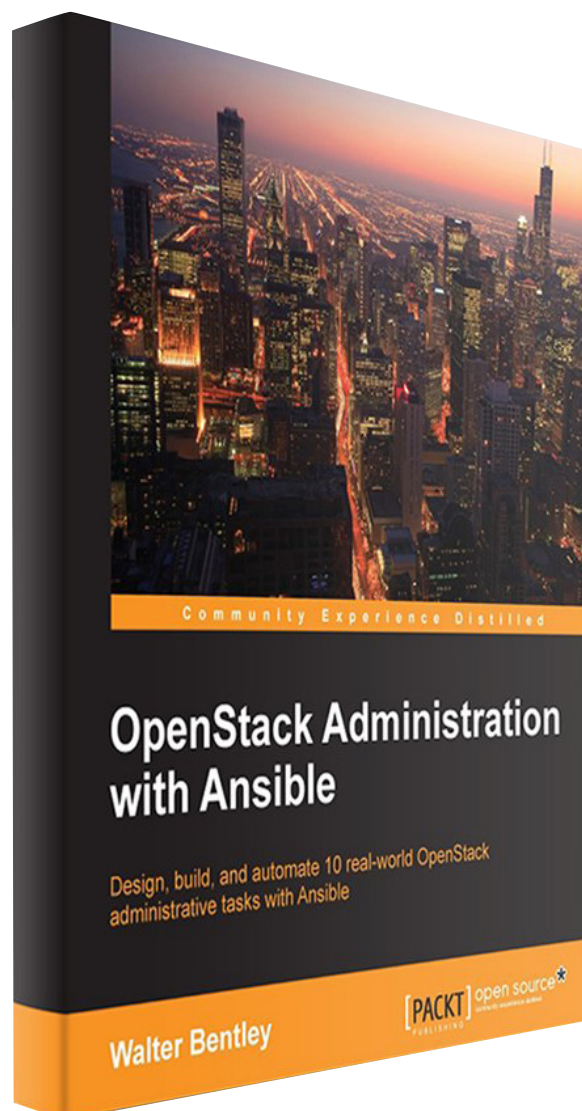


Migrating Instances

A free chapter sample from
OpenStack Administration with Ansible



For more information visit:

[www.packtpub.com/virtualization-and-cloud/
openstack-administration-ansible](http://www.packtpub.com/virtualization-and-cloud/openstack-administration-ansible)

[PACKT]
PUBLISHING

6

Migrating Instances

In this chapter, we will cover the task of migrating instances using the native OpenStack capability built into the Compute service (Nova). As mentioned earlier, the existence of this functionality is unknown by many. Before this chapter is over, we will prove this capability by demonstrating how to manually migrate instances; as well as, review the steps required to automate this task and finally create a playbook with roles to fully automate instance migration to a specified compute node.

In this chapter, we will cover the following topics,

- Instance migration
- Automation considerations
- Coding the playbook and roles
- Playbook and role review

Instance migration

Whenever the topic of instance migration comes up, it normally ends in a spirited conversation among my OpenStack peers for various reasons. So as a responsible adult, I will go on record and say instance migration is not perfect.

It has its flaws and can be quirky at best. Migration, whether live or not, has a practical use case in your OpenStack cloud. Within OpenStack, you have the capability of migrating instances from one compute node to another. You may do this for maintenance purposes and/or to rebalance the resource utilization across the cloud. Also, keep in mind that there are multiple ways to clear out a compute node for maintenance and we will cover this in more detail in *Chapter 8, Deploying OpenStack Features*.



As mentioned earlier, the OpenStack Compute service (Nova) has the functionality to migrate instances in a traditional method and the ability to live-migrate an instance as well.

We will first examine the traditional migration method and its properties.

The traditional migration method moves an instance by shutting down the instance, copying the instance image or file to the next available compute node, starting the instance on the new node, and lastly removing the instance from the original node. The areas to focus on in this method are:

- The instance is shutdown
- The instance image or file will take time to copy to a new compute node
- The new compute node selection is done by Nova Scheduler, you can not assign one without the additional steps required
- The instance is then brought back online once the copy is completed

Note that some may consider this method to be intrusive. The idea of shutting down an instance to move it was not a desirable scenario back in the virtualization days. Remember that we are in a new era, *the era of cloud and disposable resources*.

Since resources are readily available and you have the control to determine how to consume these resources, there should be no issue in taking an instance offline. Right? Yes, I know it will take a while to shake that *pet* mentality, you will get there. In the event the circumstances allow for this, which normally means you did a good job distributing across your hypervisors the instances running your application(s), you can very easily use this method to migrate instances.

A working example of the traditional instance migration command via the Nova CLI will be as follows:

```
$ nova migrate <instance>
$ nova migrate testinst
```


The other migration method would be to perform live instance migration. This method will remove the requirement of shutting down the instance, as it was highlighted in the traditional migration process described above. Instead of shutting down the instance, it is suspended (still in a running state) till it is reassigned to a new compute node. There are additional system requirements that are needed in order to leverage the live-migration functionality. The requirements are as follows:

- Some sort of shared or external storage capability must exist between your compute nodes

- With live-migration, you can select the new compute node but you must assure that the new node has the resources required for the new instance
- The old and new compute nodes must have the same CPU (OpenStack releases before Kilo may encounter an issue if this is not the case)

The first requirement is the most important one on the list and deserves some further explanation. The additional storage requirement can be covered in three different ways:


- The first way to satisfy the demand is to configure your hypervisors to store and have access to the shared storage; for instance, placement. It means that the instances are stored on the shared storage device and not on the ephemeral storage. This could involve mounting NFS share on the compute node to be used to store instances or through fiber channel sharing a LUN across the compute nodes, for example.
- The second approach to satisfy the shared or external storage requirement can be to leverage direct block storage, where your instances are backed by image-based root disks.
- The third and final approach could be the boot from volume storage capability. This is where you are booting instances off of Cinder-based volumes. Of course, you will need the block storage service (Cinder) enabled and configured within your OpenStack cloud.

 The key message, in relation to utilizing the live-migration capability, within Nova is that your *instances must exist on some sort of shared or external storage and cannot use ephemeral storage local to the compute node.*

A working example of an instance live-migration command via the Nova CLI would be as follows:

```
$ nova live-migration <instance><new compute node>
$ nova live-migration testinst compute01
```

As mentioned earlier, the whole concept of instance migration can range from being very simple all the way to being extremely complex. We hope that you can now clearly understand what is required and the process followed during an instance migration. Let's now examine the process of manually migrating an instance using the CLI.

 For simplicity purposes, we will demonstrate the manual commands using the OpenStack CLI only.

Manually migrating instances

The Compute service (Nova) is responsible for managing the instance migration process. Nova behind the scenes will execute all the steps needed to reassign the instance(s) to the new node and the movement of the instance image or file. Similar to every OpenStack service, you must authenticate against Keystone either by sourcing the OpenRC file discussed in *Chapter 1, An Introduction to OpenStack*, or by passing authentication parameters in-line with the command. The two tasks, individually, require different parameter values to be provided in order to successfully execute the command. See the following example:

For instance migration using OpenRC file, use the following commands:

```
$ source openrc
$ nova migrate <instance>
```

For instance backup passing the authentication parameters inline:

```
$ nova --os-username=<OS_USERNAME> --os-password=<OS_PASSWORD> --os-tenant-name=<OS_TENANT_NAME> --os-auth-url=<OS_AUTH_URL> migrate <instance>
```

With the `nova migrate` command you can add an optional additional argument to the command to report the instance migration process. The `--poll` argument can be used with various other Nova commands as well. It is something I will not use regularly when automating OpenStack tasks, for obvious reasons. Since the migration process can take some time and we are executing the task manually, it helps to keep track of its progress. An example of adding that optional argument would be:

```
$ nova migrate --poll <instance>
```

Since the traditional `nova migrate` command without the `--poll` argument does not output anything on the screen, you will need to execute a subsequent command to check on the migration status. The follow-up command will be the `nova migration-list` command.

A real life working example with an OpenRC file will look something similar to this:

```
$ source openrc
$ nova list
$ nova migrate test-1ae02fae-93ca-4485-a797-e7f781a7a25b
```

The output of the `nova migration-list` command will appear similar to:

```

root@infra1_utility_container-c750da76:~# nova migration-list
+-----+-----+-----+-----+-----+-----+-----+
| Source Node | Dest Node | Source Compute | Dest Compute | Dest Host | Status | Instance UUID |
+-----+-----+-----+-----+-----+-----+-----+
| 021579-compute03 | 021579-compute02 | 021579-compute03 | 021579-compute02 | 172.29.236.11 | confirmed | 058507b2-8a5d-47e5-9bc6-690b71838f5e |
T14:50:18.000000 |
| 021579-compute02 | 021579-compute03 | 021579-compute02 | 021579-compute03 | 172.29.236.12 | confirmed | 058507b2-8a5d-47e5-9bc6-690b71838f5e |
T13:42:07.000000 |
| 021579-compute02 | 021579-compute01 | 021579-compute02 | 021579-compute01 | 172.29.236.10 | finished | 1ae02fae-93ca-4485-a797-e7f781a7a25b |
T13:44:57.000000 |
+-----+-----+-----+-----+-----+-----+-----+

```

The complete output provided in the preceding command will vary, based on any previous migrations executed. The key information to focus on is the `Status` of the migration for the instance you just attempted to migrate. The status will be reported as either *migrating* or *finished*. Once the status is updated to *finished*, you can confirm the migration of the instance. After migration the instance will be in a `VERIFY_RESIZE` state by default, whether or not if you actually resized it. If you issue a `nova show` command the output will appear similar to this:

```

root@infra1_utility_container-c750da76:~# nova show test-1ae02fae-93ca-4485-a797-e7f781a7a25b
+-----+-----+
| Property | Value |
+-----+-----+
| OS-DCF:diskConfig | AUTO |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-SRV-ATTR:host | 021579-compute01 |
| OS-EXT-SRV-ATTR:hypervisor_hostname | 021579-compute01 |
| OS-EXT-SRV-ATTR:instance_name | instance-00000002 |
| OS-EXT-SIS:power_state | 1 |
| OS-EXT-SIS:task_state | - |
| OS-EXT-SIS:vm_state | resized |
| OS-SRV-USG:launched_at | 2015-09-17T13:45:00.000000 |
| OS-SRV-USG:terminated_at | - |
| accessIPv4 | |
| accessIPv6 | |
| config_drive | |
| created | 2015-08-10T00:00:45Z |
| flavor | m1.tiny (1) |
| hostId | d6a2be3a4f7132a048a15447f7e7275e7736b4124dcea0dc5032d909 |
| id | 1ae02fae-93ca-4485-a797-e7f781a7a25b |
| image | cirros-0.3.3 (8681a786-ed10-403a-9f28-429d32f482ef) |
| key_name | - |
| metadata | {} |
| name | test-1ae02fae-93ca-4485-a797-e7f781a7a25b |
| os-extended-volumes:volumes_attached | [] |
| private-network network | 10.1.100.3 |
| progress | 0 |
| security_groups | default |
| status | VERIFY_RESIZE |
| tenant_id | 903127b0aae141f298ffacf2cf6394dc |
| updated | 2015-09-17T13:44:57Z |
| user_id | 8c1a0ba3f4cf49e3bdeb515492636e53 |
+-----+-----+

```

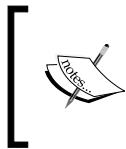
You will then need to execute the `nova resize-confirm` command to put the instance back in the `ACTIVE` state. The following example demonstrates this task:

```
$ nova resize-confirm test-1ae02fae-93ca-4485-a797-e7f781a7a25b
```

At this point you are good to go! Your instance will get migrated to a new compute node and run in an active state. For those of us who have learned to accept the traditional migration process, the next statement normally is, "*Why can't I migrate an instance to a specific compute node, using the nova migrate command?*". We will talk about this in the next section.

Migrating an instance to a specific compute node

The honest and straight answer to the above question is, I have no clue why this capability was not included. The good thing, just like most things within OpenStack, is that there is always a way to get it to do what you want.



Note that the steps outlined in the next section are 100% a workaround (a mid-grade dirty workaround) and *should not* be used within a production environment, without first executing multiple levels of testing to ensure the expected functionality.

As covered in the preceding sections, you cannot migrate an instance to a specific compute node using the traditional migration method. The option just does not exist (hope that changes soon). However, you can trick the Nova scheduler to place the instance on a selected compute node by disabling the other compute nodes. The Nova scheduler will then have no choice but to migrate the instance to the compute node you selected. Yes, in your mind you just called me an idiot; don't worry, it is not as intrusive as it sounds on paper.

The OpenStack control plane services are designed to report the status of the distributed components, such as compute nodes and/or cinder nodes; for example, the report received is stored within the OpenStack database and this is how the control plane services know if a particular node is up or down. Similarly, the control plane services can also force a report of a nodes status.

The Compute service (Nova) is an example service that can force a report on the status of a compute node. This will simply mark a compute node as up or down within the database and never actually *do* anything physically to the compute node. All the instances running on those compute nodes will remain running and the overall functionality of the node will go unchanged. However, the time for which the node is disabled within the database will prevent new instances to be created there. If you have a very busy and continuously changing OpenStack cloud and are not using a segregated set of compute nodes, this work-around is probably not a wise idea.

Due to its intrusive nature, it feels like a perfect administrative task to try and automate. With something like this, timing and accuracy is very critical. Wasting something as small as a minute could equate to the failure of being able to create any number of new instances by the cloud consumers inside your OpenStack cloud. For tasks of this nature, automation is king. In the next few sections, we will review the required steps to automate this task.

Automation considerations

This task also did not require any new framework decisions. All the other automation decisions we reviewed previously and carried over.

Before we start, it is worth noting that when automating a task, such as this one, (migrating an instance and disabling compute nodes) it is best to collect the details concerning them both before and after the migration. Having those details will simplify the process of reversing your changes, if required. Yes, this will add additional tasks to your role making it slightly more complex but still well worth it.

With that said, we are now ready to proceed to create our next playbook and role.

Coding the playbook and roles

In this section, we will now create the playbook and role that will allow you to migrate an instance to a specific compute node using the traditional `nova migrate` command. Unlike the other tasks that we have created so far, there is really only one way to handle this task. We will take the steps outlined in the preceding two sections, automate them to make sure that we have to supply only a few variable values, and then execute only one command.

This chapter started off with talking about instance migration and how there are two options within Nova to handle this; namely, traditional migration and live-migration. The traditional migration process is basically a one step process but in order to properly automate this task we will need to add a few more steps to the process. A brief outline of the tasks that we will have to create are:

1. List the compute nodes.
2. Collect pre-migration instance details.
3. Disable all compute nodes except for the one we want the instance to migrate to.
4. Migrate the instance.

5. Enable all compute nodes.
6. Confirm instance migration.
7. Collect post-migration instance details.

Since we are only creating a role in this example, we can start with the `main.yml` file within the role directory named `instance-migrate/tasks`. The initial contents of this file will look similar to the following code:

```
---  
  
- name: Retrieve hypervisor list  
  shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}  
  hypervisor-list | awk 'NR > 3' | awk '$4 != "{{ desthype }}" { print $4 }'  
  register: hypelist
```

The first step to retrieving the complete list of compute nodes within your OpenStack cloud is very easy if we use the `nova hypervisor-list` command. Once you get these results, it is best to strip down the output to provide just the information you need. Again, we will do this using the `awk` command and pipe (`|`) symbol. You will note that this is similar to how we did it in the last chapter. Remember that the `shell` module is used here because we are executing commands that require shell-specific operations.

For this particular task, we have to get a bit magical with the `awk` commands:

```
awk 'NR > 3' | awk '$4 != "{{ desthype }}" { print $4 }'
```

Not only will it pull off the first three lines of the standard CLI output, it will also check the fourth column and print the complete output excluding the value within the `{{ desthype }}` variable being passed. The consolidated output will then be registered into a variable named `hypelist`.

The next task will now collect the pre-migration instance details that will be stored for later use within the role. The code to accomplish this looks similar to the following code:

```
- name: Collect pre-migration instance details  
  shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}  
  list --name {{ instance }} --fields OS-EXT-SRV-ATTR:host | awk 'NR > 3' | awk '{ print $4 }'  
  register: preinststat
```

For this task, we are again using the Nova CLI to provide the instance details using the `nova list` command. You could have, just as well, used the `nova show` command to list the instance details. A distinct difference between the two commands is that with the `nova list` command you can choose which fields to return for the output. To do this, add the optional argument of `--fields` and a comma-delimited list of instance related field names to the command.

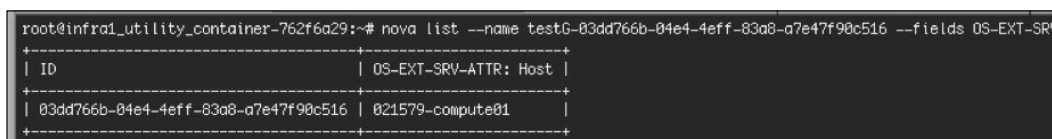
The following example will return only the instance ID, name, and status:

```
nova list --fields name,status
```

In our particular case, we want to know the compute node that the particular instance is currently running on. Thus our command will look similar to:

```
nova list --name {{ instance }} --fields OS-EXT-SRV-ATTR:host
```

The output will look similar to the following screenshot:



```
root@infra1_utility_container-762f6a29:~# nova list --name test6-03dd766b-04e4-4eff-83a8-a7e47f90c516 --fields OS-EXT-SRV-ATTR:host
+-----+-----+
| ID | OS-EXT-SRV-ATTR: Host |
+-----+-----+
| 03dd766b-04e4-4eff-83a8-a7e47f90c516 | 021579-compute01 |
+-----+-----+
```

The third task will be to disable the compute node(s) that you do not want the instance to migrate to; remember that we are only disabling the compute nodes within Nova and not physically changing the state of the compute node(s).

The code to do this will look similar to the following example:

```
- name: Disable unselected hypervisors
command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
service-disable {{ item }} nova-compute --reason '{{ migreason }}'
with_items: hypelist.stdout_lines
```

With use of the `nova service-disable` command, you can tell Nova to disable any particular Nova related service on remote hosts. In order to have Nova scheduler ignore or skip a compute node, you need to disable the `nova-compute` service. The command also requires a reason to be provided, of which will be stored in the Nova database for later reference if required. It is in this task where we will use the list of compute node(s) stored in the `hypelist` variable collected earlier.



Note that we will not disable the compute node that we want the instance to be migrated to, as we have filtered it out of the list already.

Moving onto the fourth task, we will now execute the instance migration. At this point, only the compute node you specify is enabled and nothing special needs to be done in reference to the `nova migrate` command except adding the `--poll` argument, so we can pause the role execution until the migration completes. See the following supporting code:

```
- name: Migrate instance
command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
migrate --poll {{ instance }}
```

Once the migration is completed, we need to immediately enable the compute node(s) that were disabled. One of the things I appreciate about OpenStack is, if you are given a command to disable something, you are normally given a command to re-enable it. So, we will simply execute the `nova service-enable` command and we will use the `hypelist` variable to provide the list of compute node(s) to execute against. The following code is used:

```
- name: Enable the disabled hypervisors
command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
service-enable {{ item }} nova-compute
with_items: hypelist.stdout_lines
```

Now that the migration is complete and the compute node(s) is enabled, we can focus on completing the instance migration process. The last step in an instance migration process is to notify Nova that you acknowledge that the instance was moved. At first glance, I could live without this step, but in hindsight some sort of confirmation does make overall sense. The code for this task is as follows:

```
- name: Confirm instance migration
command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
resize-confirm {{ instance }}
```

The last two final tasks will be used to provide the individual running the playbook with a visual confirmation of what was done. You can consider this more of an automation fail safe and less of a requirement. With an administrative task as complex as this, it is always a good common practice to output some details of what was changed on your system:

```
- name: Collect post-migration instance details
shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_
PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{
OS_AUTH_URL }}
list --name {{ instance }} --fields OS-EXT-SRV-ATTR:host,status | awk
'NR > 3' | awk '{ print $4 " and has a status of " $6 }' | awk 'NR ==
1'
register: postinststat

- name: Show instance location and status
debug: msg="{{ instance }}" was migrated from {{ item.0 }} to {{ item.1
}}}"
with_together:
- preinststat.stdout_lines
- postinststat.stdout_lines
```

These two tasks will first collect post-migration instance details and then use the information collected from the `preinststat` and `postinststat` variables to output to the screen a synopsis of the changes. The synopsis template used will be:

```
<instance migrated> was migrated from <compute node> to <compute node>
and has a status of <instance current status>
```



Feel free to go in and change it to fit your needs. This is just my opinionated approach. It felt right to keep it simple, while still supplying the pertinent details we care about when handling a migration. Upon the review of the playbook recap, if something went wrong and/or was implemented incorrectly, you should be able to quickly target steps for remediation.

Congratulations again; you have just completed your fourth OpenStack administration role. To support this role, we now need to create the variable file that will go along with it – the variable file named `main.yml`, which will be located in the `instance-migrate/vars` directory.



Keep in mind that the values defined in the variable file are intended to be changed before each execution, for normal everyday use.

For this role, we kept it pretty simple on the variables front and only needed to define three variables:

```
---
desthype: 021579-compute02
instance: testG-2c00131c-c2c7-4eae-aa90-981e54ca7b04
migreason: "Migrating instance to new compute node"
```

Let's take a moment to break down each variable. The summary will be:

```
desthype # this value would be the name of the compute node you wish
to migrate the instance to

instance # the name of the instance to be migrated

migreason: # a string encapsulated in quotes to explain the reason for
migrating the instance (keep the string brief)
```

With the variable file completed, we can now move on to creating the master playbook file. The file will be named `migrate.yml` and saved to the root of the playbook directory.



The playbook and role names can be anything you choose. The specific names have been provided here in order to allow for you to easily follow along and reference the completed code found in the GitHub repository. The only warning is, whatever you decide to name the roles must remain uniform when referenced from within the playbook(s).

The contents of the `migrate.yml` file will be:

```
---
# This playbook used to migrate instance to specific compute node.

- hosts: util_container
  user: root
  remote_user: root
  sudo: yes
  roles:
  - instance-migrate
```

The summary of this file is as follows:

```
hosts          # the host or host group to execute the playbook against

user           # the user to use when executing the playbook locally

remote_user   # the user to use when executing the playbook on the
remote host(s)

sudo           # will tell Ansible to sudo into the above user on the
remote host(s)

roles         # provide a list of roles to execute as part of this
playbook
```

Adding content to our host inventory file and the global variable file was done two chapters ago, so we already have that part covered. The values defined earlier will remain the same. The following is a quick recap of how those files are configured.

The `hosts` file in the root of the playbook directory:

```
[localhost]
localhostansible_connection=local

[util_container]
172.29.236.199
```

The global variable file inside the `group_vars/` directory:

```
# Here are variables related globally to the util_container host group

OS_USERNAME: ansible
OS_PASSWORD: passwd
OS_TENANT_NAME: admin
OS_AUTH_URL: http://172.29.236.7:35357/v2.0
```



A word of caution

Due to the contents of this file, it should be stored as a secure file within whatever code repository you may want to use to store your Ansible playbooks/roles. Gaining access to this information can compromise your OpenStack cloud security.

We are moving along very smoothly now, smile, you did it! Hoping that by this point everything is becoming a bit clearer. Keeping with our tradition, we will finish up the chapter with a quick review of the playbook and the role just created.

Playbook and role review

Let's jump right into examining the role we created called `instance-migrate`. The completed role and file named `main.yml` located in the `instance-migrate/tasks` directory looks similar to the following example:

```
---

- name: Retrieve hypervisor list
  shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
  hypervisor-list | awk 'NR > 3' | awk '$4 != "{{ desthype }}" { print $4 }'
  register: hypelist

- name: Collect pre-migration instance details
  shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
  list --name {{ instance }} --fields OS-EXT-SRV-ATTR:host | awk 'NR > 3' | awk '{ print $4 }'
  register: preinststat

- name: Disable unselected hypervisors
  command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
  service-disable {{ item }} nova-compute --reason '{{ migreason }}'
  with_items: hypelist.stdout_lines

- name: Migrate instance
  command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
  migrate --poll {{ instance }}

- name: Enable the disabled hypervisors
  command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
  service-enable {{ item }} nova-compute --reason '{{ migreason }}'
  with_items: hypelist.stdout_lines
```

```

service-enable {{ item }} nova-compute
with_items: hypelist.stdout_lines

- name: Confirm instance migration
command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
resize-confirm {{ instance }}

- name: Collect post-migration instance details
shell: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-tenant-name={{ OS_TENANT_NAME }} --os-auth-url={{ OS_AUTH_URL }}
list --name {{ instance }} --fields OS-EXT-SRV-ATTR:host,status | awk 'NR > 3' | awk '{ print $4 " and has a status of " $6 }' | awk 'NR == 1'
register: postinststat

- name: Show instance location and status
debug: msg="{{ instance }} was migrated from {{ item.0 }} to {{ item.1 }}"
with_together:
- preinststat.stdout_lines
- postinststat.stdout_lines

```

The corresponding variable file named `main.yml`, located in the `instance-migrate/vars` directory, for this role will look similar to the following example:

```

---
desthype: 021579-compute02
instance: testG-2c00131c-c2c7-4eae-aa90-981e54ca7b04
migreason: "Migrating instance to new compute node"

```

Next, the master playbook file named `migrate.yml`, located in the root of the `playbook` directory, will look similar to the following example:

```

---
# This playbook used to migrate instance to specific compute node.

- hosts: util_container
user: root
remote_user: root
sudo: yes
roles:
- instance-migrate

```


Following that, we created the `hosts` file, which is also located in the root of the playbook directory:


```
[localhost]
localhostansible_connection=local

[util_container]
172.29.236.199
```

Finally, creating the global variable file named `util_container` and saving it to the `group_vars/` directory of the playbook will complete the playbook:

```
# Here are variables related globally to the util_container host group

OS_USERNAME: ansible
OS_PASSWORD: passwd
OS_TENANT_NAME: admin
OS_AUTH_URL: http://172.29.236.7:35357/v2.0
```

 The complete set of code can again be found in the following GitHub repository:
<https://github.com/os-admin-with-ansible/os-admin-with-ansible>

We have finally landed on my favorite part of creating Ansible playbooks and roles, which is to test out our great work. Fortunately, for you I have knocked out all the bugs already (wink wink). Assuming you have cloned the GitHub repository from the link provided in the note, the command to test the playbook from the Deployment node will be as follows:

```
$ cd os-admin-with-ansible
$ ansible-playbook -i hosts migrate.yml
```

A sample of the playbook execution output is as shown in the following screenshot:

```

PLAY [util_container] *****
GATHERING FACTS *****
ok: [172.29.236.85]

TASK: [instance-migrate | Retrieve hypervisor list] *****
changed: [172.29.236.85]

TASK: [instance-migrate | Collect pre-migration instance details] *****
changed: [172.29.236.85]

TASK: [instance-migrate | Disable unselected hypervisors] *****
changed: [172.29.236.85] => (item=021579-compute01)
changed: [172.29.236.85] => (item=021579-compute03)

TASK: [instance-migrate | Migrate instance] *****
changed: [172.29.236.85]

TASK: [instance-migrate | Enable the disabled hypervisors] *****
changed: [172.29.236.85] => (item=021579-compute01)
changed: [172.29.236.85] => (item=021579-compute03)

TASK: [instance-migrate | Confirm instance migration] *****
changed: [172.29.236.85]

TASK: [instance-migrate | Collect post-migration instance details] *****
changed: [172.29.236.85]

TASK: [instance-migrate | Show instance location and status] *****
ok: [172.29.236.85] => (item=[u'021579-compute03', u'021579-compute02 and has a status of ACTIVE']) => {
  "item": {
    "021579-compute03",
    "021579-compute02 and has a status of ACTIVE"
  },
  "msg": "testG-2c00131c-c2c7-4eae-aa90-981e54ca7b04 was migrated from 021579-compute03 to 021579-compute02 and has a status of ACTIVE"
}

PLAY RECAP *****
172.29.236.85 : ok=9  changed=7  unreachable=0  failed=0

```

Summary

It's nice to have completed yet another chapter covering real-life OpenStack administrative duties. The more you create playbooks and roles, the faster you will be able to create a new code just by simply reusing the code created earlier for other purposes. Before this book is over, you will have a nice collection of playbooks/roles to reference for future Ansible automation.

Taking a moment to recap this chapter, you will recall that we covered what an instance migration is and why you might want to use this functionality, we reviewed the two possible migration methods—traditional and live-migration—learned how to manually migrate an instance, and also a work-around on how to use traditional migration to migrate an instance to a specific compute node. Lastly, we created the Ansible playbook and role to automate that work-around approach.

The next chapter is near and dear to my heart, as it required me to spend hours perfecting it for a customer I was working with at the time. It even granted me a *Hands-on Labs* talk slot at the OpenStack Summit that was hosted in Vancouver. While it may not be a traditional OpenStack administrative task asked of you, it will help demonstrate the functional power OpenStack natively has around being able to strictly isolate tenants consuming your clouds resources. In the next chapter, we will cover the steps required to implement multi-isolation within your cloud and demonstrate why automating such a complex administrative task is extremely important. Grab another cup of coffee, have a quick stretch, and let's start with *Chapter 7, Setting up Isolated Tenants*.