Homework 5

Iman Tabrizian
ECE1508

February 17, 2019

In order to deploy the mentioned topology I used the general structure like below:

1. Checking for existence of a previous switches / hosts
2. If hosts / switches doesn't exist boot the VM
3. Creating OVS bridge named *br1* in each switch / host
4. Going through each of the connections between vms and switches and checking weather connection existed before
5. Creating new vxlan tunnel if the connection didn't exist

Now I want to describe each of the functions and what they do step by step.

```python
def getVM(vmName):
    nova = getNovaClient()
    vmName = overlay_config.username + '-' + vmName
    server = nova.servers.find(name=vmName)
    return server
```

This function receives *vmName* as input and returns the vm object returned by nova client.

```python
def getNovaClient():
    nova = novaClient.Client(
        overlay_config.username,
        overlay_config.password,
        overlay_config.tenant_name,
        overlay_config.auth_url,
        region_name=overlay_config.region,
        no_cache=True
        )
    return nova
```

This function simply returns the nova client by filling in the appropriate credentials from the *overlay_config* file.

```python
def bootVM(vmName):
    assert type(vmName) in (str, unicode), "'vmName' is type %s" % type(vmName)

    # Pre-pend vmName with your username
    vmName = overlay_config.username + '-' + vmName

    logger.info("Creating VM %s" % vmName)

    # STUDENTS FILL THIS PART OUT
    nova = getNovaClient()
    net_name = overlay_config.tenant_name + '-net'
    net = nova.networks.find(label=net_name)
    flavor = nova.flavors.find(name=overlay_config.flavor)
    image = nova.images.find(name=overlay_config.image)
    server = nova.servers.create(
        vmName,
        image,
        flavor,
        key_name=overlay_config.key_name,
        security_groups=[overlay_config.username],
        nics=[{'net-id': net.id}]
        )

    waitUntilVMActive(server)

    return server
```

This function creates a new vm prefixed using the username. Then, it retrieves the flavor, network and image objects from using the nova client and finally it creates the server using *nova.servers.create*. In the end, function waits until the vm becomes active and then returns.

```python
def setOverlayInterface(hostVMObj, hostOverlayIP):
    logger.info("Setting overlay for %s with IP %s" %
                        (hostVMObj.name, hostOverlayIP))
    networkName = overlay_config.tenant_name + '-net'

    hostVMIP = hostVMObj.networks.get(networkName)[0]
    hostSSH = getSSHSession(hostVMIP, 'ubuntu', 'savi')
    # Ensure OVS daemon is up and running
    waitUntilOVSActive(hostSSH)

    # STUDENTS FILL THIS PART OUT
    runCommandOverSSH(hostSSH, 'sudo ovs-vsctl --may-exist add-br br1')
    if hostOverlayIP is not None:
        runCommandOverSSH(hostSSH, 'sudo ovs-vsctl --may-exist add-port br1 br1-internal
        runCommandOverSSH(hostSSH, 'sudo ifconfig br1-internal %s/24 mtu 1450 up' % host
```

This function simply adds an ovs bridge. If the *hostOverlayIP* parameter is not equal to *None*, it will also add one port to this bridge and configure its mtu as mentioned in the assignment.

```
def connectNodes(node1, node2):
    logger.info("Making VXLAN links between %s and %s" % (node1.name, node2.name))
    networkName = overlay_config.tenant_name + '-net'

    node1IP = node1.networks.get(networkName)[0]
    node1SSH = getSSHSession(node1IP, 'ubuntu', 'savi')

    node2IP = node2.networks.get(networkName)[0]
    node2SSH = getSSHSession(node2IP, 'ubuntu', 'savi')

    # Ensure OVS daemon is up and running in both nodes
    waitUntilOVSActive(node1SSH)
    waitUntilOVSActive(node2SSH)

    # STUDENTS FILL THIS PART OUT
    vni = generateVNI()
    runCommandOverSSH(node1SSH, 'sudo ovs-vsctl add-port br1 %s
            -- set interface %s type=vxlan options:remote_ip=%s options:key=%s'
            % (
                node1.name + '-' + node2.name,
                node1.name + '-' + node2.name,
                node2IP,
                vni
                ))
    runCommandOverSSH(node2SSH, 'sudo ovs-vsctl add-port br1 %s
            -- set interface %s type=vxlan options:remote_ip=%s options:key=%s'
            % (
                node2.name + '-' + node1.name,
                node2.name + '-' + node1.name,
                node1IP,
                vni))
```

This function connects two hosts using VXLAN tunneling. First, it creates a *VNI* using the helper function provided in order to ensure uniqueness of future VNIs. Then, adds a port to the previously created bridge (br1). The name of this port is in this format *(source)-(dest)*

```
def deployOverlay():
    print "In deployOverlay()"

    # Dictionaries to map switch/host names to their Nova VM objects
    createdSwitches = {}
    createdHosts = {}
    createdConnections = []

    # STUDENTS FILL THIS PART OUT
    for switch in overlay_config.topology.keys():
        switchVMObj = None

        if switch in createdSwitches:
            switchVMObj = createdSwitches[switch]
        else:
            switchVMObj = bootVM(switch)
            createdSwitches[switch] = switchVMObj
            setOverlayInterface(switchVMObj, None)

        for endpoint in overlay_config.topology[switch]:
            endpointVMObj = None
            if type(endpoint) is str:
                if endpoint + '-' + switch in createdConnections:
                    continue
                elif switch + '-' + endpoint in createdConnections:
                    continue
                else:
                    createdConnections.append(switch + '-' + endpoint)

                if endpoint in createdSwitches:
                    endpointVMObj = createdSwitches[endpoint]
                else:
                   endpointVMObj = bootVM(endpoint)
                   createdSwitches[endpoint] = endpointVMObj
                   setOverlayInterface(endpointVMObj, None)
            elif type(endpoint) is tuple:
                if endpoint[0] + '-' + switch in createdConnections:
                    continue
                elif switch + '-' + endpoint[0] in createdConnections:
                    continue
                else:
                    createdConnections.append(switch + '-' + endpoint[0])
                if endpoint in createdHosts:
                    endpointVMObj = createdHosts[endpoint[0]]
                else:
                   endpointVMObj = bootVM(endpoint[0])
                   createdHosts[endpoint] = endpointVMObj
                   setOverlayInterface(endpointVMObj, endpoint[1])

            connectNodes(switchVMObj, endpointVMObj)
```

This function first checks whether the switch/host was previously created or not. If it wasn't previously created, it creates a new VM with appropriate overlay IP addresses. Then, checks wheather the connection exists or not (in order to avoid loops). If connection didn't existed before, it creates a new connection using *connectNodes* function and stores the connection in order to avoid creating loops in the future.

```
def listOverlay():
    print "In listOverlay()"

    # STUDENTS FILL THIS PART OUT
    networkName = overlay_config.tenant_name + '-net'

    nova = getNovaClient()
    for server in nova.servers.list():
        if server.name.startswith(overlay_config.username + '-'):
            vmIP = server.networks.get(networkName)[0]
            print server.name + ':' + ' ' + server.id + ' (' + vmIP + ')'
```

This function simply iterates over all of the VMs available in this project and prints the names of the ones that begin with *(username)*-

---

```
def cleanupOverlay():
    print "In cleanupOverlay()"

    # STUDENTS FILL THIS PART OUT
```

```
nova = getNovaClient()
for server in nova.servers.list():
    if server.name.split('-')[0] == overlay_config.username:
        print server.name + ' deleted!'
        server.delete()
```

This function simply iterates over the all of the VMs available in this project and deletes all of the VMs that match the current user.