Private Cloud Appliance

Automate Infrastructure Lifecycle Management on PCA using Ansible

ORACLE WHITE PAPER | SEPTEMBER 2018

Contents

Sample code is provided for educational purposes or to assist your development or administration efforts. Your use rights and restrictions for each sample code item are described in the applicable license agreement. Except as may be expressly stated in the applicable license agreement or product documentation, sample code is provided "as is" and is not supported by Oracle.

Introduction

Oracle Private Cloud Appliance is an Engineered System designed for rapid and turn-key deployment of private cloud at an industry-leading price point. The agile and intelligent infrastructure allows for scaling compute capacity on demand, zero downtime upgrades and supports your choice of external storage. Whether running Linux, Microsoft Windows or Oracle Solaris applications, Oracle Private Cloud Appliance supports a wide range of mixed workloads in medium-to-large sized data centers. High-performance, low-latency Oracle Fabric Interconnect and Oracle SDN allow automated configuration of the server and storage networks. The embedded controller software automates the installation, configuration, and management of all infrastructure components.

Automation is a key requirement for achieving cloud-like agility. Ansible is an IT automation engine that automates cloud provisioning, configuration management, application deployment and orchestration. Ansible uses no agents and can be used to automate repetitive IT tasks in multi-node deployments. A machine that has Ansible installed (‘Control Machine’) pushes code blocks (‘Ansible modules’) to the remote machines (‘Managed nodes’) and executes them over SSH. This paper describes the process to use the custom Ansible module ‘ovmm_vm.py’ to automate creation, deletion, halting and starting a Virtual Machine in Oracle PCA.

The module interfaces with the REST APIs for Oracle VM and hence, can be even used in an Oracle VM environment outside of a PCA.

Prerequisites

The versions of major software components used in this setup are:

- **Version of Oracle PCA software.** 2.3.1+ (The module works with Oracle VM 3.3+)
- **Version of Ansible.** 2.1.0.0 or newer
- Download and install the Ansible RPM from OTN. The files contained in the RPM are as follows:

```
[root@dhcp-10-211-54-119]# rpm -qpl pca_ansible_examples-1.0-
1.el7.noarch.rpm

/usr/lib/python2.7/site-packages/pca/plugins/ovmm_vm.py
/usr/lib/python2.7/site-packages/pca/plugins/ovmm_vm.pyc
/usr/lib/python2.7/site-packages/pca/plugins/ovmm_vm.pyo
/usr/share/doc/pca_ansible_examples-1.0
/usr/share/doc/pca_ansible_examples-1.0/COPYING
/usr/share/doc/pca_ansible_examples-1.0/Copyright
/usr/share/pca_ansible_examples/examples
/usr/share/pca_ansible_examples/examples/deletevm.yml
/usr/share/pca_ansible_examples/examples/play.yml
/usr/share/pca_ansible_examples/examples/startvm.yml
/usr/share/pca_ansible_examples/examples/stopvm.yml
```

Ansible checks for the custom module in the **/library subdirectory** of the directory where your playbook is stored. Thus, in the above directory structure, you can place the ovmm_vm.py file in directory `/usr/share/pca_ansible_examples/examples/library/`

Note: For the purpose of simplifying the directory paths in this paper, we will place the playbooks in the Ansible directory (/etc/ansible) and the module in the library subdirectory of the Ansible directory (here etc/ansible/library) on Control Machine.

Control Machine

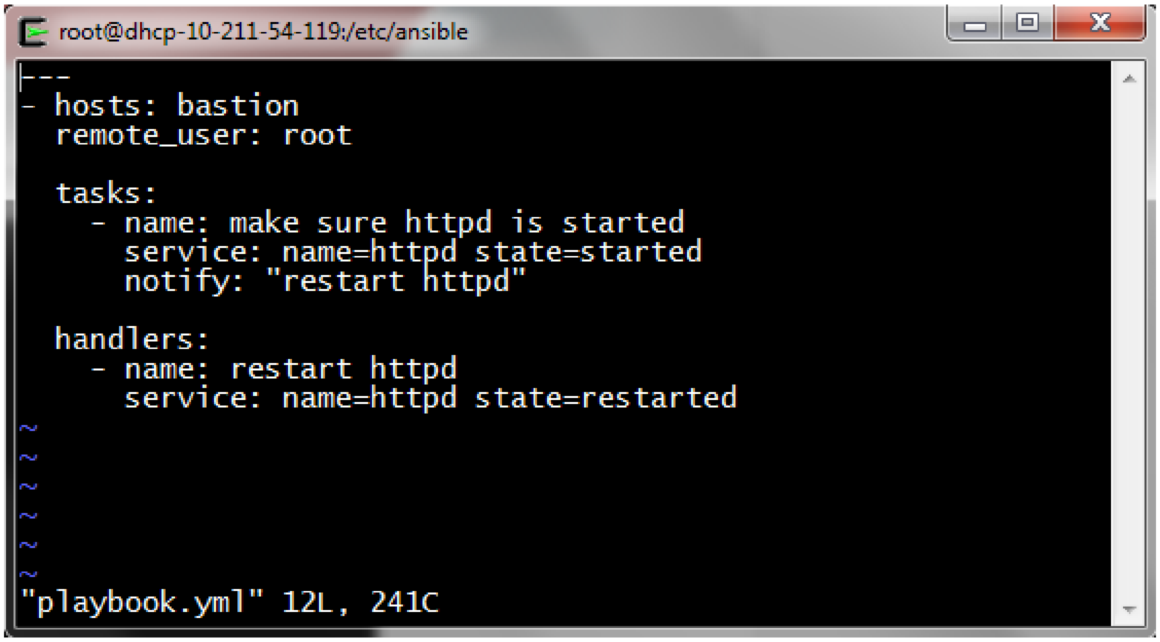
```
/etc/ansible
 └─► Play.yml , deletevm.yml, startvm.yml, stopvm.yml
 └─► Library
      └─► ovmm_vm.py
```

How Ansible Works

Ansible works by pushing code blocks ('modules') to remote hosts ('managed nodes'), executing them then removing them after the remote host is in the 'desired' state.

The automation jobs are described in YAML language. **Playbooks**, written in YAML, are used to manage configurations and deployment to remote systems.

Playbooks contain **plays** which map remote hosts to tasks. A **task** is nothing more than a call to a Module. **Modules** are task plugins that do the actual work on a remote host.



```
---
- hosts: bastion
  remote_user: root

  tasks:
    - name: make sure httpd is started
      service: name=httpd state=started
      notify: "restart httpd"

  handlers:
    - name: restart httpd
      service: name=httpd state=restarted
~
~
~
~
~
~
"playbook.yml" 12L, 241C
```

Figure 1: Sample Ansible Playbook

Figure 1 shows a sample playbook that has only one play to be executed on remote host ‘bastion’ as remote user root. The task is to make sure httpd service is started on bastion host. This task calls the ‘service’ module and declares the desired state as ‘started’ for service httpd.

Handlers are special tasks that are carried out if the preceding task was successful i.e. it made a change in state of the remote host. Thus, in this case the handler “restart httpd” would only be called if httpd service had to be started on bastion host by the task ‘make sure httpd is statrted’. In other words, if httpd was already running on bastion (i.e task ‘make sure httpd is running’ didn’t make any change to the remote host bastion), the handler would not have been executed.

Architecture

Figure 2 shows an Ansible Control Machine components and also the architecture of Oracle VM Manager located on the Management node of PCA. Ansible manages the PCA by connecting to the REST APIs for Oracle VM Manager (accessible by HTTPS over port 7002 as shown in figure). The detailed architecture for Oracle VM can be found here.

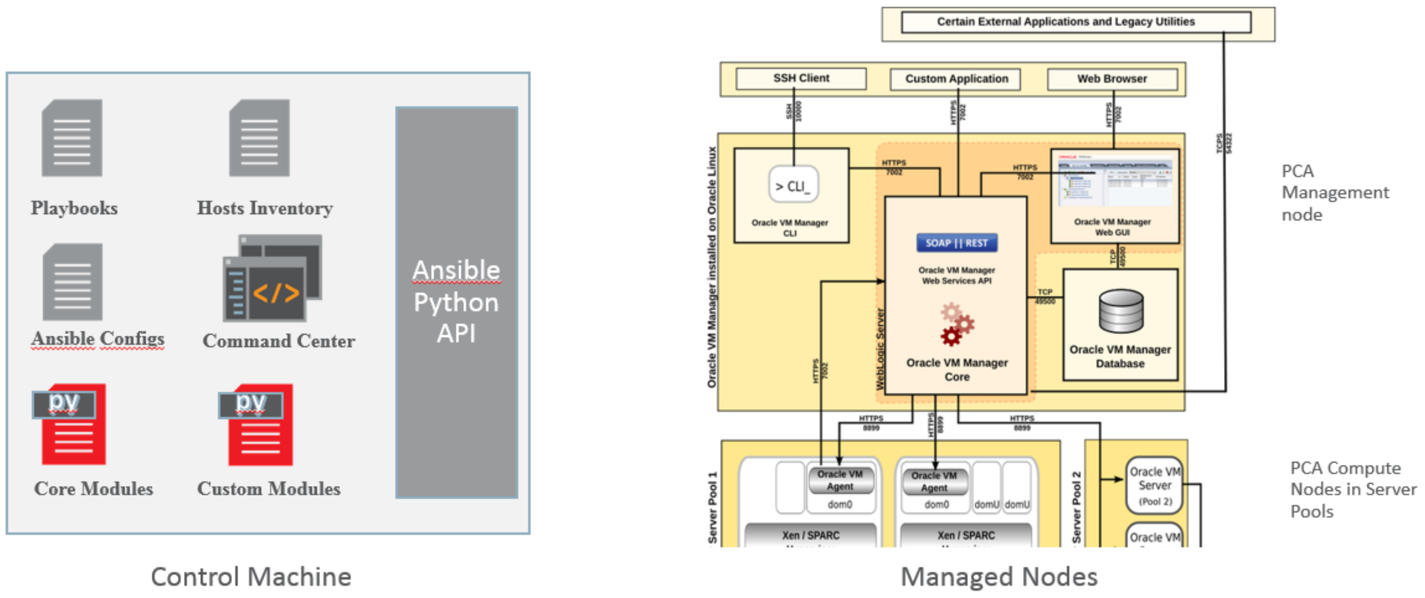
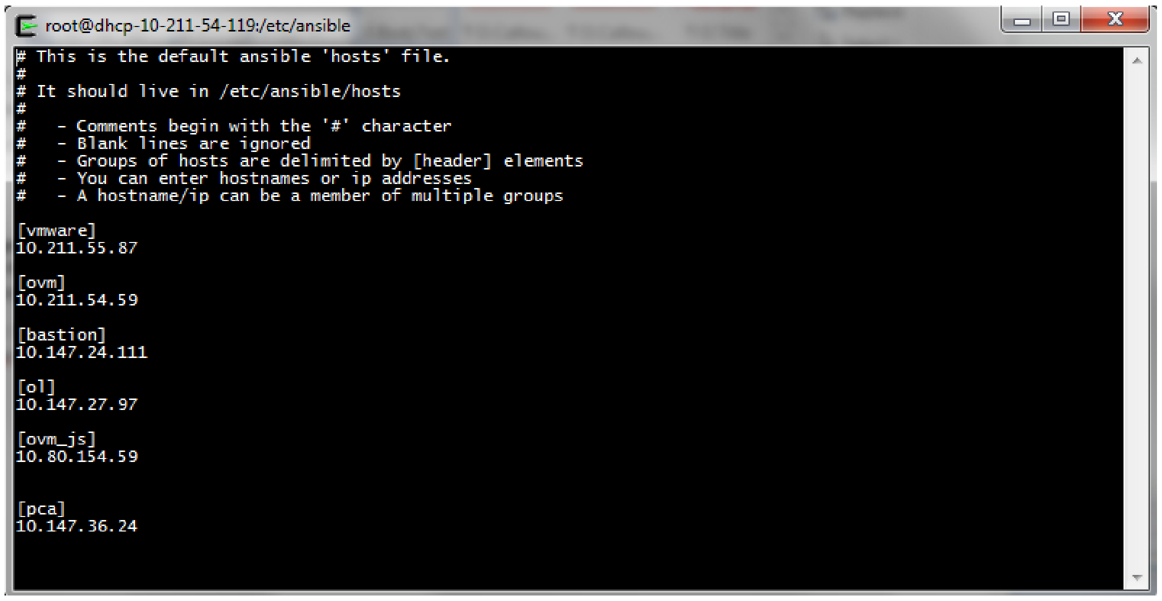


Figure 2. Architecture diagram for Ansible based deployments

This lab setup has the following components:

- **Control Machine**: An Oracle Linux 7 system external to the PCA on which Ansible is installed serves as the Control Machine. Ansible package is available via the `ol7_developer_EPEL` Channel.

- **Managed Nodes**: The remote systems that Ansible manages remotely. In our case, the managed nodes would be the PCA management nodes. The location of managed nodes (Host names or IP address) is defined in *Inventory File*. The default inventory file is `etc/ansible/hosts` on the control machine.



```
# This is the default ansible 'hosts' file.
#
# It should live in /etc/ansible/hosts
#
#   - Comments begin with the '#' character
#   - Blank lines are ignored
#   - Groups of hosts are delimited by [header] elements
#   - You can enter hostnames or ip addresses
#   - A hostname/ip can be a member of multiple groups

[vmware]
10.211.55.87

[ovm]
10.211.54.59

[bastion]
10.147.24.111

[ol]
10.147.27.97

[ovm_js]
10.80.154.59


[pca]
10.147.36.24
```

Figure 3. A sample inventory file located at /etc/ansible/hosts

Custom Module Arguments

The custom module ovmm_vm.py takes the following arguments as inputs from the user.

- **state=**dict(required=True, choices=['present', 'absent', 'start', 'stop']),
- **name**=dict(required=True),
- **description**=dict(required=False),
- **ovm_user**=dict(required=True),
- **ovm_pass**=dict(required=True),
- **ovm_host**=dict(required=True),
- **ovm_port**=dict(required=True),
- **server_pool**=dict(required=False),
- **repository**=dict(required=False),
- **vm_domain_type**=dict(default='XEN_HVM', choices=["XEN_HVM","XEN_HVM_PV_DRIVERS","XEN_PVM","LDOMS_PVM","UNKNOWN"]),
- **memory**=dict(required=False, default=4096, type='int'),
- **max_memory**=dict(required=False, default=None, type='int'),
- **vcpu_cores**=dict(required=False, default=2, type='int'),
- **max_vcpu_cores**=dict(required=False, default=None, type='int'),
- **operating_system**=dict(required=False),
- **networks**=dict(required=False, type='list'),
- **disks**=dict(required=False, type='list'),
- **boot_order**=dict(required=False, type='list')

Note: The arguments with required=True have to be supplied while making a call to this module, else the module execution fails and Ansible throws a message "*missing required arguments: <argument_name>*".

Note: The ovm_user and ovm_pass are required for basic HTTP Authentication. They are not required if you set up SSL Certificate verification for Oracle VM as discussed in Appendix I. In this case, (required=False) needs to be set for both ovm_user and ovm_pass in order to not supply

The 'ovmm_vm' module can be used for automating the following operations on a PCA.

» Creating a Virtual Machine
» Deleting a Virtual Machine
» Starting a Virtual Machine
» Stopping a Virtual Machine

Argument '**state**' can take 4 values: ***Present, absent, start, stop***. Each value represents the desired final state of the VM. The value of argument 'state' determines the desired action.

» **State=present** means that the specified VM should exist on the PCA after execution of the module. Thus, if it is already present, Ansible returns the ID of the existing VM without making any change to the remote system. If the specified VM doesn't exist, it is created with the specified configuration

» **State=absent** means the VM will be deleted if it exists on the PCA**.** This deletes the VM disk mapping, VM disk and VNICs along with the Virtual Machine.

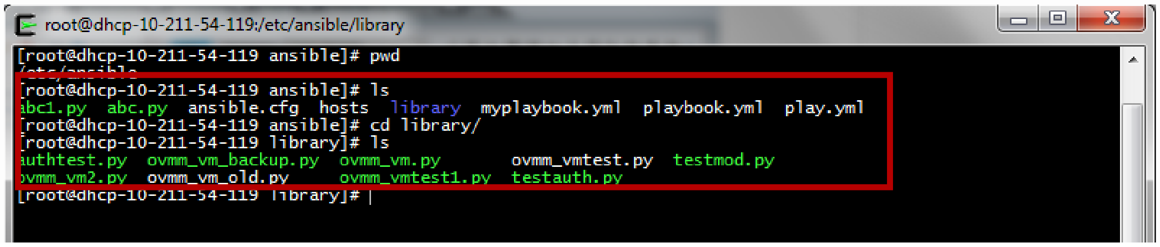
» **State=start** would start the specified VM, would do nothing if the VM doesn't exist or is already running

» **State=stop** will stop the specified VM if it is in ‘running’ state and do nothing otherwise

Executing the Module

The custom module and the Ansible playbook that uses the module need to be placed on the Ansible control machine. For custom modules, Ansible will look in the library directory relative to the playbook, for example: `playbooks/library/your-module.`

Figure 4 shows the relative location of playbook **play.yml** and Ansible module **ovmm_vm.py** on the Control Machine.

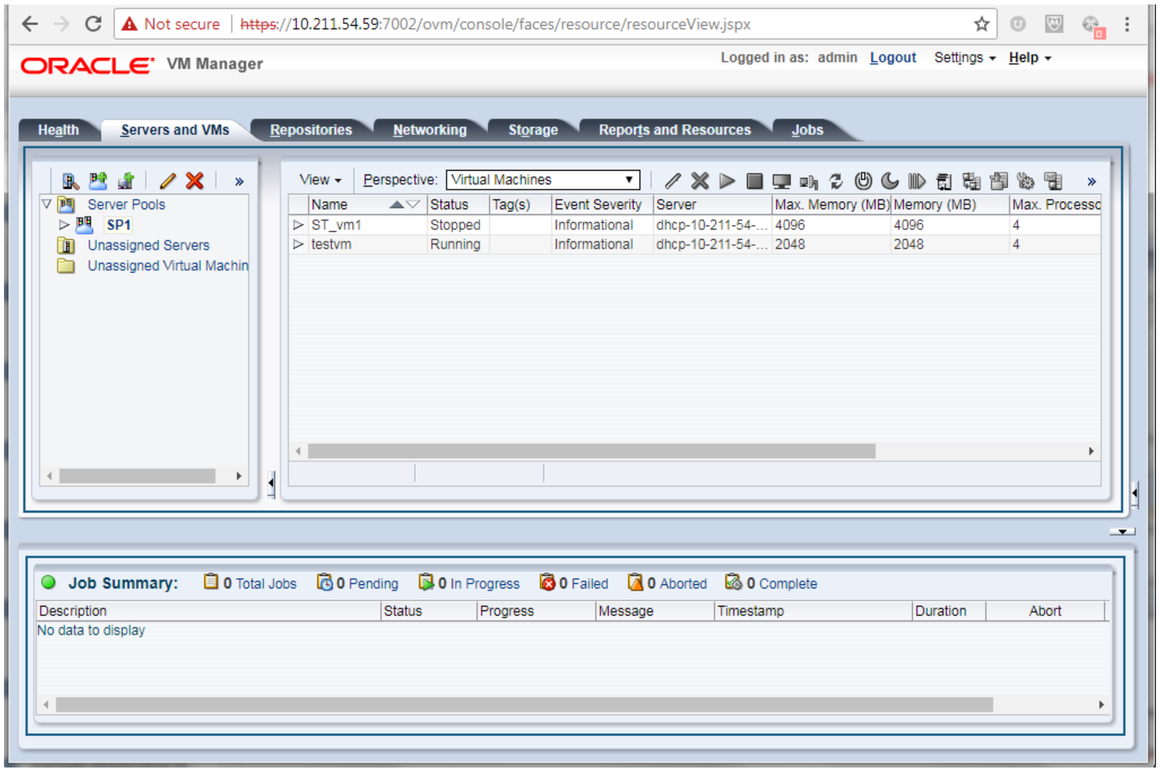


```
root@dhcp-10-211-54-119:/etc/ansible/library
[root@dhcp-10-211-54-119 ansible]# pwd
/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ls
abc1.py  abc.py  ansible.cfg  hosts  library  myplaybook.yml  playbook.yml  play.yml
[root@dhcp-10-211-54-119 ansible]# cd library/
[root@dhcp-10-211-54-119 library]# ls
authtest.py  ovmm_vm_backup.py  ovmm_vm.py        ovmm_vmtest.py  testmod.py
ovmm_vm2.py  ovmm_vm_old.py     ovmm_vmtest1.py  testauth.py
[root@dhcp-10-211-54-119 library]# |
```

Figure 4. Playbooks are located at /etc/ansible and custom modules are in /etc/ansible/library

Each of the tasks described in module ovmm_vm.py can be carried out by either making a call to the module from Ansible command line (**Ad-hoc command**) or using **Ansible Playbook**. Both methods are discussed in the following sections.

Figure 5 shows the Oracle VM Manager GUI for PCA. We have 2 VMs on Server Pool SP1, named ‘*ST_vm1*’ in Stopped state and ‘*testvm*’ in the Running state. Let’s automate the lifecycle management tasks that can be performed using the custom Ansible module ovmm_vm.py


Not secure
https://10.211.54.59:7002/ovm/console/faces/resource/resourceView.jspx
ORACLE VM Manager
Logged in as: admin Logout Settings Help
Health
Servers and VMs
Repositories
Networking
Storage
Reports and Resources
Jobs
Server Pools
SP1
Unassigned Servers
Unassigned Virtual Machin
View
Perspective: Virtual Machines

Name	Status	Tag(s)	Event Severity	Server	Max. Memory (MB)	Memory (MB)	Max. Processo
ST_vm1	Stopped		Informational	dhcp-10-211-54-...	4096	4096	4
testvm	Running		Informational	dhcp-10-211-54-...	2048	2048	4

Job Summary: 0 Total Jobs 0 Pending 0 In Progress 0 Failed 0 Aborted 0 Complete

Description	Status	Progress	Message	Timestamp	Duration	Abort
No data to display						

Figure 5. Oracle VM Manager GUI showing 2 VMs – ST_vm1 in stopped state and testvm in Running state

Ansible Ad-Hoc Command Method

An ad-hoc command is something that you might type in to do something really quick, but don't want to save for later. The syntax for ad-hoc commands is

$ ansible <hostname> -m <module_name> -a <arguments_to_module>

Playbook Execution

Playbooks are Ansible's configuration, deployment, and orchestration language. They can describe a policy you want your remote systems to enforce, or a set of steps in a general IT process. Playbooks are written in YAML format. The syntax for running a playbook is:

$ ansible-playbook –i<inventory file> <playbook_name.yml>

Note: Ansible uses the default inventory file located at /etc/ansible/hosts if flag –i is not specified while executing the playbook.

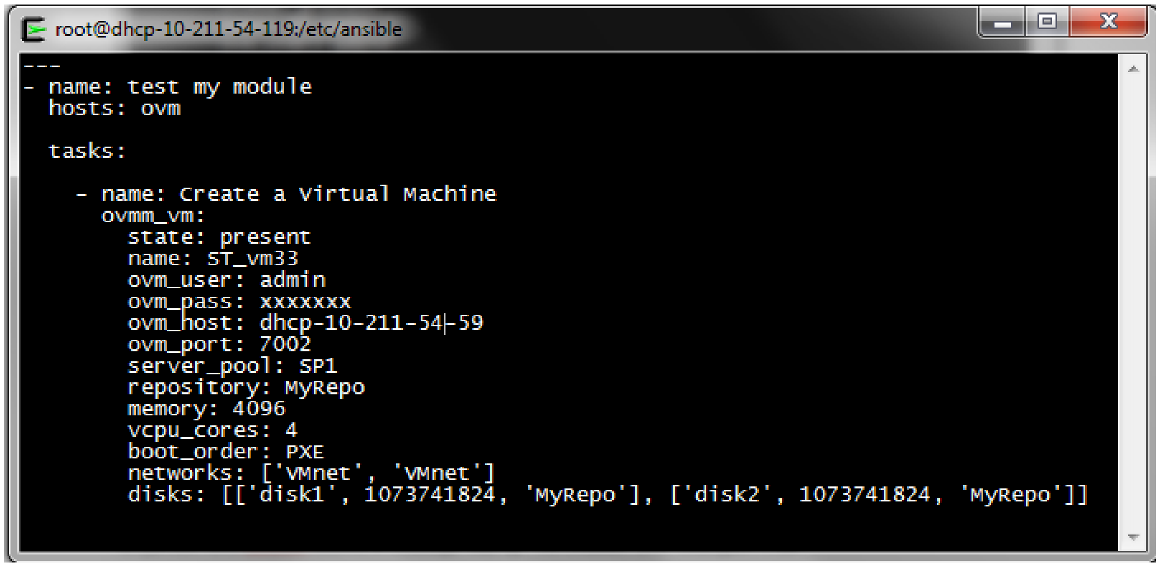
Examples of executing the module *ovmm_vm.py* using playbook are shown in the following section.

Create a VM

To create a VM with Virtual Disks and Virtual NICs, the following arguments need to be specified while executing the module ovmm_vm.py

- **state=**present
- **name**
- **ovm_user**
- **ovm_pass**
- **ovm_host**
- **ovm_port**
- **repository**
- **server_pool**
- **networks** – provide a list of all the networks corresponding to the Virtual NICs that the VM should have. For two VNics on same network, type the network name twice as shown in Figure 6.
- **disks –** provide a list of virtual disks in the form of a tuple *[Virtual Disks name, size in bytes, Repository]* as shown in Fig 6.

The playbook used for creating the VM is shown on Figure 6.



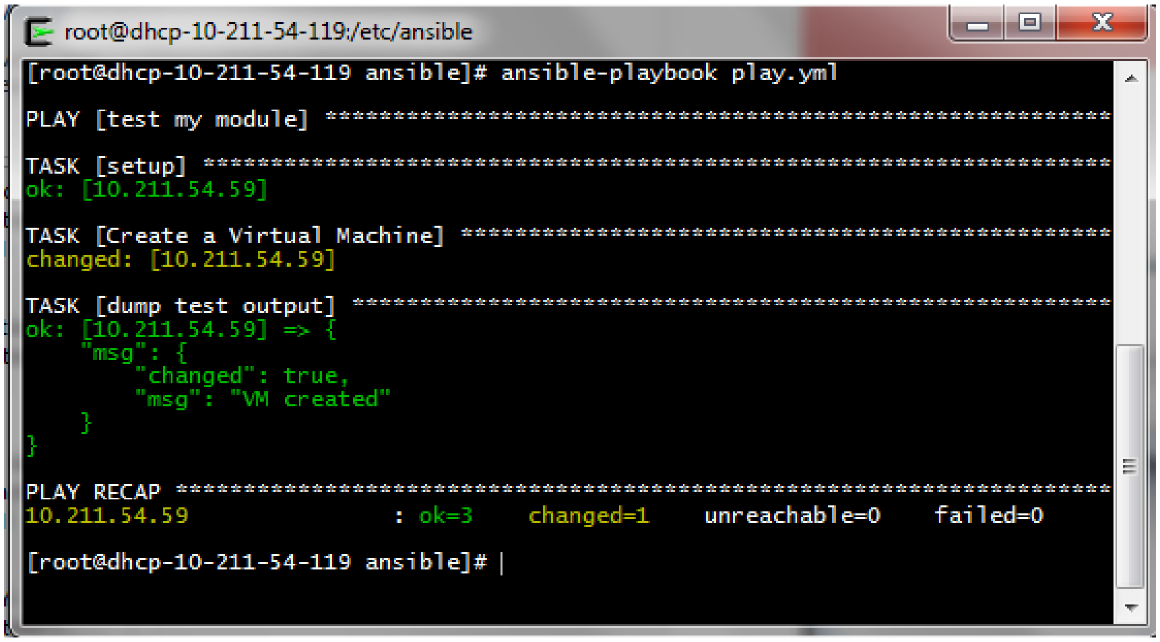
```
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Create a Virtual Machine
      ovmm_vm:
        state: present
        name: ST_vm33
        ovm_user: admin
        ovm_pass: xxxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002
        server_pool: SP1
        repository: MyRepo
        memory: 4096
        vcpu_cores: 4
        boot_order: PXE
        networks: ['VMnet', 'VMnet']
        disks: [['disk1', 1073741824, 'MyRepo'], ['disk2', 1073741824, 'MyRepo']]
```

Figure 6. Playbook play.yml showing the task ‘Create a VM’ on remote host ovm

The execution of the code for creating a VM is shown in Appendix I.

Case 1:
Specified VM doesn’t exist on PCA, Desired State: present

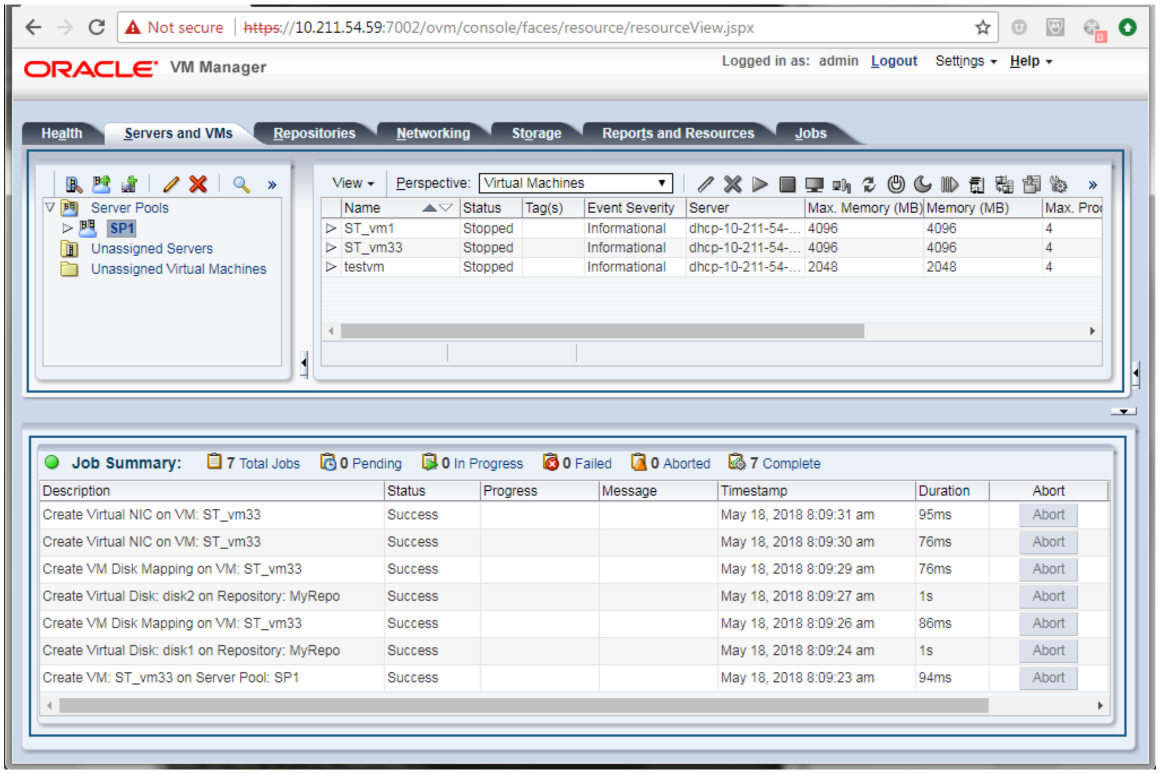
In this case, as the specified VM doesn’t exist on PCA, we expect the module to create it on the specified server pool and repository on PCA and display a changed=True message. Figure 7 shows this execution on the VM ‘ST_vm33’ and Ansible returns a message saying “VM created” and “changed=True”.



```
[root@dhcp-10-211-54-119 ansible]# ansible-playbook play.yml

PLAY [test my module] ***********************************************************

TASK [setup] ********************************************************************
ok: [10.211.54.59]

TASK [Create a Virtual Machine] *************************************************
changed: [10.211.54.59]

TASK [dump test output] *********************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": true,
        "msg": "VM created"
    }
}

PLAY RECAP **********************************************************************
10.211.54.59               : ok=3    changed=1    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

Figure 7. Execution of Playbook play.yml when the VM doesn't exist on the remote host

The fact that the VM 'ST_vm33' is created on the remote host can be seen in Figure .


Not secure | https://10.211.54.59:7002/ovm/console/faces/resource/resourceView.jspx
ORACLE VM Manager
Logged in as: admin Logout Settings Help
Health
Servers and VMs
Repositories
Networking
Storage
Reports and Resources
Jobs
Server Pools
SP1
Unassigned Servers
Unassigned Virtual Machines
View
Perspective: Virtual Machines

Name	Status	Tag(s)	Event Severity	Server	Max. Memory (MB)	Memory (MB)	Max. Pro
ST_vm1	Stopped		Informational	dhcp-10-211-54-...	4096	4096	4
ST_vm33	Stopped		Informational	dhcp-10-211-54-...	4096	4096	4
testvm	Stopped		Informational	dhcp-10-211-54-...	2048	2048	4

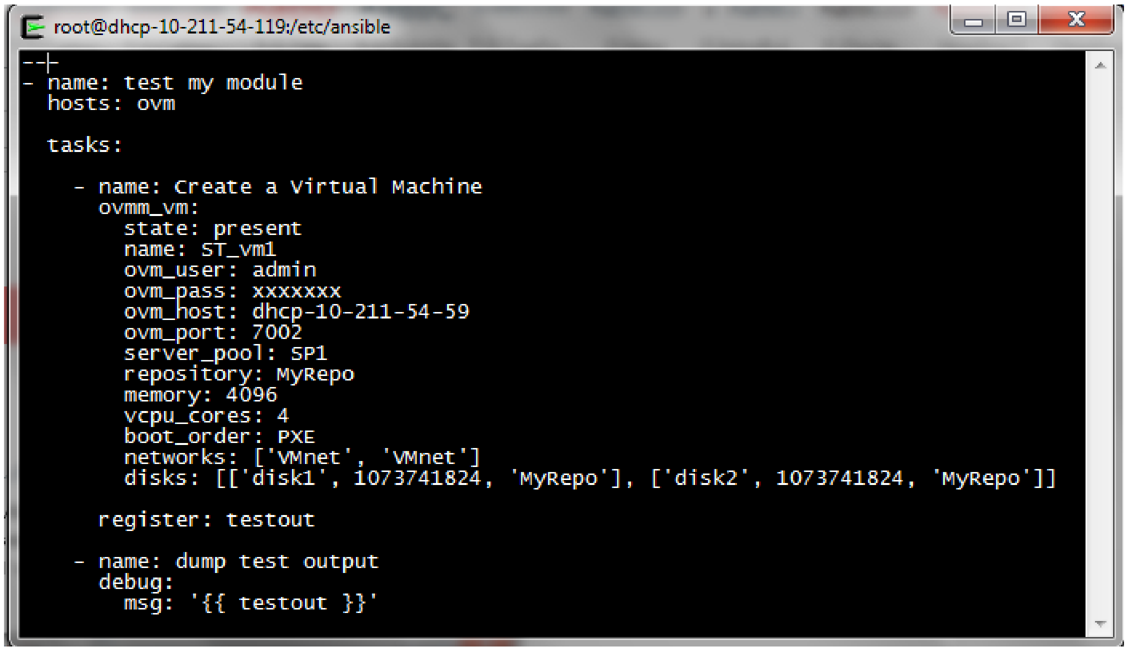
Job Summary: 7 Total Jobs 0 Pending 0 In Progress 0 Failed 0 Aborted 7 Complete

Description	Status	Progress	Message	Timestamp	Duration	Abort
Create Virtual NIC on VM: ST_vm33	Success			May 18, 2018 8:09:31 am	95ms	Abort
Create Virtual NIC on VM: ST_vm33	Success			May 18, 2018 8:09:30 am	76ms	Abort
Create VM Disk Mapping on VM: ST_vm33	Success			May 18, 2018 8:09:29 am	76ms	Abort
Create Virtual Disk: disk2 on Repository: MyRepo	Success			May 18, 2018 8:09:27 am	1s	Abort
Create VM Disk Mapping on VM: ST_vm33	Success			May 18, 2018 8:09:26 am	86ms	Abort
Create Virtual Disk: disk1 on Repository: MyRepo	Success			May 18, 2018 8:09:24 am	1s	Abort
Create VM: ST_vm33 on Server Pool: SP1	Success			May 18, 2018 8:09:23 am	94ms	Abort

Figure 8. The VM 'ST_vm33' is successfully created in the Oracle VM Manager GUI

Case 2:
Specified VM exists on PCA, Desired State: present

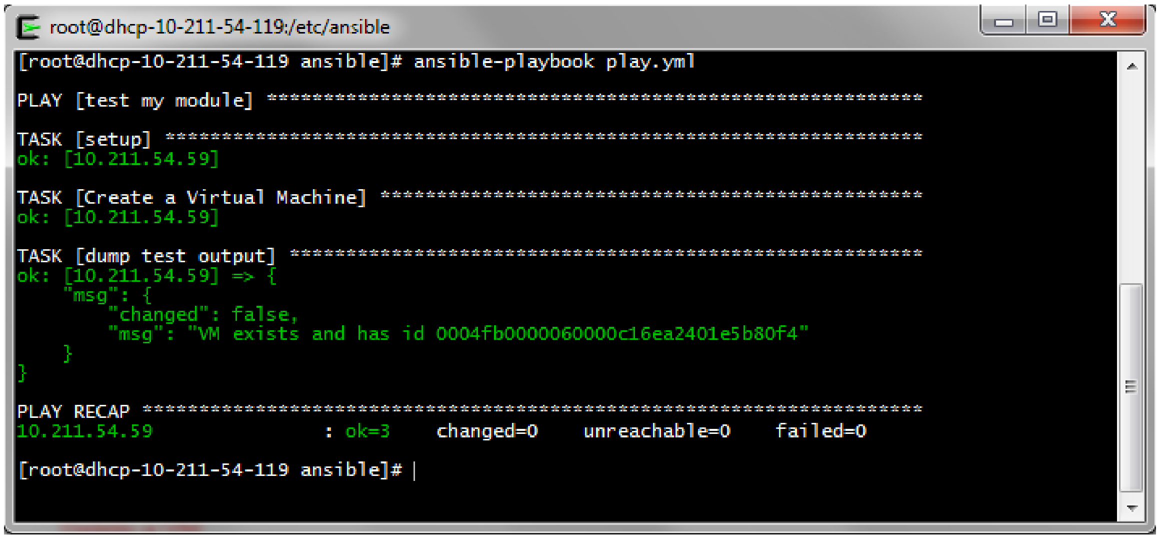
In this case, as the specified VM exists on PCA, we expect the module to not make any changes on PCA and display a changed=False message. Figure 9 shows the playbook play.yml used for this case. Figure 10 shows this execution on the VM ‘ST_vm1’ and Ansible returns a message saying “VM exists” and “changed=False”.



```
root@dhcp-10-211-54-119:/etc/ansible
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Create a Virtual Machine
      ovmm_vm:
        state: present
        name: ST_vm1
        ovm_user: admin
        ovm_pass: xxxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002
        server_pool: SP1
        repository: MyRepo
        memory: 4096
        vcpu_cores: 4
        boot_order: PXE
        networks: ['VMnet', 'VMnet']
        disks: [['disk1', 1073741824, 'MyRepo'], ['disk2', 1073741824, 'MyRepo']]

      register: testout

    - name: dump test output
      debug:
        msg: '{{ testout }}'
```

Figure 9. Playbook play.yml with state=present when the VM already exists on the remote host

If the specified VM already exists on PCA, Ansible makes no change to the remote system and returns the ID of the already existing VM with same name. In this case, the specified VM ‘ST_vm1’ exists and has ID ‘0004fb0000060000c16ea2401e5b80f4’



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook play.yml

PLAY [test my module] **********************************************************

TASK [setup] *******************************************************************
ok: [10.211.54.59]

TASK [Create a Virtual Machine] ************************************************
ok: [10.211.54.59]

TASK [dump test output] ********************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": false,
        "msg": "VM exists and has id 0004fb0000060000c16ea2401e5b80f4"
    }
}

PLAY RECAP *********************************************************************
10.211.54.59               : ok=3    changed=0    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

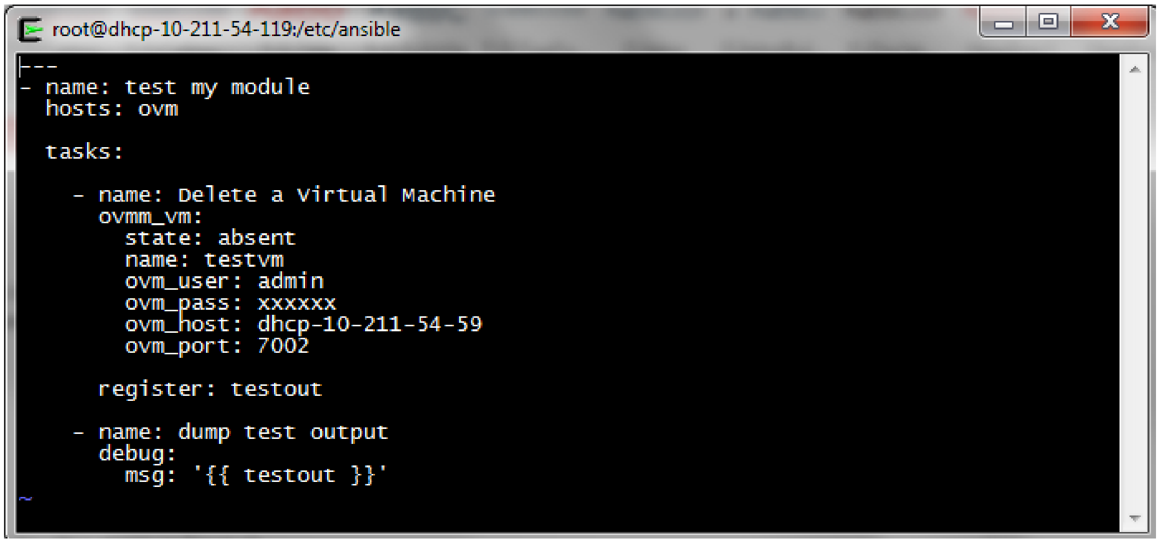
Figure 10. Execution of Playbook play.yml with state=present when the VM already exists on the remote host

Delete a VM

To delete a VM along with the VM disk mapping and Virtual Disk, the following arguments need to be specified while executing the module ovmm_vm.py

- **state=**absent
- **name**
- **ovm_user**
- **ovm_pass**
- **ovm_host**
- **ovm_port**

The playbook used for deleting the VM is shown on Figure 11.



```
root@dhcp-10-211-54-119:/etc/ansible
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Delete a Virtual Machine
      ovmm_vm:
        state: absent
        name: testvm
        ovm_user: admin
        ovm_pass: xxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002

      register: testout

    - name: dump test output
      debug:
        msg: '{{ testout }}'
~
```

Figure 11. Playbook deletevm.yml showing the task ‘Delete a VM’ on remote host ovm

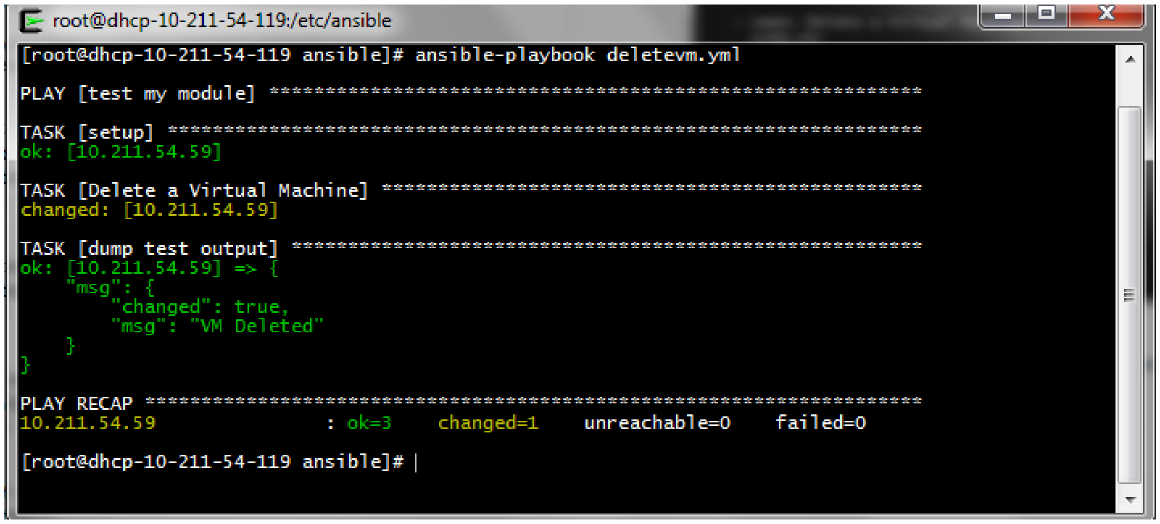
Note: The module ovmm_vm.py carried out the deletion of a VM by calling the deleteVM function (defined in ovmm_vm.py). This function deletes the VM disk mapping, virtual disk and the Virtual machine from the PCA as shown in Figure 13.

Case 1:
Specified VM is running on PCA, Desired State: absent

In this case, the VM ‘testvm’ exists and is running on the PCA, so Ansible should kill it and then delete it on the remote system and show a changed=True message to show that VM is deleted.

Figure 11 shows the playbook deletevm.yml used for this case. The VM “testvm” is present on the remote host “ovm” and should be deleted after execution of this playbook. The state=absent

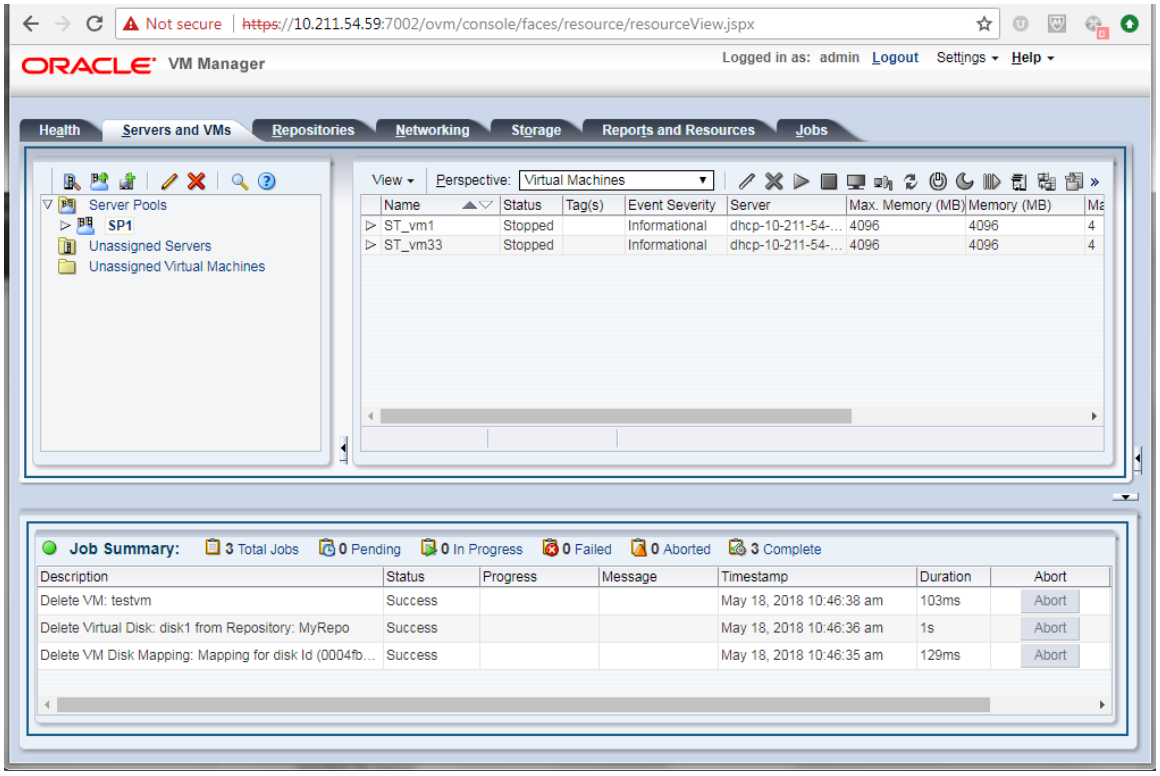
When this playbook is executed, we expect Ansible to change the state of remote host ‘ovm’ by deleting the specified VM ‘testvm’.
Figure 12 shows the result of execution of this playbook. The output of the execution clearly shows that task ‘Delete a Virtual Machine’ changed the state of remote host ovm. The test output shows “changed=True” and has a message “VM deleted”.



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook deletevm.yml

PLAY [test my module] *********************************************************

TASK [setup] *******************************************************************
ok: [10.211.54.59]

TASK [Delete a Virtual Machine] ************************************************
changed: [10.211.54.59]

TASK [dump test output] ********************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": true,
        "msg": "VM Deleted"
    }
}

PLAY RECAP *********************************************************************
10.211.54.59               : ok=3    changed=1    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

Figure 12. Execution of Playbook deletevm.yml when the remote host is not in desired state

The fact that the VM ‘testvm’ has actually been deleted can be verified by looking at the Oracle VM Manager GUI as shown in Figure 13.


Not secure | https://10.211.54.59:7002/ovm/console/faces/resource/resourceView.jspx
ORACLE VM Manager
Logged in as: admin Logout Settings Help
Health
Servers and VMs
Repositories
Networking
Storage
Reports and Resources
Jobs
Server Pools
SP1
Unassigned Servers
Unassigned Virtual Machines
View
Perspective: Virtual Machines

Name	Status	Tag(s)	Event Severity	Server	Max. Memory (MB)	Memory (MB)	Ma
ST_vm1	Stopped		Informational	dhcp-10-211-54-...	4096	4096	4
ST_vm33	Stopped		Informational	dhcp-10-211-54-...	4096	4096	4

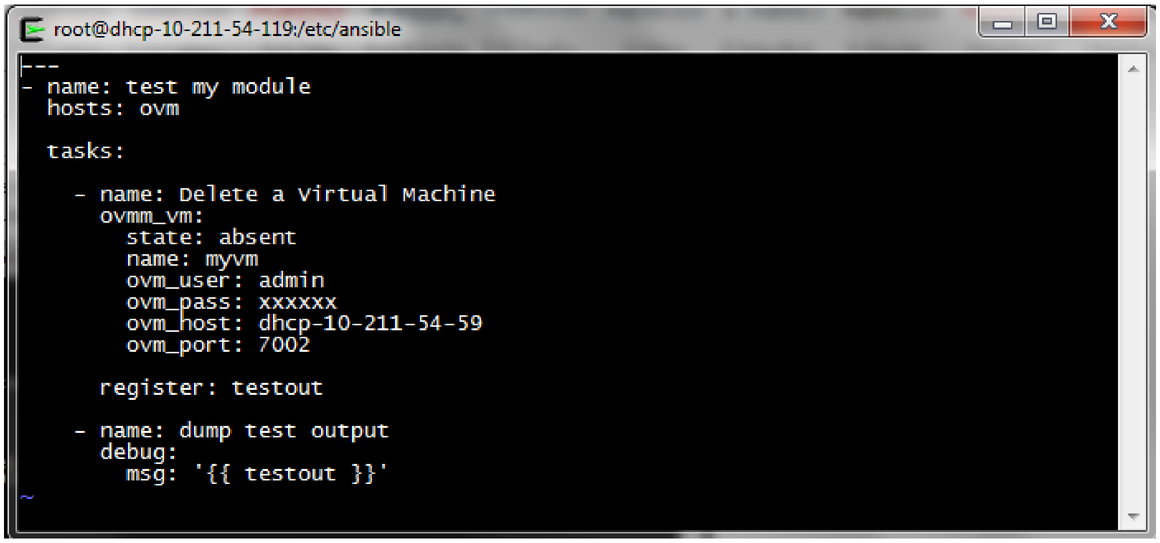
Job Summary: 3 Total Jobs 0 Pending 0 In Progress 0 Failed 0 Aborted 3 Complete

Description	Status	Progress	Message	Timestamp	Duration	Abort
Delete VM: testvm	Success			May 18, 2018 10:46:38 am	103ms	Abort
Delete Virtual Disk: disk1 from Repository: MyRepo	Success			May 18, 2018 10:46:36 am	1s	Abort
Delete VM Disk Mapping: Mapping for disk Id (0004fb...	Success			May 18, 2018 10:46:35 am	129ms	Abort

Figure 13. The VM ‘testvm’ is deleted as seen in the Oracle VM Manager GUI

Case 2:
Specified VM doesn't exist on PCA, Desired State: absent

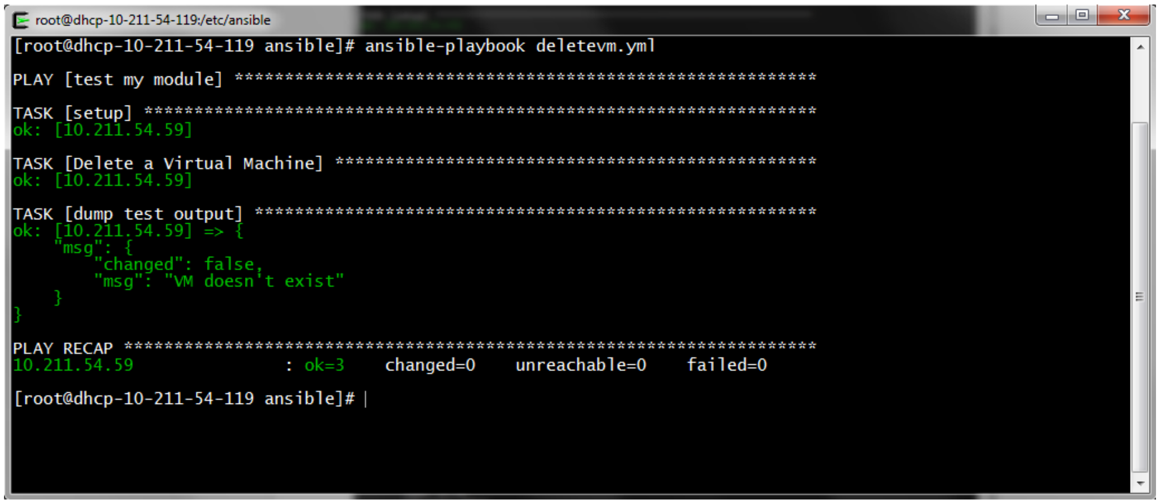
In this case, the VM 'myvm' doesn't exist on the PCA, so Ansible should not make any change to the remote system and should exit gracefully.
Figure 14 shows the playbook deletevm.yml used for this case. This playbook contains one play called "*test my module*" to be executed on the remote host *ovm* (specified in the /etc/ansible/hosts file). The playbook has 2 tasks – Delete a VM and dump test output.



```
root@dhcp-10-211-54-119:/etc/ansible
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Delete a Virtual Machine
      ovmm_vm:
        state: absent
        name: myvm
        ovm_user: admin
        ovm_pass: xxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002

      register: testout

    - name: dump test output
      debug:
        msg: '{{ testout }}'
~
```

Figure 14. Playbook play.yml showing two tasks to be executed on remote host 'ovm'

When this playbook is executed, we expect Ansible to not make any change on the remote system 'ovm' as the specified VM 'myvm' doesn't exist. In other words, the remote host 'ovm' is already in the desired state.
Figure 15 shows the result of execution of this playbook. The output of the execution clearly shows that nothing was changed on the remote host and our module returns a message "VM doesn't exist"



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook deletevm.yml

PLAY [test my module] **************************************************************

TASK [setup] ***********************************************************************
ok: [10.211.54.59]

TASK [Delete a Virtual Machine] ****************************************************
ok: [10.211.54.59]

TASK [dump test output] ************************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": false,
        "msg": "VM doesn't exist"
    }
}

PLAY RECAP *************************************************************************
10.211.54.59               : ok=3    changed=0    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

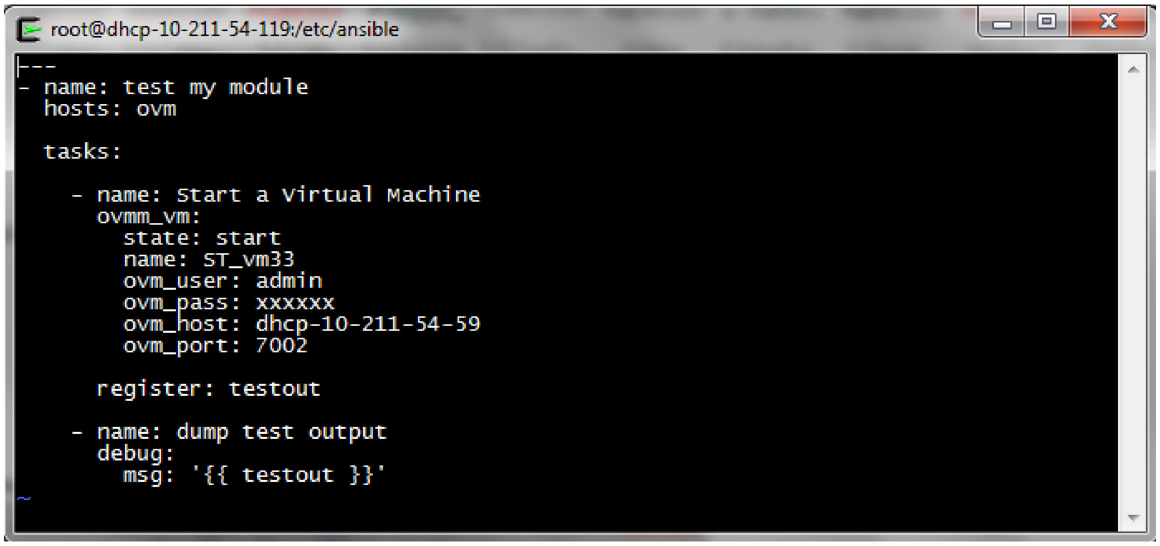
Figure 15. Execution of Playbook play.yml when the remote host is already in desired state

Start a VM

To start a VM, the following arguments need to be specified while executing the module ovmm_vm.py

- **state=**start
- **name**
- **ovm_user**
- **ovm_pass**
- **ovm_host**
- **ovm_port**

The playbook used for starting a VM on PCA is shown in Figure 16.



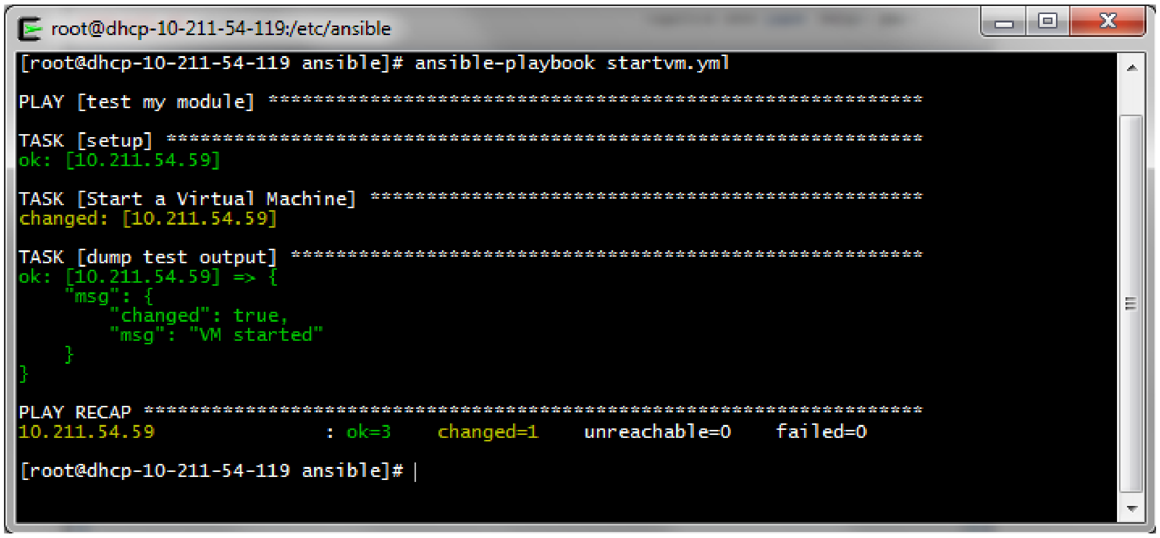
root@dhcp-10-211-54-119:/etc/ansible

```
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Start a Virtual Machine
      ovmm_vm:
        state: start
        name: ST_vm33
        ovm_user: admin
        ovm_pass: xxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002

      register: testout

    - name: dump test output
      debug:
        msg: '{{ testout }}'
~
```

Figure 16. Playbook startvm.yml showing the task ‘Start a VM’ on remote host ovm

Case:
Specified VM exists on PCA and is in stopped condition, Desired State: start

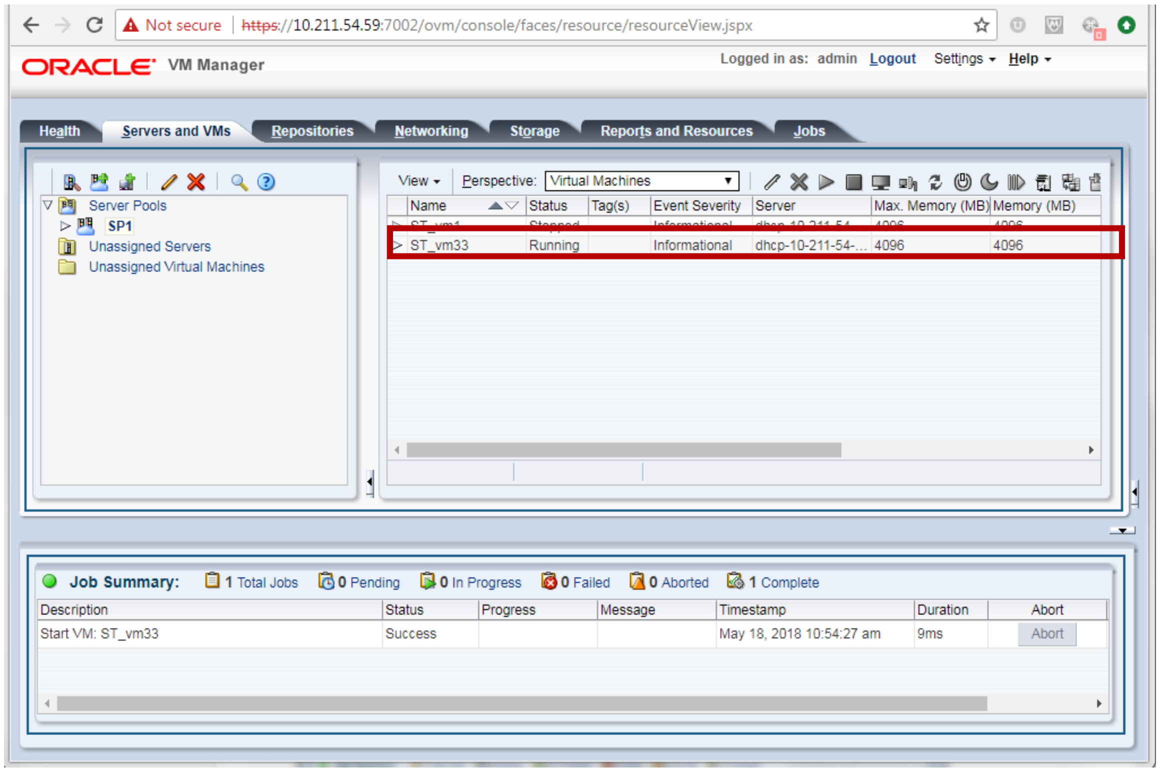
In this case, as the specified VM is stopped, we expect the module to start it on the PCA and display a changed=True message. Figure 17 shows this execution on the VM ‘ST_vm33’ and Ansible returns a message saying “VM started” and “changed=True”.



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook startvm.yml

PLAY [test my module] *****************************************************************

TASK [setup] **************************************************************************
ok: [10.211.54.59]

TASK [Start a Virtual Machine] ********************************************************
changed: [10.211.54.59]

TASK [dump test output] ***************************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": true,
        "msg": "VM started"
    }
}

PLAY RECAP ****************************************************************************
10.211.54.59               : ok=3    changed=1    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

Figure 17. Ansible changes the state of remote system to start the specified VM on remote host

The fact that the VM ‘ST_vm33’ has actually been started can be verified by looking at the Oracle VM Manager GUI as shown in Figure 18.



Not secure
https://10.211.54.59:7002/ovm/console/faces/resource/resourceView.jspx
ORACLE VM Manager
Logged in as: admin Logout Settings Help
Health
Servers and VMs
Repositories
Networking
Storage
Reports and Resources
Jobs
Server Pools
SP1
Unassigned Servers
Unassigned Virtual Machines
View
Perspective: Virtual Machines

Name	Status	Tag(s)	Event Severity	Server	Max. Memory (MB)	Memory (MB)
ST_vm1	Stopped		Informational	dhcp-10-211-54-	4096	4096
ST_vm33	Running		Informational	dhcp-10-211-54-...	4096	4096

Job Summary: 1 Total Jobs 0 Pending 0 In Progress 0 Failed 0 Aborted 1 Complete

Description	Status	Progress	Message	Timestamp	Duration	Abort
Start VM: ST_vm33	Success			May 18, 2018 10:54:27 am	9ms	Abort

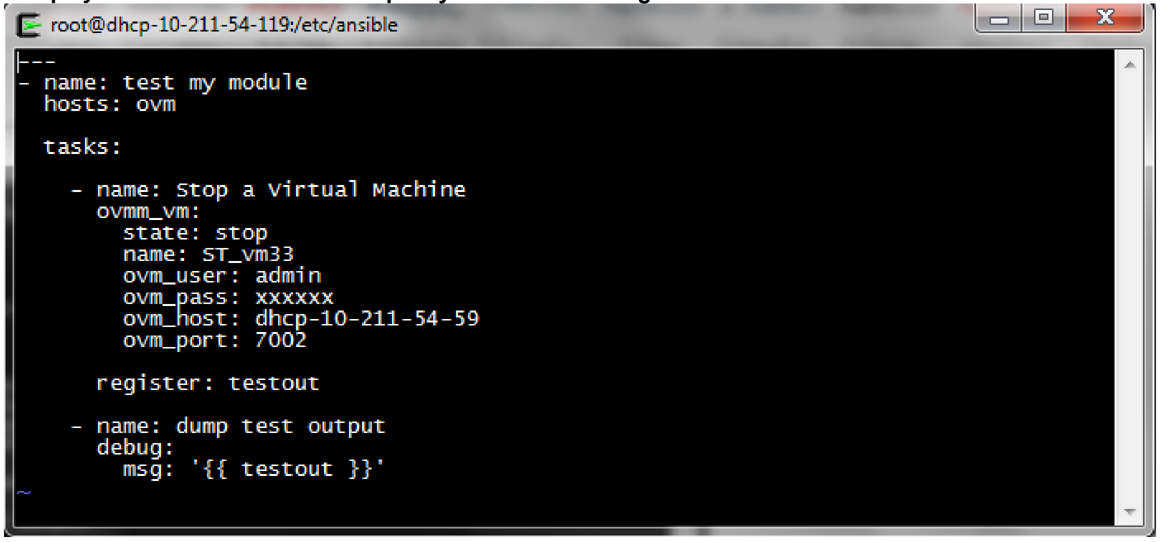
Figure 18. The VM ‘ST_vm33’ can be seen running in the Oracle VM Manager GUI

Stop a VM

To stop a VM, the following arguments need to be specified while executing the module ovmm_vm.py

- **state=**stop
- **name**
- **ovm_user**
- **ovm_pass**
- **ovm_host**
- **ovm_port**

The playbook used for this task ‘stopvm.yml’ is shown in Figure 19.



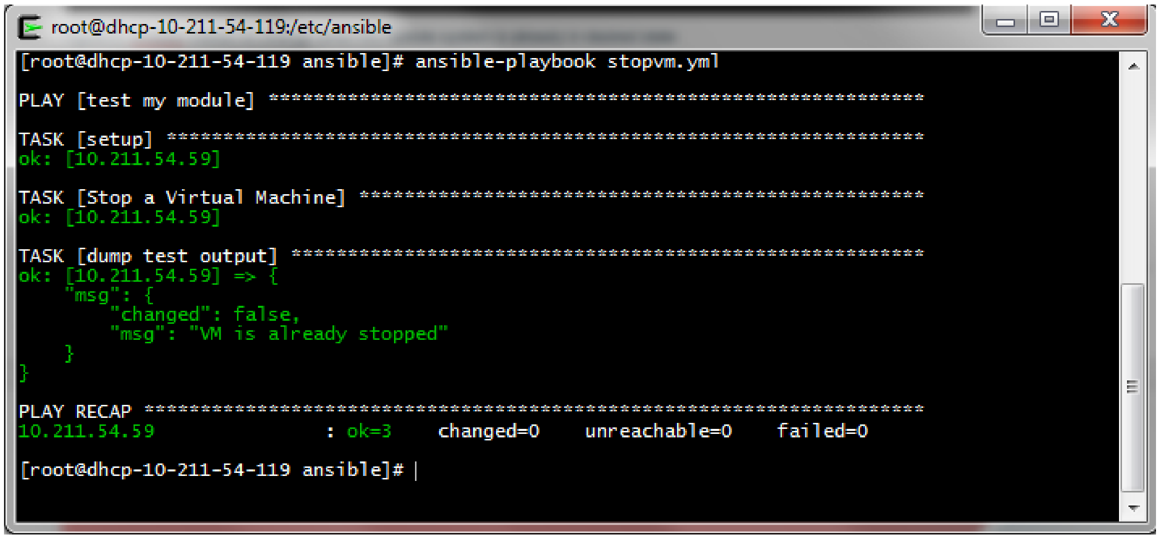
```
---
- name: test my module
  hosts: ovm

  tasks:

    - name: Stop a Virtual Machine
      ovmm_vm:
        state: stop
        name: ST_vm33
        ovm_user: admin
        ovm_pass: xxxxxx
        ovm_host: dhcp-10-211-54-59
        ovm_port: 7002

      register: testout

    - name: dump test output
      debug:
        msg: '{{ testout }}'
~
```

Figure 19. Playbook stopvm.yml showing task “Stop a VM’ to be executed on host ovm

Case:

Specified VM exists on PCA and is in stopped condition, Desired State: stop

In this case, as the specified VM is already stopped, we expect the module to not change anything on the PCA. This property of Ansible is referred to as **Idempotency**. Figure 20 shows this execution on the VM ‘ST_vm1’ and Ansible returns a message saying “VM is already stopped” as expected.



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook stopvm.yml

PLAY [test my module] *****************************************************************

TASK [setup] **************************************************************************
ok: [10.211.54.59]

TASK [Stop a Virtual Machine] *********************************************************
ok: [10.211.54.59]

TASK [dump test output] ***************************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": false,
        "msg": "VM is already stopped"
    }
}

PLAY RECAP ****************************************************************************
10.211.54.59               : ok=3    changed=0    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

Figure 20. Ansible makes no change when the remote system is already in desired state

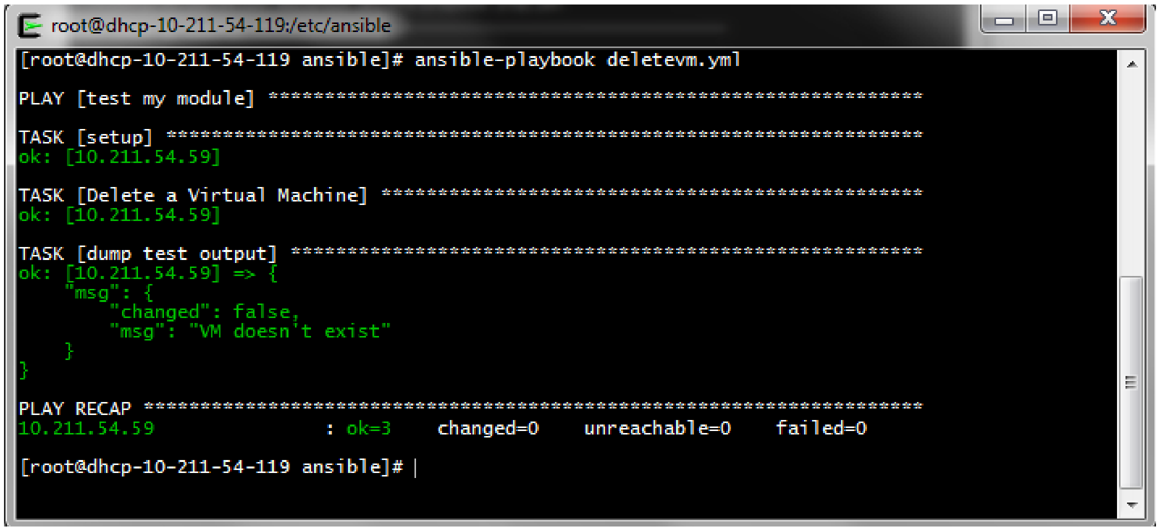
In this case, if the VM ‘ST_vm1’ was in the Running state prior to execution of the playbook, Ansible would have stopped the VM and displayed “changed: True and VM stopped” message.

Idempotency

The custom module ‘ovmm_vm.py’ is Idempotent i.e. running the module multiple times has no side effects. In other words, an operation is idempotent if the result of performing it once is exactly the same as the result of performing it repeatedly without any intervening actions.

To illustrate this concept, we can use the same playbook as shown in Figure 10. The playbook specifies that the VM ‘testvm’ should be absent from the target host ovm. Running the playbook once, deleted the specified VM from the target host. This means that the VM ‘testvm’ doesn’t exist anymore and running this playbook now should not make any change to the remote host ‘ovm’.

Figure 21 shows that when we run the playbook play.yml again, the task “Delete a Virtual Machine” doesn’t change anything on the remote host. We can also see from the test output that Ansible returns “changed=False” and the message says “VM doesn’t exist” as expected.
This playbook can now be run any number of times without making any change to the remote host. This is an important property of Ansible, which allows the user to only specify the desired state of the remote host without having to worry about the current state.



```
root@dhcp-10-211-54-119:/etc/ansible
[root@dhcp-10-211-54-119 ansible]# ansible-playbook deletevm.yml

PLAY [test my module] **********************************************************

TASK [setup] *******************************************************************
ok: [10.211.54.59]

TASK [Delete a Virtual Machine] ************************************************
ok: [10.211.54.59]

TASK [dump test output] ********************************************************
ok: [10.211.54.59] => {
    "msg": {
        "changed": false,
        "msg": "VM doesn't exist"
    }
}

PLAY RECAP *********************************************************************
10.211.54.59               : ok=3    changed=0    unreachable=0    failed=0

[root@dhcp-10-211-54-119 ansible]# |
```

Figure 21. Ansible module ‘ovmm_vm.py’ is Idempotent

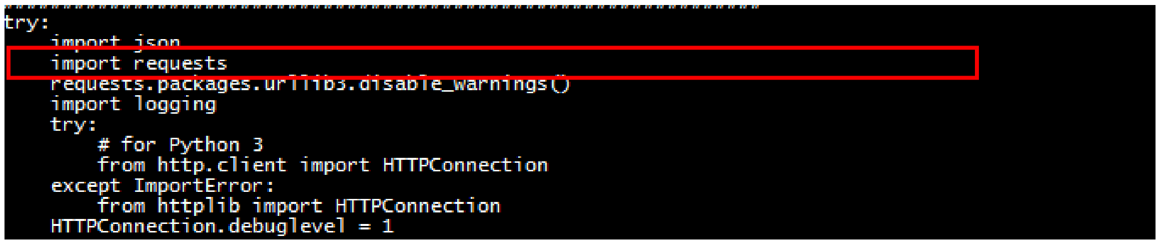
Appendix I: Code execution

Prerequisites for Ansible Control Machine

The following packages need to be installed on the Ansible Control Machine in order to initiate REST calls to PCA:

- **Requests:** HTTP library for python**.** Installed using **$ yum install python-requests**
- **urllib3:** HTTP client of python used by Requests to keep sessions alive and HTTP connection pooling. Installed using **$ yum install python-urllib3**

We import the Requests Python library to take care of our HTTP request processing using REST. The auth function returns a REST session by passing the username and password credentials for Oracle VM for basic authentication. It is possible to have certificate based SSL Authentication as described in the next section 'Authentication'.



```
try:
    import json
    import requests
    requests.packages.urllib3.disable_warnings()
    import logging
    try:
        # for Python 3
        from http.client import HTTPConnection
    except ImportError:
        from httplib import HTTPConnection
    HTTPConnection.debuglevel = 1
```

Figure 22. Prerequisites for executing Ansible module

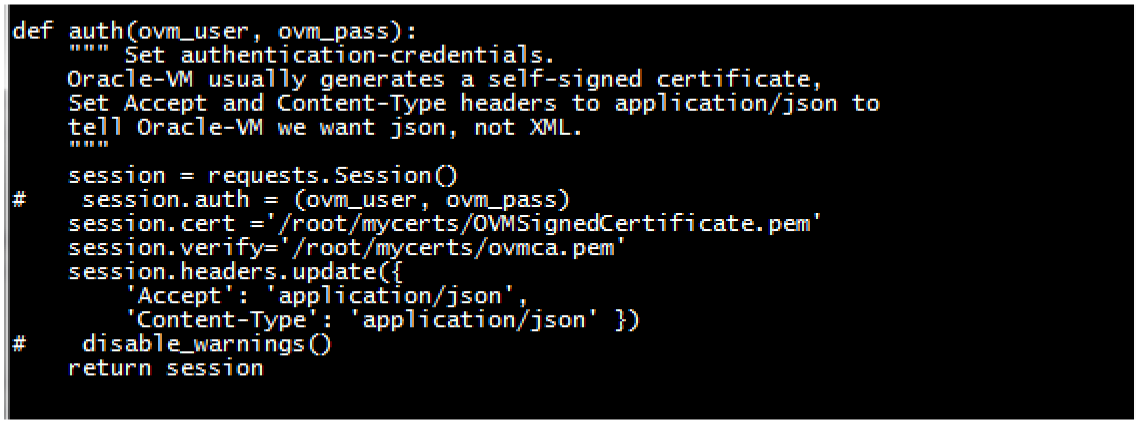
Authentication

This lab setup has the following components:

- **HTTP Basic Authentication**: Basic authentication is the simplest means of authenticating REST calls to OVM ws-api using REST; simply provide the user name and password used to log into the Oracle VM Manager (admin by default). This is simple solution, but weak security since it will send clear text password when you make a REST call.
- **SSL Certificate Authentication**: It is possible to use a signed SSL certificate to authenticate against Oracle VM Manager via the REST API. To do this, you must have the certificate and key stored in a single PEM file available to your application. The Requests library allows you to send a certificate with each request, or to set it to be used for every request within a session:

  ```
  session.cert='/path/to/mycertificate.pem
  ```

  As long as the certificate is valid and can be authenticated by Oracle VM Manager, the session is automatically logged in using your certificate.



```
def auth(ovm_user, ovm_pass):
    """ Set authentication-credentials.
    Oracle-VM usually generates a self-signed certificate,
    Set Accept and Content-Type headers to application/json to
    tell Oracle-VM we want json, not XML.
    """
    session = requests.Session()
#     session.auth = (ovm_user, ovm_pass)
    session.cert ='/root/mycerts/OVMSignedCertificate.pem'
    session.verify='/root/mycerts/ovmca.pem'
    session.headers.update({
        'Accept': 'application/json',
        'Content-Type': 'application/json' })
#     disable_warnings()
    return session
```

Figure 23. Initiating a REST session using auth function

Note: In the above figure, you can see that Certificate based authentication has been enabled to set up a HTTPS requests to the REST API. Thus, you don’t need the ovm_user and ovm_pass parameters to authenticate and hence, the line *session.auth = (ovm_user, ovm_pass) has been commented*.

Using SSL Certificate Authentication

PART I: Changes on Master Management Node of PCA

1) Create a New SSL Key on Master Management Node of PCA

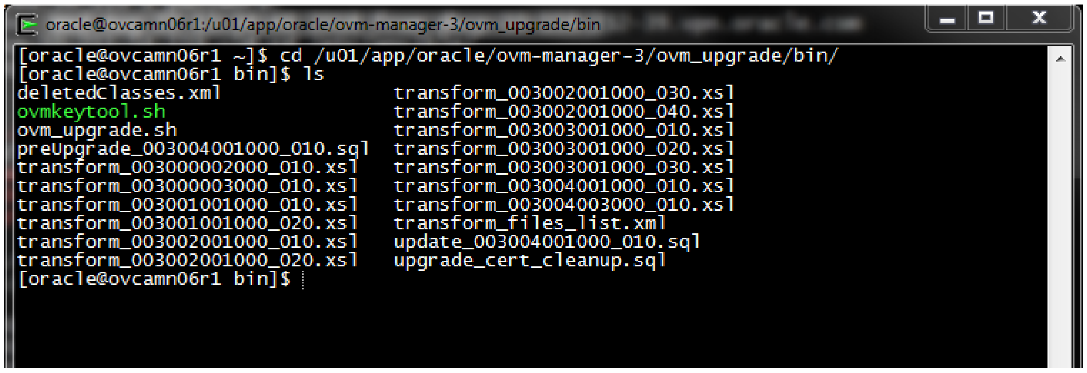
For accessing the Oracle VM REST API over Virtual IP of the Private Cloud Appliance, a new SSL key signed by the Oracle VM internal CA, has to be generated. Oracle VM Manager includes a key management utility to help manage SSL certificates.

Oracle VM Key Tool: `/u01/app/oracle/ovm-manager-3/ovm_upgrade/bin/ovmkeytool.sh`

By default, Oracle VM Manager generates and signs an SSL certificate that is valid for ten years from the date of installation. You can use the **genssl** command to generate a new SSL certificate that is signed by the Oracle VM Manager internal CA.

Here’s how you can generate a new SSL key for the VIP of PCA:

➢ Log onto the Master Management node of PCA as **oracle** user and navigate to the directory where ovmkeytool.sh is located.



```
oracle@ovcamn06r1:/u01/app/oracle/ovm-manager-3/ovm_upgrade/bin
[oracle@ovcamn06r1 ~]$ cd /u01/app/oracle/ovm-manager-3/ovm_upgrade/bin/
[oracle@ovcamn06r1 bin]$ ls
deletedClasses.xml                transform_003002001000_030.xsl
ovmkeytool.sh                     transform_003002001000_040.xsl
ovm_upgrade.sh                    transform_003003001000_010.xsl
preUpgrade_003004001000_010.sql   transform_003003001000_020.xsl
transform_003000002000_010.xsl    transform_003003001000_030.xsl
transform_003000003000_010.xsl    transform_003004001000_010.xsl
transform_003001001000_010.xsl    transform_003004003000_010.xsl
transform_003001001000_020.xsl    transform_files_list.xml
transform_003002001000_010.xsl    update_003004001000_010.sql
transform_003002001000_020.xsl    upgrade_cert_cleanup.sql
[oracle@ovcamn06r1 bin]$
```

Figure 24. ovmkeytool on the Master Management Node of Oracle PCA

➢ Use the **genssl** command with ovmkeytool.sh to generate a new SSL certificate

```
[oracle@ovcamn06r1 bin]$ ./ovmkeytool.sh gensslkey
Path for SSL keystore: [/u01/app/oracle/ovm-manager-
3/domains/ovm_domain/security/ovmssl.jks]
The hostname should be the fully qualified hostname of the system
(this is the hostname you'd use to access this system from outside the
local domain).  Depending on your machine setup the value below may not
be
correct.
Fully qualified hostname: 10.147.x.x
Key distinguished name is "CN=10.147.x.x, OU=Oracle VM Manager,
  O=Oracle Corporation, L=Redwood City, ST=California, C=US".  Use these
values? [yes]
Alternate hostnames (separated by commas): [10.147.x.x,
myserver.example.com]
You may either specify passwords or use random passwords.
If you choose to use a random password, only WebLogic, the Oracle VM
Manager, and this application will have access to the information stored
in this keystore.
Use random passwords? [yes]
Generating SSL key and certificate and persisting them to the
keystore...
Updating keystore information in WebLogic
Oracle MiddleWare Home (MW_HOME): [/u01/app/oracle/Middleware]
WebLogic domain directory: [/u01/app/oracle/ovm-manager-
3/domains/ovm_domain]
Oracle WebLogic Server name: [AdminServer]
WebLogic username: [weblogic]
WebLogic password: [********]
WLST session logged at: /tmp/wlst-session178461015146984067.log
```

Note that the command prompts you to provide the values for various steps through the procedure as the new SSL certificate is generated. Notably, you must enter a valid fully qualified domain name for the server. This value is used for the hostname in the SSL certificate and must match the hostname that is used to access the Oracle VM Manager web-based user interface.

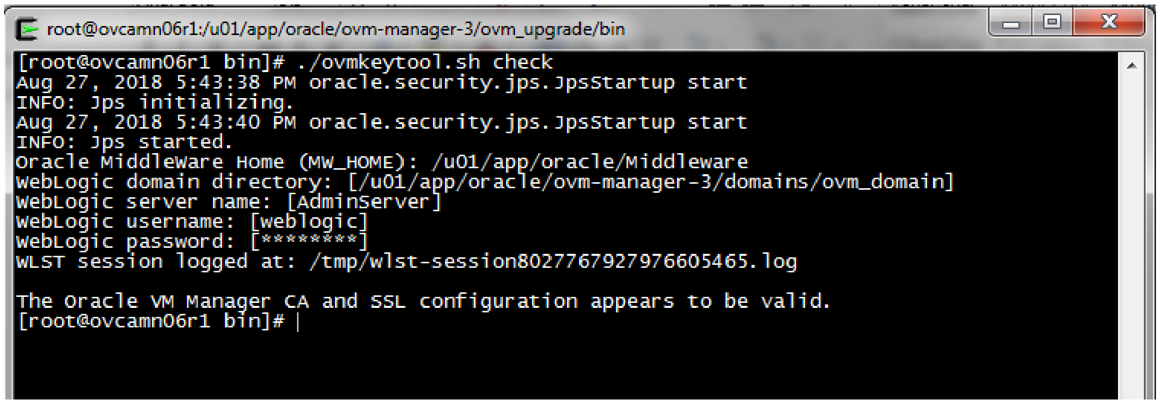
➢ Restart the ovca service and wait for 3-5 minutes to make sure everything is running

```
 [root@ovcamn06r1 bin]$ service ovca restart

[root@ovcamn06r1 bin]$ service ovca status
Checking Oracle Fabric Manager: Running
```

```
MySQL running (1658352)                                    [  OK  ]
Oracle VM Manager is running...
Oracle VM Manager CLI is running...
tinyproxy (pid 1660422 1660421 1660420 1660419 1660418 1660416 1660415
1660414 1660413 1660412 1660408) is running...
dhcpd (pid  1660442) is running...
snmptrapd (pid  1660460) is running...
log server (pid 1657404) is running...
remaster server (pid 1657406) is running...
http server (pid 1660463) is running...
taskmonitor server (pid 1660467) is running...
xmlrpc server (pid 1660465) is running...
nodestate server (pid 1660469) is running...
sync server (pid 1660471) is running...
monitor server (pid 1660473) is running...
```

➢ Use the check command to verify the current CA and SSL configuration. If any issues exist with the configuration, the command displays information to help you resolve them.

The following is an example of the check command:



root@ovcamn06r1:/u01/app/oracle/ovm-manager-3/ovm_upgrade/bin

```
[root@ovcamn06r1 bin]# ./ovmkeytool.sh check
Aug 27, 2018 5:43:38 PM oracle.security.jps.JpsStartup start
INFO: Jps initializing.
Aug 27, 2018 5:43:40 PM oracle.security.jps.JpsStartup start
INFO: Jps started.
Oracle Middleware Home (MW_HOME): /u01/app/oracle/Middleware
WebLogic domain directory: [/u01/app/oracle/ovm-manager-3/domains/ovm_domain]
WebLogic server name: [AdminServer]
WebLogic username: [weblogic]
WebLogic password: [********]
WLST session logged at: /tmp/wlst-session8027767927976605465.log

The Oracle VM Manager CA and SSL configuration appears to be valid.
[root@ovcamn06r1 bin]# |
```

Figure 25. Running ovmkeytool.sh to check whether OVM CA and SSL configuration are correct

We can see that the Oracle VM Manager Certificate Authority and SSL configuration have been correctly configured. Now, this certificate can be used for SSL verification of the REST session.

PART II: Changes on Ansible Control Machine

Create your own signed certificate file that will be referenced in your function/code when instantiating a REST session object; the name of the signed certificate in this example is OVMSignedCertificate.pem. However, the file can be any name and live on any server where your python code/script is installed and should ultimately be moved to a directory where your code will expect to find it; the certificate should work from any server including the Oracle

VM management server/node - it does not need to exist on the Oracle VM management node unless that is where you are running your code.

The process below is the same for Oracle VM 3.3 and 3.4.

1) Create temporary key files

This step is only performed once to create the signed certificate file that will be used by your application/script/code to access Oracle VM WS-API using REST; the certificate file that will be used in your python code is used to authenticate against the certificate file that is shipped with Oracle VM (ovmca.pem).

First create a key and certificate locally using the OpenSSL tool available on the Oracle VM management server. The mykey.pem file shown in the example below is an important file that is used in a later step to create the certificate you will use in your code; it must not have a passphrase. The first two lines below create this important file. It takes two steps because specifying -des3 (des3 is higher security) on the first line requires at least a four character passphrase; give it any passphrase that comes to mind because the second line below removes the passphrase.

```
$ openssl genrsa -des3 -out mykey-with-pass.pem 2048
$ openssl rsa -in mykey-with-pass.pem -out mykey.pem
$ openssl req -new -key mykey.pem -out mycertreq.csr
$ openssl x509 -req -days 365 -in mycertreq.csr -signkey mykey.pem -out mycert.pem
```

2) Create signed certificate file using python

You will need to include the absolute path for "mycert.pem" file in the example below if you started python in a different directory from the one you created the temporary cert files.

Step 2.1: Add your certificate to Oracle VM Manager

This step adds the certification key created in Step 1 to the Oracle VM Manager permanently. It also gets a key that is returned from Oracle VM Manager and writes it to one of our temporary files.

```
import json
import requests
from requests.packages.urllib3 import disable_warnings
from requests.packages.urllib3.exceptions import InsecureRequestWarning

s=requests.Session()
s.verify=False
s.auth=('admin','Welcome1')
s.headers.update({'Accept': 'application/json', 'Content-Type': 'application/json'})
disable_warnings()

cert=open('mycert.pem').read()
body={'certificate': cert}
r=s.post('https://<mymanager.fqdn>:7002/ovm/core/wsapi/rest/Utilities/Certificate?sign=True',
 data=json.dumps(body))
```

```
signed_cert=r.json()['certificate']
f=open('signed.pem','w')
f.write(signed_cert)
f.close()
```

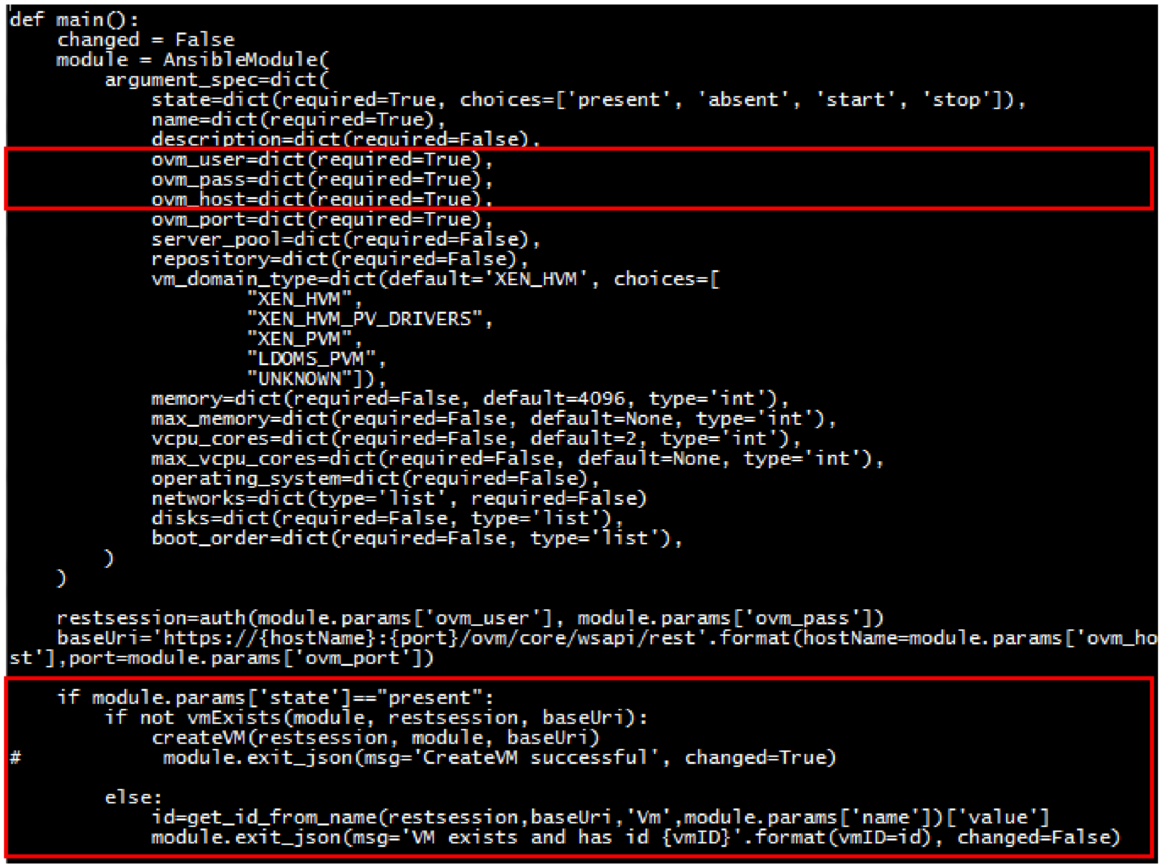
Step 2.2: Create the final signed certificate file

This step creates the file you will point to in your python code when creating a REST session. To use this certificate to authenticate to either the REST or SOAP API using Python, it should be combined with its key.

You can do this easily on the command line: **$ cat signed.pem mykey.pem >> OVMSignedCertificate.pem**

Once you have created a certificate, you can use the REST API to sign it (python, using requests module).

Creating a Virtual Machine



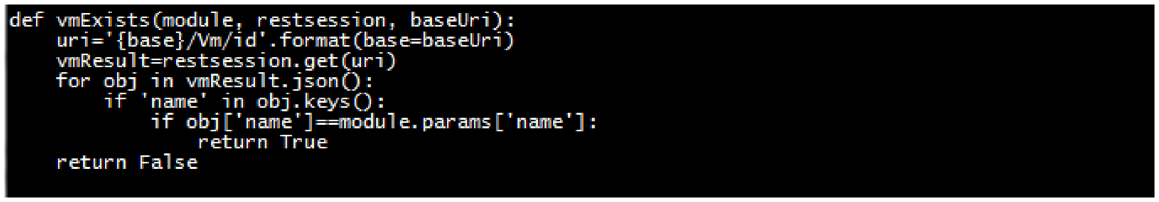
```
def main():
    changed = False
    module = AnsibleModule(
        argument_spec=dict(
            state=dict(required=True, choices=['present', 'absent', 'start', 'stop']),
            name=dict(required=True),
            description=dict(required=False),
            ovm_user=dict(required=True),
            ovm_pass=dict(required=True),
            ovm_host=dict(required=True),
            ovm_port=dict(required=True),
            server_pool=dict(required=False),
            repository=dict(required=False),
            vm_domain_type=dict(default='XEN_HVM', choices=[
                    "XEN_HVM",
                    "XEN_HVM_PV_DRIVERS",
                    "XEN_PVM",
                    "LDOMS_PVM",
                    "UNKNOWN"]),
            memory=dict(required=False, default=4096, type='int'),
            max_memory=dict(required=False, default=None, type='int'),
            vcpu_cores=dict(required=False, default=2, type='int'),
            max_vcpu_cores=dict(required=False, default=None, type='int'),
            operating_system=dict(required=False),
            networks=dict(type='list', required=False)
            disks=dict(required=False, type='list'),
            boot_order=dict(required=False, type='list'),
        )
    )

    restsession=auth(module.params['ovm_user'], module.params['ovm_pass'])
    baseUri='https://{hostName}:{port}/ovm/core/wsapi/rest'.format(hostName=module.params['ovm_ho
st'],port=module.params['ovm_port'])

    if module.params['state']=="present":
        if not vmExists(module, restsession, baseUri):
            createVM(restsession, module, baseUri)
#             module.exit_json(msg='CreateVM successful', changed=True)

        else:
            id=get_id_from_name(restsession,baseUri,'Vm',module.params['name'])['value']
            module.exit_json(msg='VM exists and has id {vmID}'.format(vmID=id), changed=False)
```

Figure 26. ‘Main’ function of the ovmm_vm.py module

The main function takes arguments from the user (provided as YAML playbook). Above figure shows that the baseUri in our examples is https://<ovm_manager_host>:7002/ovm/core/wsapi/rest.

Note: Once SSL verification has been set up as shown in the previous steps, we no longer need to supply clear text ovm_user and ovm_password. In this case, we can make **(required=False)** for ovm_user and ovm_pass parameters. In that case, you don’t need to supply these as arguments in the playbook.

When the desired state is specified as “present”, the module checks is the specified VM exists. This is done by a call to the function ‘vmExists’.

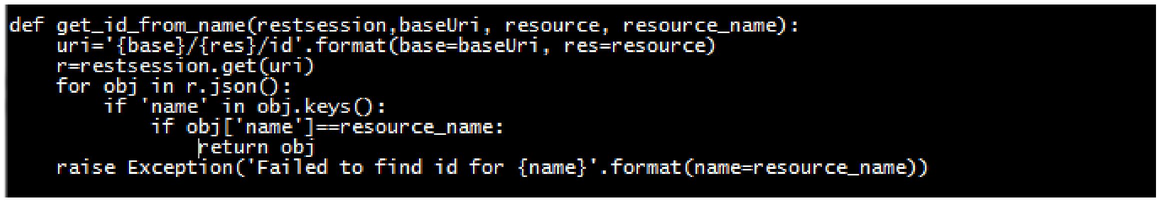


```
def vmExists(module, restsession, baseUri):
    uri='{base}/Vm/id'.format(base=baseUri)
    vmResult=restsession.get(uri)
    for obj in vmResult.json():
        if 'name' in obj.keys():
            if obj['name']==module.params['name']:
                return True
    return False
```

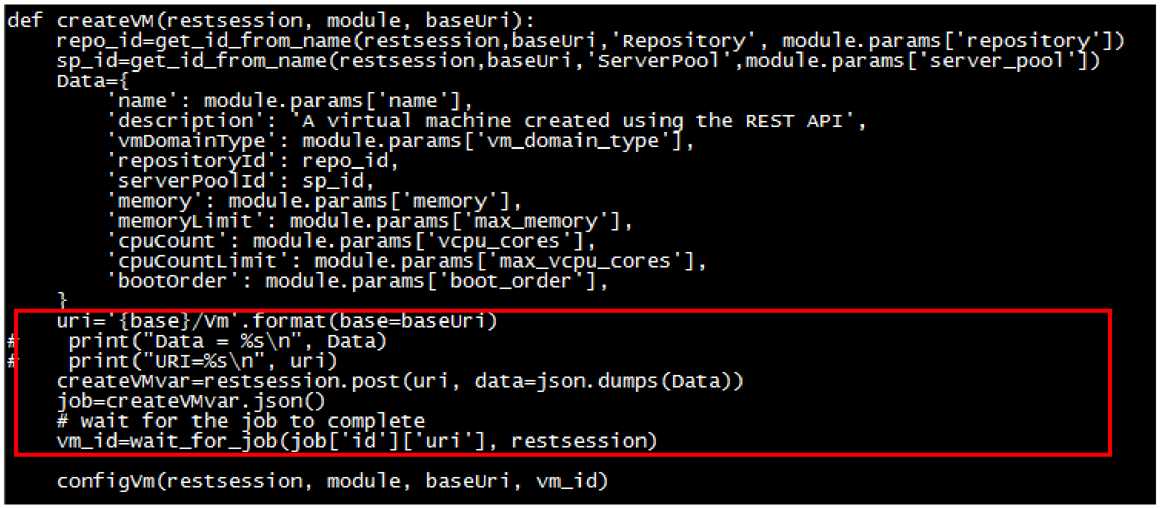
Figure 27. ‘vmExists’ function of the ovmm_vm.py module

If the VM with the name specified by the user doesn’t exist on the target host, the **createVM** function is called. It takes the restsession, modeule and the baseUri described above as arguments.

First, the ID for repository and server pool on the PCA where the VM needs to be created, is obtained by calling the **get_id_from_name** function. The VM Data provided by the user as input YAML file is then passed as JSON input to the POST request to uri https://<ovm_manager_host>:7002/ovm/core/wsapi/rest/Vm



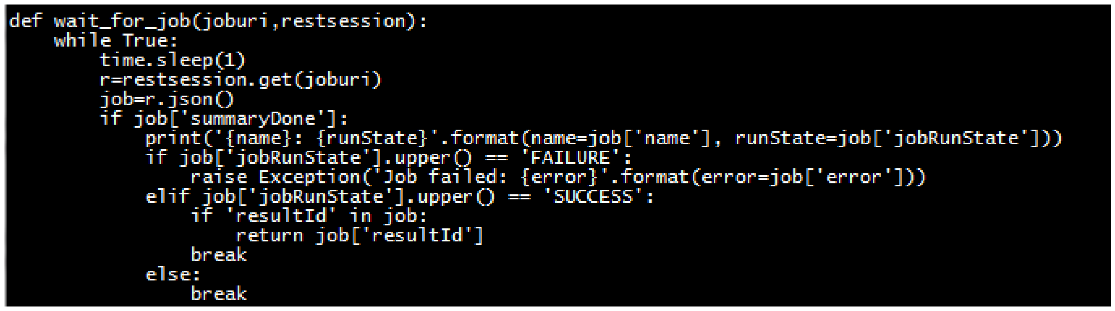
```
def get_id_from_name(restsession,baseUri, resource, resource_name):
    uri='{base}/{res}/id'.format(base=baseUri, res=resource)
    r=restsession.get(uri)
    for obj in r.json():
        if 'name' in obj.keys():
            if obj['name']==resource_name:
                return obj
    raise Exception('Failed to find id for {name}'.format(name=resource_name))
```



```
def createVM(restsession, module, baseUri):
    repo_id=get_id_from_name(restsession,baseUri,'Repository', module.params['repository'])
    sp_id=get_id_from_name(restsession,baseUri,'ServerPool',module.params['server_pool'])
    Data={
        'name': module.params['name'],
        'description': 'A virtual machine created using the REST API',
        'vmDomainType': module.params['vm_domain_type'],
        'repositoryId': repo_id,
        'serverPoolId': sp_id,
        'memory': module.params['memory'],
        'memoryLimit': module.params['max_memory'],
        'cpuCount': module.params['vcpu_cores'],
        'cpuCountLimit': module.params['max_vcpu_cores'],
        'bootOrder': module.params['boot_order'],
    }
    uri='{base}/Vm'.format(base=baseUri)
#    print("Data = %s\n", Data)
#    print("URI=%s\n", uri)
    createVMvar=restsession.post(uri, data=json.dumps(Data))
    job=createVMvar.json()
    # wait for the job to complete
    vm_id=wait_for_job(job['id']['uri'], restsession)

    configVm(restsession, module, baseUri, vm_id)
```

Figure 28. ‘get_id_from_name’ function (above) and ‘CreateVM’ function of the ovmm_vm.py module

The vm_id variable is obtained by calling the **wait_for_job** function, which returns the ID of the job once it is successfully completed or an error if the job run failed.

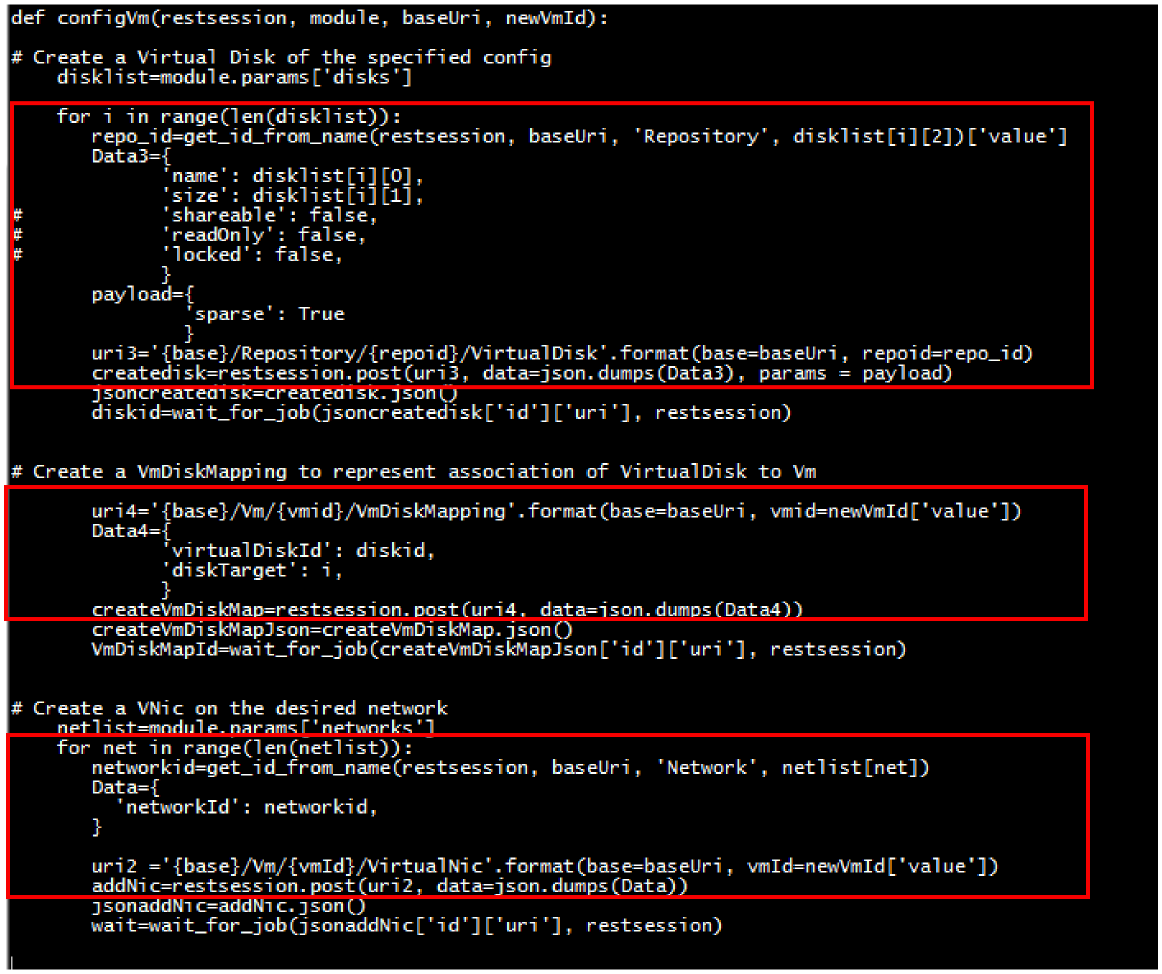


```
def wait_for_job(joburi,restsession):
    while True:
        time.sleep(1)
        r=restsession.get(joburi)
        job=r.json()
        if job['summaryDone']:
            print('{name}: {runState}'.format(name=job['name'], runState=job['jobRunState']))
            if job['jobRunState'].upper() == 'FAILURE':
                raise Exception('Job failed: {error}'.format(error=job['error']))
            elif job['jobRunState'].upper() == 'SUCCESS':
                if 'resultId' in job:
                    return job['resultId']
                break
            else:
                break
```

The result of the createVM function is a bare bones VM with the specified name, cpu cores, memory in the specified location on a PCA. This VM still doesn't have any disks or Virtual NICs attached. The next step is to call configvm function.

Attaching Disks and VNICs to a newly created VM

After a VM of specified configuration is created on the PCA, the next step is to attach VNICs on specified networks and VirtualDisks of specified size, name and in the desired repository on PCA. This is done by a call to configvm function after the createvm function has finished creating a bare bones VM.



```
def configVm(restsession, module, baseUri, newVmId):

# Create a Virtual Disk of the specified config
    disklist=module.params['disks']

    for i in range(len(disklist)):
       repo_id=get_id_from_name(restsession, baseUri, 'Repository', disklist[i][2])['value']
       Data3={
              'name': disklist[i][0],
              'size': disklist[i][1],
#             'shareable': false,
#             'readOnly': false,
#             'locked': false,
              }
       payload={
                'sparse': True
                }
       uri3='{base}/Repository/{repoid}/VirtualDisk'.format(base=baseUri, repoid=repo_id)
       createdisk=restsession.post(uri3, data=json.dumps(Data3), params = payload)
       jsoncreatedisk=createdisk.json()
       diskid=wait_for_job(jsoncreatedisk['id']['uri'], restsession)


# Create a VmDiskMapping to represent association of VirtualDisk to Vm

       uri4='{base}/Vm/{vmid}/VmDiskMapping'.format(base=baseUri, vmid=newVmId['value'])
       Data4={
              'virtualDiskId': diskid,
              'diskTarget': i,
              }
       createVmDiskMap=restsession.post(uri4, data=json.dumps(Data4))
       createVmDiskMapJson=createVmDiskMap.json()
       VmDiskMapId=wait_for_job(createVmDiskMapJson['id']['uri'], restsession)


# Create a VNic on the desired network
    netlist=module.params['networks']
    for net in range(len(netlist)):
       networkid=get_id_from_name(restsession, baseUri, 'Network', netlist[net])
       Data={
         'networkId': networkid,
       }

       uri2 ='{base}/Vm/{vmId}/VirtualNic'.format(base=baseUri, vmId=newVmId['value'])
       addNic=restsession.post(uri2, data=json.dumps(Data))
       jsonaddNic=addNic.json()
       wait=wait_for_job(jsonaddNic['id']['uri'], restsession)
```

The ‘configvm’ function does the following things:

➢ **Create Virtual Disks.**

The user specifies the details for Virtual Disks to be attached to the newly created VM in the form of a parameter ‘disks’ via the playbook.
The data type for ‘disks’ is list and Ansible expects the input to be in the form of a list of lists as shown in playbook in Figure 6.

disks : [[‘disk1_name’, disk1_size bytes, ‘disk1_repo’] , [‘disk2_name’, disk2_size bytes, ‘disk2_repo’]..]

Disk 1 parameters Disk 2 parameters

The function then loops over the list, creating each virtual disk of required name, size and on the specified repository. This is done by sending a POST request to uri3 as shown in Figure above.

- The request path contains the **repository ID** of the repo where the Virtual Disk needs to be created.
- The request body contains the ‘**name**’ and ‘**size**’ as required inputs
- The parameter ‘**sparse**’ should be passed as a query parameter. Set it to True to create a Sparse disk image.

➢ **Create VM Disk Mapping.**

After creating each disk, Oracle VM also needs to create a VM Disk Mapping to represent the association of Virtual Disk and VM. Thus, for each virtual disk, a Disk Mapping is created by sending a POST request to uri4 as shown in above figure.

- The request path contains the **VM ID** of the VM to which the Virtual NICs needs to be associated.
- The request body contains the ‘**virtualDiskId**’ and ‘**diskTarget**’ as required inputs. VirtualDiskId is obtained after the Oracle VM finishes creating a disk in last step. diskTarget maps the slot on the VM to which the VirtualDisk should be attached.

➢ **Create Virtual NICs.**

The user specifies the names of the networks on which the VirtualNics for the VM need to be created as a parameter ‘networks’ in the playbook. The data type for ‘networks’ is list and Ansible expects the input to be in

networks : [‘Network_name’, ‘Network_name’, …]

If you need 2 VNICs on the same network, specify the network name twice as shown in Figure 6.

The function then loops over the list, creating virtual NICs on each specified network by sending a POST request to uri2 as shown in Figure above.

- The request path contains the **VM ID** to which the VNIC needs to be associated.
- The request body contains the ‘**networkId**’ as required inputs

The module exits with a message “VM created” if the execution was successful and returns an error otherwise.

Conclusion

Ansible is a simple IT automation tool that can automate repeatable IT tasks, thus reducing human error and also bringing down the operating expenses. The custom Ansible module 'ovmm_vm' can be used to manage the complete lifecycle of a Virtual Machine on Oracle Private Cloud Appliance by automating creation, deletion, starting and stopping of a Virtual Machine. This paper described the Ansible Ad-Hoc command and the Playbook method of executing the custom Ansible module.

The module 'ovmm_vm.py' can be used to rapidly deploy a minimal server Virtual Machine on PCA. Ansible Playbooks for deploying Oracle Databases and applications can be used in conjunction with this module to rapidly deploy a VM with specified software in minutes on a PCA.

Resources

Oracle VM REST API guide

Ansible documentation

Sample code is provided for educational purposes or to assist your development or administration efforts. Your use rights and restrictions for each sample code item are described in the applicable license agreement. Except as may be expressly stated in the applicable license agreement or product documentation, sample code is provided "as is" and is not supported by Oracle.

ORACLE®

Oracle Corporation, World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065, USA

Worldwide Inquiries
Phone: +1.650.506.7000
Fax: +1.650.506.7200

CONNECT WITH US

blogs.oracle.com/oracle
facebook.com/oracle
twitter.com/oracle
oracle.com

Integrated Cloud Applications & Platform Services

Copyright © 2018, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only, and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document, and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group. *0918*

Automate Infrastructure Lifecycle Management on PCA using Ansible
September 2018
Author: Sonit Tayal

Oracle is committed to developing practices and products that help protect the environment