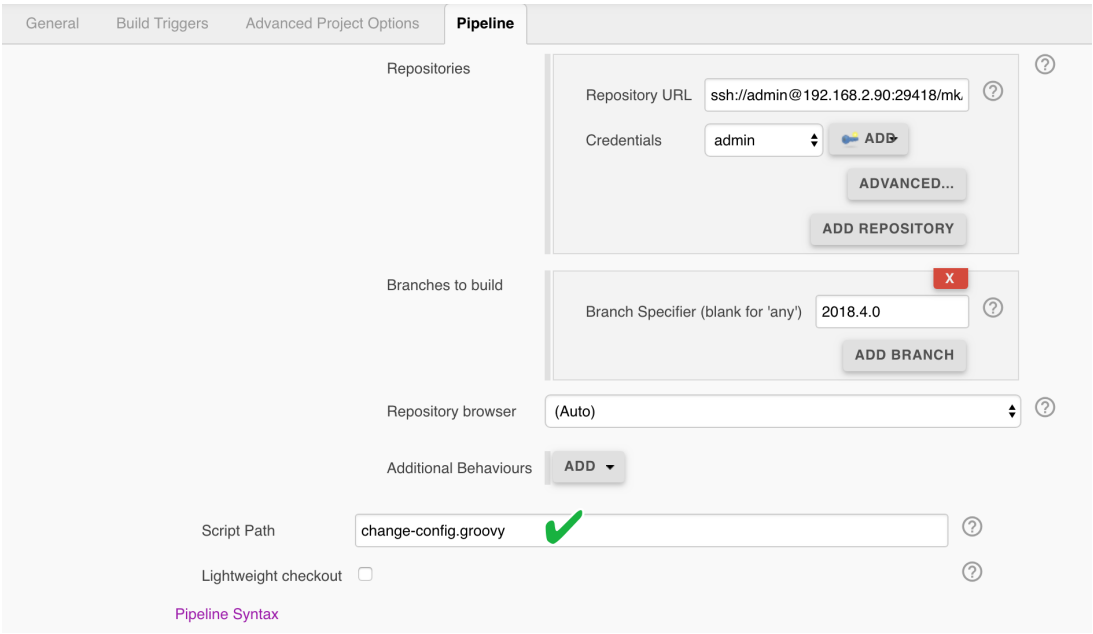
Step 2 Find the pipeline called **Deploy - update service(s) config** and click on the name to view the details:



Step 3 Click on **Configure** button on the left hand side to view more details about what this pipeline will run:



Step 4 Notice the parameters which can be passed in to this pipeline. Scroll down to the **Pipeline** section and take note of the *Script Path*:



change-config.groovy script contains the logic which is the essence of this pipeline. Let's open the file and take a look. The location of this file is currently local; we cloned the publicly available pipeline from: `https://github.com/Mirantis/mk-pipelines` into `/home/repo/mk/mk-pipelines`

Step 5 Open the *change-config.groovy* file and look for the parameter *PULL_MODEL* being used:

```
root@cfg01:~# cat /home/repo/mk/mk-pipelines/change-config.groovy
...
 if (common.validInputParam("PULL_MODEL") && PULL_MODEL.toBoolean() == true) {
            stage('Update the reclass cluster model') {
                def saltMasterTarget = ['expression': 'I@salt:master', 'type': 'compou
                result = salt.runSaltCommand(pepperEnv, 'local', saltMasterTarget, 'st
                                             null, "reclass.storage.data")
                salt.checkResult(result)
            }
        }
```

This shows us that if we check the **PULL_MODEL** parameter, it will run the salt state *reclass.storage.data*.

> **Important:**
>
> By understanding how pipelines work under-the-hood and knowing how to trace through the model, you will be well prepared for starting any debugging process and locating the source of potential issues.

Let's walk through the Salt state *reclass.storage.data*. Salt formulas are located in `/srv/salt/env/prd` ; we will start here. Keeping in mind our task is related to **PULL_MODEL** parameter, we should look for what repository this formula will pull.

Step 6 Open `/srv/salt/env/prd/reclass/storage/data.sls` for viewing:

```
root@cfg01:~# cat /srv/salt/env/prd/reclass/storage/data.sls
{%- from "reclass/map.jinja" import storage with context %}
{%- if storage.enabled %}
```

```
{%- if storage.data_source.engine == "git" %}

reclass_git_data_dir:
  git.latest:
  - name: {{ storage.data_source.address }}
  - target: {{ storage.base_dir }}
  - reload_pillar: True
  - rev: {{ storage.data_source.revision|default(storage.data_source.branch) }}
  {%- if grains.saltversion >= "2015.8.0" %}
  - branch: {{ storage.data_source.branch|default(storage.data_source.revision) }}
  {%- endif %}
  - force_reset: {{ storage.data_source.force_reset|default(False) }}

{%- elif storage.data_source.engine == "archive" %}
...
```

Amongst the Jinja templating, we can recognize that this state will look through *storage.data_source* pillar to determine the different resources to pull. Notice that it will only pull if *storage.data_source.engine == "git"*. Let's take a look at this pillar in our **cfg01** node to see its values.

Step 7 List the Pillar items on the **cfg01** node with the key *reclass:storage:data_source*:

```
root@cfg01:~# salt 'cfg01*' pillar.items reclass:storage:data_source
cfg01.trainings.local:
    ----------
    reclass:storage:data_source:
        ----------
        address:
            https://github.com/Mirantis/mk-lab-salt-model.git
        branch:
            master
        engine:
            local
```

In this case, Reclass data source address is *https://github.com/Mirantis/mk-lab-salt-model.git*. Thus pulling the repository before running our pipeline could be problematic. Let's find where this is defined, then change its value to our git repository. Furthermore, the *engine* key is set to *local*. We need to change this to *git*.

Step 8 Grep through the Reclass model and look for the github link referenced from the previous step:

```
root@cfg01:~# cd /srv/salt/reclass ; grep -r \
              https://github.com/Mirantis/mk-lab-salt-model.git

context-2018.4.0.yml:  reclass_repository: https://github.com/Mirantis/mk-lab-salt-mode
classes/cluster/lab28/infra/config.yml:
     reclass_data_repository: "https://github.com/Mirantis/mk-lab-salt-model.git"
```

This output tells us two things:

- Model repository was initially defined as part of the context by default
- We can change this in the cluster model

Step 9 Open the `classes/cluster/lab28/infra/config.yml` and change the *reclass_data_repository* value to point to the gerrit link. Then change the *reclass:storage:data_source:engine* to *git* because this is a *git* type repository:

```
root@cfg01:/srv/salt/reclass# vim classes/cluster/lab28/infra/config.yml
# Change the following line
reclass_data_repository: "https://192.168.2.80:8070/reclass-gerrit/mymodel.git"

# Scroll down and change the 'engine' keyword to 'git'
reclass:
  storage:
    data_source:
      engine: git
```

Step 10 Refresh the Salt resources then run the *reclass* state to apply your changes:

```
root@cfg01:/srv/salt/reclass# salt '*' saltutil.sync_all
```

```
root@cfg01:/srv/salt/reclass# salt '*' state.apply reclass
...
Summary for cfg01.trainings.local
-------------
Succeeded: 36
Failed:     0
-------------
Total states run:     36
Total run time:    5.561 s
```

Notes:

Only cfg01 node will report changes. If you are unsure about which node should have the state applied to, use the '*' to select all nodes. Salt state's idempotent nature will ensure it's safe to run states multiple times without negative side effects. Or, the model will not include such a state to run on that node.

Step 11 Check the pillar to ensure that the *reclass:storage:data_source* has changed to the expected address:

```
root@cfg01:/srv/salt/reclass# salt 'cfg01*' pillar.items reclass:storage:data_source
cfg01.trainings.local:
    ----------
    reclass:storage:data_source:
        ----------
        address:
            https://192.168.2.80:8070/reclass-gerrit/mymodel.git
        branch:
            master
        engine:
            git
```

Step 12 Once you have verified the changes, push the changes to master:

```
root@cfg01:/srv/salt/reclass# git add .
```

```
root@cfg01:/srv/salt/reclass# git commit -m "Reclass storage data_source changed"
root@cfg01:/srv/salt/reclass# git push
username: admin
password: r00tme
```

9.3. Update OpenStack Configuration

Great! Now that our Reclass storage is properly setup, let's proceed to make a change in our model related to OpenStack then use the pipeline to apply our changes.

Now we will put on our system administrator hat and make changes in the "developer" machine - meaning `/root/mymodel` directory we cloned from the repository. This way you further acknowledge that you are not making changes directly to the model `/srv/salt/reclass` being referenced by Reclass on a live environment.

Step 1 Navigate to `/root/mymodel`, pull the latest changes from master, then open the `classes/cluster/lab28/openstack/control_init.yml` file and add a new flavor definition:

```
root@cfg01:~# cd /root/mymodel
```

```
root@cfg01:~/mymodel# git pull
```

```
root@cfg01:~/mymodel# vim classes/cluster/lab28/openstack/control_init.yml
# Underneathe the existing 'medium' flavor, add a 'large' flavor such as the following:
parameters:
  nova:
    client:
      server:
        admin_identity:
          flavor:
          ...
            large:
              flavor_id: auto
              ram: 2048
              disk: 5
              vcpus: 2
```

Step 2 Take a look at the changes you made:

```
root@cfg01:~/mymodel# git status
   modified:   classes/cluster/lab28/openstack/control_init.yml
```

Gerrit requires you to have *commit-msg* hooks when pushing changes to the repository. This edits the commit messages to insert a *change-Id* tag, a unique identifier to track commits across cherry-picks and rebases.

Thankfully Gerrit provides a hook-file which automatically generates this *commit-msg* hook for us in the form of a script. We just need to retrieve it and have git utilize this script.

Step 3 Make a request to Gerrit to retrieve the hook file and save it to `/root/mymodel/.git/hooks/commit-msg`:

```
root@cfg01:~/mymodel# curl --insecure -Lo /root/mymodel/.git/hooks/commit-msg \
          https://192.168.2.80:8070/tools/hooks/commit-msg

  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100  4693  100  4693    0     0   9199      0 --:--:-- --:--:-- --:--:--  9201
```

Step 4 Make the hook file executable:

```
root@cfg01:~/mymodel# chmod u+x .git/hooks/commit-msg
```

Notes:

For more details on commit-msg hooks, visit: https://gerrit-review.googlesource.com/Documentation/cmd-hook-commit-msg.html

```
root@cfg01:~/mymodel# git add classes/cluster/lab28/*
root@cfg01:~/mymodel# git commit -m "Add new flavor and change git repo"
```

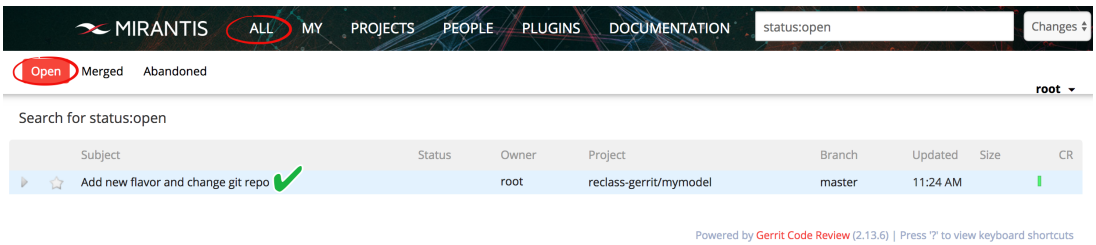
Once you have committed your changes, you will push to the *staging area*. In gerrit, this is denoted by *HEAD:refs/for/<branch>*. Here, the code review process will take place before being merged with master.

Step 5 Push your changes using the credentials *admin* user and *r00tme* password to *HEAD:refs/for/master*:

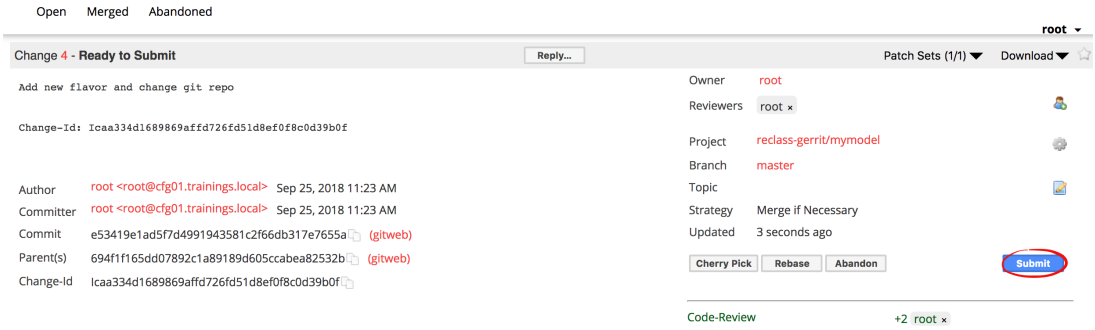
```
root@cfg01:~/mymodel# git push origin HEAD:refs/for/master

  Username for 'https://192.168.2.80:8070': admin
  Password for 'https://admin@192.168.2.80:8070': r00tme
...
remote: New Changes:
remote:   https://192.168.2.80:8070/4 Add new flavor and change git repo
remote:
To https://192.168.2.80:8070/reclass-gerrit/mymodel.git
 * [new branch]      HEAD -> refs/for/master
```

Step 6 Navigate back to your VNC window and open the Gerrit UI located at `https://192.168.2.80:8070` and log-in with the credentials *admin* user and *r00tme* password.

Step 7 Navigate to **ALL** > **Open**, then click on the subject line to review the change:



Step 8 Review the changes then click **Code-Review+2**, then click **Submit**:

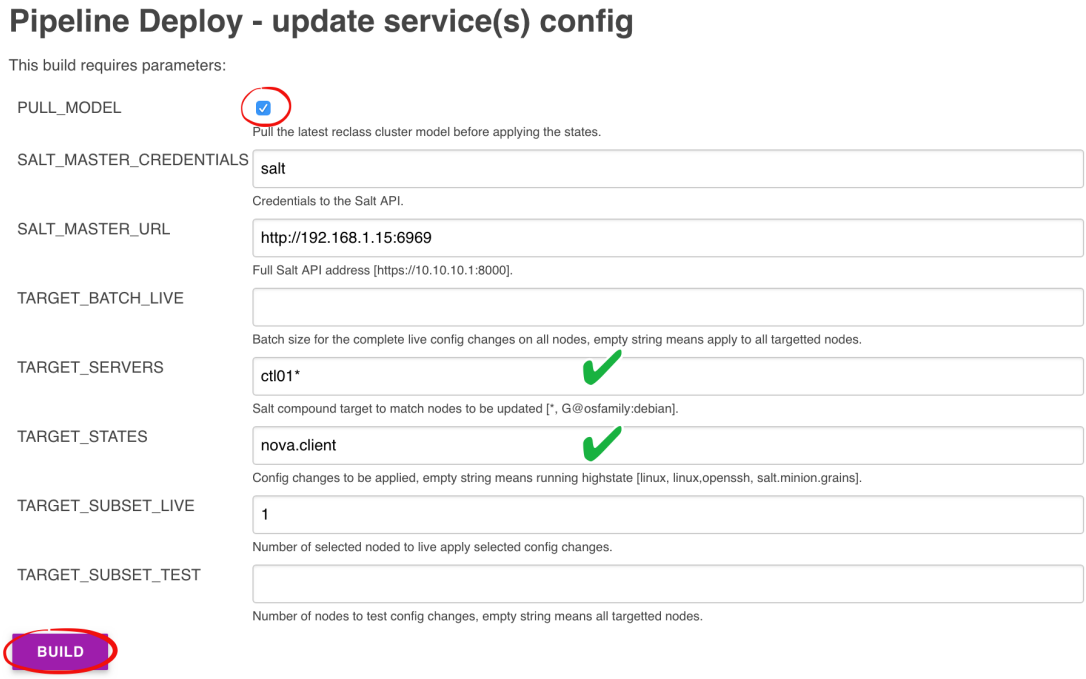


Since you are the *admin* user within a privileged *group*, you have the option to submit with **+2**. Other groups may only allow its users to submit **+1** meaning more users will have to review the code before it is submitted to master.

Step 9 Open the Jenkins UI on the VNC browser at `https://192.168.2.80:8081`. Log-in with the credentials *admin* / *r00tme*

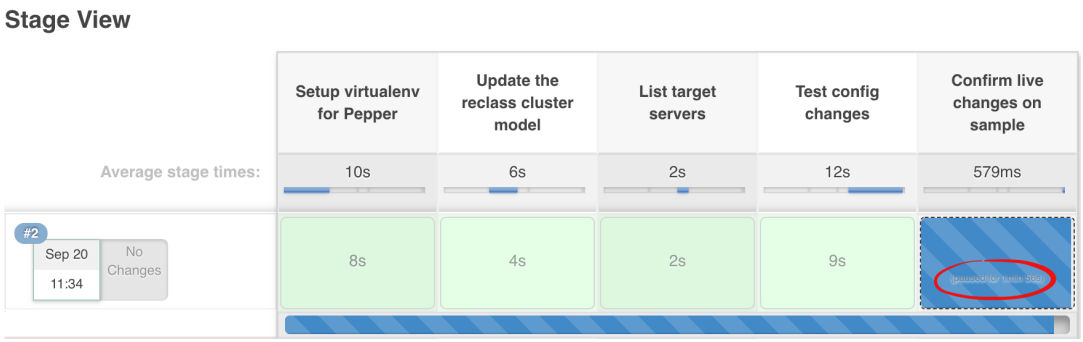
Step 10 Open the **Pipeline Deploy - update sevice(s) config** pipeline, click **Build with Parameters**, and fill out the following parameters. Leave the rest as default. Then click **BUILD**:

- PULL_MODEL: checked
- TARGET_SERVERS: ctl01*
- TARGET_STATES: nova.client



Step 11 Monitor the Progress in the *Status* tab; you will be asked to approve certain changes to the live cluster

Step 12 When the pipeline reaches *Confirm live changes on sample* step, you will need to check the sample environment:



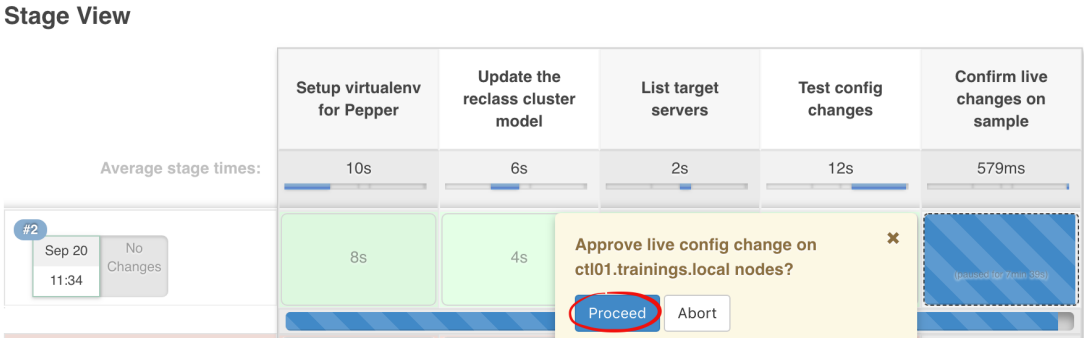
Before you click **Proceed** on the pop-up, let's do our due dillegence and check our **ctl01** node to see if the new flavor was created. (If you run a pipeline on multiple nodes, this step will tell you which node is the *sample* node to check your changes).

Step 13 Change focus to your SSH session to the lab, and issue the following command from the **cfg01** node:

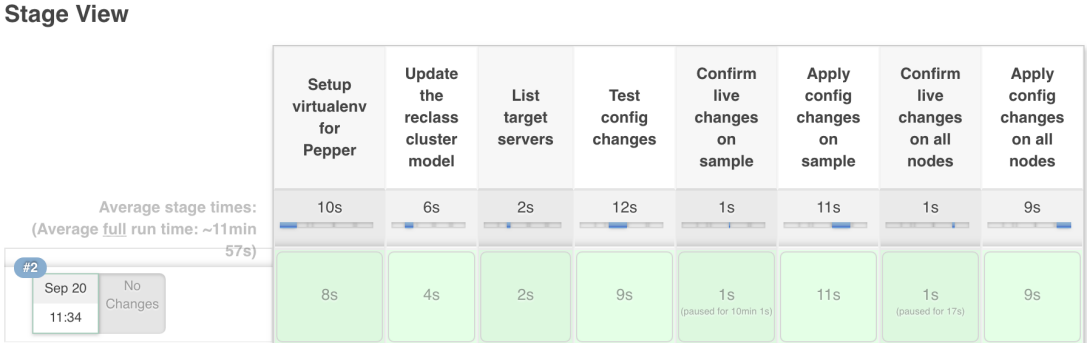
```
root@cfg01:~# salt 'ctl01*' cmd.run '. /root/keystonercv3; openstack flavor list'
ctl01.trainings.local:
+--------------------+--------+------+------+-----------+-------+-----------+
| ID                 | Name   |  RAM | Disk | Ephemeral | VCPUs | Is Public |
+--------------------+--------+------+------+-----------+-------+-----------+
| 36ba2dc9-7dc8-4b23 | medium | 1024 |    2 |         0 |     1 | True      |
| 6d6f1103-1b99-422e | small  |  512 |    1 |         0 |     1 | True      |
| 945322c2-dfa2-4d14 | large  | 2048 |    5 |         0 |     2 | True      |
+--------------------+--------+------+------+-----------+-------+-----------+
```

Our large flavor was created!

Step 14 Navigate back to your Jenkins UI and approve the change by clicking **Proceed**



Step 15 Since we only applied the change to one of the control nodes, *Confirm live changes on all nodes* also indicates **ctl01**. Click **Proceed** again to complete the pipeline.



Notes:

In OpenStack, you only need to speak with one of the control nodes via API calls, and OpenStack will create the resource (in this case Flavor) which is persisted in its database. If you went to ctl02 to ask for flavors, you will receive the same view as if you went to ctl01 (provided you are using the same OpenStack credentials).

9.4. Chapter Summary and Useful Notes

In this chapter, you have learned how to setup a new gerrit / git repository by adding its definition in the model. Then, we dissected the groovy script used by the update services config pipeline. By doing so, we concluded that the git repository was pointing upstream, which we changed by applying changes to the config.yml in the cluster model. Lastly, we added a new flavor to OpenStack using the model and pushed our changes to the local repository - after which we ran the pipeline to apply the changes to our infrastructure.

Notes

- **System / Cluster Model Location**: `/srv/salt/reclass/classes`
- **Salt Formula Location**: `/srv/salt/env/prd/`
- **Retrieve Pillars from all nodes**: *salt '*' pillar.items*
- **Retrieve a Specific Pillar from a specific node**: *salt 'ctl01*' pillar.items nova:client*
- **Look for a string in all reclass model**: *cd /srv/salt/reclass ; grep -r "string-to-search"*

Congrautlations, you have finished the *9. DriveTrain Operations* chapter!