Computer Lab 2, Part I

This notebook consists of instructions, exercises and questions that form the practical part of Lab II, Part I. In this assignment, you will learn the basics of the OpenStack Python APIs that can be used to interact directly with the IaaS services Nova (compute) and Swift (Object Store). Please prepare your solution and answers to questions directly in this notebook, and export it to PDF. Upload that PDF as to the student portal to complete Part I of the Lab.

In [192]:
```
import os
import sys
import time
import swiftclient.client
import paramiko
import subprocess

OS_TENANT_NAME="ACC-Course"
OS_PROJECT_NAME="ACC-Course"
OS_USERNAME="shjo0681"
OS_AUTH_URL='http://smog.uppmax.uu.se:5000/v2.0'
OS_PASSWORD='xxx'

os.environ['OS_TENANT_NAME'] = OS_TENANT_NAME
os.environ['OS_PROJECT_NAME'] = OS_PROJECT_NAME
os.environ['OS_USERNAME'] = OS_USERNAME
os.environ['OS_AUTH_URL'] = OS_AUTH_URL
os.environ['OS_PASSWORD'] = OS_PASSWORD
```

To establish a client connection, we will need to pass a dictionary with information about the tenant, user, credentials and the API Identity endpoint. Here, I have sourced the "openrc.sh file" obtained from the Horizon dashboard in the underlying shell prior to starting the notebook. Hence, in order to actually run the code below, you would need to do the same with your own credentials.

In [193]:
```
config = {'user':'shjo0681',
          'key':'xxx',
          'tenant_name':"ACC-Course",
          'authurl':"http://smog.uppmax.uu.se:5000/v2.0"}
```

First, we obtain a client connection to Swift (we are usign the v2 APIs)

In [194]:
```
conn = swiftclient.client.Connection(auth_version=2, **config)
```

In [195]:
```
# Create a container, use a UUID to make sure it has a globally unique name
import uuid
bucket_name = "lab2_{0}".format(str(uuid.uuid4()))
conn.put_container(bucket_name)

#Saving bucket name for deletion later.
bucket_reference = bucket_name
```

Question 1:

What does it mean that the object store has a global, flat namespace? What is the practical consequence for you when using it?

In [196]:
```
# List containers
(response, bucket_list) = conn.get_account()
for bucket in bucket_list:
    print bucket['name']
```

```
lab2_mm_d138c2b1-99d1-4335-a356-5d913e0b80d8
lab2_mm_ee98f366-8a4c-4628-8423-986891d3e609
lab2_puan
ljoni2138
lufr
lufr2071
noaabukk
ruul_bucket_ah
ryman
testContainer
testContainer_Saim
testContainerrrr
test_bucket
testcontainer2
testcontainerAndrew
testcontainerKalle
testcontainerS
testcontainerr
tweets
```

In [197]:
```
# Put an object in the container
print 'Putting crap into bucket: \n' + bucket_name
object_id = conn.put_object(bucket_name, "test_object", "Hi Swift")
```

```
Putting crap into bucket:
lab2_492efbef-b7ef-43e2-a598-d258123fbb26
```

Excercise 1:

Try to measure the speed with which you can put and get objects to and from Swift usign the API. Conduct your experiment several times to gather statistic and plot a) A estimated distribution of the time taken (in wall clock) to put and read an object of size 10MB in your swift container and b) vary the size of the object from 10kB to 100MB and plot the put and get throughput (in MB/s) times as a function of object size (for the smaller data sizes, you might need to repeat the experiment many times and obtain a statistical average). Include the resulting graphs and a description of your experiment in the report.

In [199]:

```
#dd if=/dev/zero of=1MB.img count=1 bs=1M makes a file
def getTime():
    return time.time()
    #return int(round(time.time() * 1000))
def getTimeTaken(start):
    return getTime() - start

#Get a target container
bucket_name = "lab2Popcorn"#.format(str(uuid.uuid4()))
print "Created a container named: " + bucket_name
bucket_exists = 0
(response, bucket_list) = conn.get_account()
for bucket in bucket_list:
    if(bucket['name'] == bucket_name):
        bucket_exists = 1
if(bucket_exists == 0):
    conn.put_container(bucket_name)

def performMeasurement(fileName):
    with open(fileName, 'r') as temp_file:
        startTimer = getTime()
        conn.put_object(bucket_name, fileName, contents=temp_file.read(), c
        conn.get_object(bucket_name, fileName)
        return getTimeTaken(startTimer)

def tenTimesAverage(fileName):
    time = 10
    totalTime = 0
    while time>0:
        totalTime = totalTime + performMeasurement(fileName)
        time = time -1
    return (totalTime / 10)

print 'Starting measurments, this might take some time'
time10MB = tenTimesAverage('10MB.img')
time50MB = tenTimesAverage('50MB.img')
print 'About halfway through measurments.'
time100MB = tenTimesAverage('100MB.img')
time1MB = tenTimesAverage('1MB.img')
time10KB = tenTimesAverage('10KB.img')
time50KB = tenTimesAverage('50KB.img')
print 'Measurments done, preparing the plot.'

# Implement you solution here. Hint, the following command
%pylab inline

def changeToMBperSecond(time, itemSize):
    mbItemSize = itemSize
    timeInSeconds = time
    return mbItemSize / timeInSeconds

time10KB = changeToMBperSecond(time10KB, 0.1)
time50KB = changeToMBperSecond(time50KB, 0.5)
time1MB = changeToMBperSecond(time1MB, 1)
```

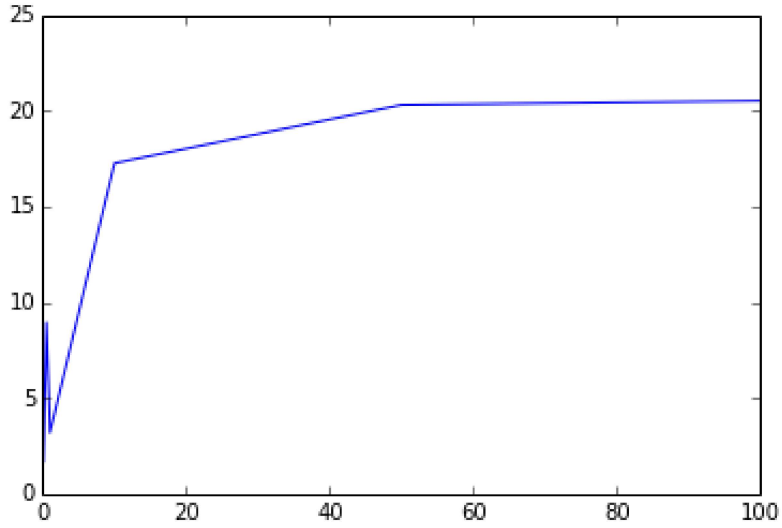
```
time1MB = changeToMBperSecond(time1MB, 1)
time10MB = changeToMBperSecond(time10MB, 10)
time50MB = changeToMBperSecond(time50MB, 50)
time100MB = changeToMBperSecond(time100MB, 100)

xaxis = [0.1,0.5,1,10,50,100]
yaxis = [time10KB, time50KB, time1MB,time10MB,time50MB,time100MB]

# will  make matplotlib/pylab available and plots will be displayed directl
plt.plot(xaxis,yaxis)
```

```
Created a container named: lab2Popcorn
Starting measurments, this might take some time
About halfway through measurments.
Measurments done, preparing the plot.
Populating the interactive namespace from numpy and matplotlib
```

Out[199]: `[<matplotlib.lines.Line2D at 0x7faa888385d0>]`



Excercise 2:

In [200]:
```
# Obtain a list of all the object names in your container.
(response, stuff_list) = conn.get_container(bucket_name)
for stuff in stuff_list:
    print stuff['name']
```

```
100MB.img
10KB.img
10MB.img
1MB.img
50KB.img
50MB.img
```

In the cell below, we obtain a client connection to the Nova endpoint. It can be used to for example start, stop and terminate instances.

In [201]:
```
config = {'user':os.environ['OS_USERNAME'],
          'key':os.environ['OS_PASSWORD'],
          'tenant_name':os.environ['OS_TENANT_NAME'],
          'authurl':os.environ['OS_AUTH_URL']}
from novaclient.client import Client
nc = Client('2',**config)
```

Excercise 3:

Boot a new instance (hint, look client.server in the API docs) with flavor 'm1.medium' (remember to provide an ssh-key so that you can access it later). In booting the instance, use the mechanism of 'user_data' (learn about this in the openstack and 'cloud-init' documentations) to provide a startup-script to 1. Update the instance, 2. install 'git', 'cowsay' and 'flask'.

In [202]:

```
# Use paramiko to access your instance and, using ssh, start the cowsay ser
# usinf the same command as in Task 4, lab 1.

# New nova client since other API doesn't work
from novaclient import client as novaclient
creds = {}
creds['username'] = os.environ['OS_USERNAME']
creds['api_key'] = os.environ['OS_PASSWORD']
creds['auth_url'] = os.environ['OS_AUTH_URL']
creds['project_id'] = os.environ['OS_PROJECT_NAME']
nova = novaclient.Client('2', **creds)

def ServerExists(name):
    serverExists = False
    listServers = nova.servers.list(detailed=True)
    for server in listServers:
        if(server.name == name):
            serverExists = True
    return serverExists

def setImage(preferedImage):
    imageExists = False
    listImages = nova.images.list(detailed=True)
    for image in listImages:
        if(image.name == preferedImage):
            return preferedImage
    return listImages[0].name

def getFreeFloatingIP():
    listFloatingIps = nova.floating_ips.list()
    for ip in listFloatingIps:
        if(ip.fixed_ip == None):
            return ip
    return nova.floating_ips.create()

serverName = 'CowSaysPopcorn'
flavor = nova.flavors.find(name='m1.large')
image = nova.images.find(name=setImage('readyConfig'))
image = '99f4625e-f425-472d-8e21-7aa9c3db1c3e'
connected = True

#rm -rf /var/lib/cloud/*
user_data_stuff = ('#!/bin/bash \n'
                   'sudo apt-get update -y && sudo apt-get install python-
                   'sudo apt-get install cowsay -y \n pip install Flask \n
#user_data_stuff = '#!/bin/bash\nsudo apt-get install python-pip\necho Upda

instance = nova.servers.create(name=serverName, flavor=flavor, image=image,
print instance
#instance = nova.servers.find(name=serverName)
time.sleep(10)
floatingIp = getFreeFloatingIP()
nova.servers.add_floating_ip(instance, floatingIp)
```

```
print floatingIp
time.sleep(3)
```

```
<Server: CowSaysPopcorn>
<FloatingIP fixed_ip=None, id=262021d3-5ad1-4e5b-bf9c-135c97542b1b, instan
ce_id=None, ip=130.238.29.37, pool=ext-net>
```

In [203]:
```
ssh = paramiko.SSHClient()
ssh.load_system_host_keys()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())

ssh.connect(floatingIp.ip, username='ubuntu', key_filename='cowsaypopcornKe
com = 'sudo git clone https://www.github.com/TDB-UU/csaas'
stdin, stdout, stderr = ssh.exec_command(com)

print 'SSH connection successful'
stderr = stderr.readlines()
stdout = stdout.readlines()
print stdout
print stderr

com = 'python /home/ubuntu/csaas/cowsay/app.py'
stdin, stdout, stderr = ssh.exec_command(com)
print stdout
print stderr

ssh.close()
print 'closing connection'
time.sleep(3)
#This code is tested and working.
print com
print floatingIp.ip
command = 'curl -i http://' + floatingIp.ip + ':5000/cowsay/api/v1.0/saysom
print "Running: " + command
p = subprocess.Popen(command, stdout=subprocess.PIPE, shell=True)
(output, err) = p.communicate()
print "Output: \n", output

# Make a request to the cowsay REST API and display the response inline in
```

```
SSH connection successful
[]
[u'sudo: unable to resolve host cowsayspopcorn\n', u"Cloning into 'csaa
s'...\n"]
<paramiko.ChannelFile from <paramiko.Channel 1 (open) window=2097152 -> <p
aramiko.Transport at 0x888bd350L (cipher aes128-ctr, 128 bits) (active; 1
open channel(s))>>>
<paramiko.ChannelFile from <paramiko.Channel 1 (open) window=2097152 -> <p
aramiko.Transport at 0x888bd350L (cipher aes128-ctr, 128 bits) (active; 1
open channel(s))>>>
closing connection
python /home/ubuntu/csaas/cowsay/app.py
130.238.29.37
Running: curl -i http://130.238.29.37:5000/cowsay/api/v1.0/saysomething
Output:
HTTP/1.0 200 OK
Content-Type: text/html; charset=utf-8
Content-Length: 175
Server: Werkzeug/0.10.4 Python/2.7.6
```

```
Date: Thu, 01 Oct 2015 16:11:03 GMT


 _________________
< Hello student >
 ---------------
        \   ^__^
         \  (oo)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

Question 2:

The above excersise showed a low-level way of 'contextualization' using user data 'cloud-init'. Do some reserach online and discuss alternative tools and techniques for contextualization of your VMs. Discuss the difference between instance meta-data and user-data. Some links to get you started:

http://docs.openstack.org/user-guide/cli_provide_user_data_to_instances.html (http://docs.openstack.org/user-guide/cli_provide_user_data_to_instances.html) http://cernvm.cern.ch/portal/contextualisation (http://cernvm.cern.ch/portal/contextualisation) https://cloudinit.readthedocs.org/en/latest/ (https://cloudinit.readthedocs.org/en/latest/)

Aim for the equivalent of ~1/2 page of an A4 paper, 12pt font, 2cm margins.

Excersise 4:

Use the Swift and Nova APIs to terminate your instance, to delete all the objects from your bucket, and then finally to delete the container.

In [204]:
```
# Clean up container in Swift
print bucket_reference
print bucket_name
print conn

for stuff in conn.get_container(bucket_reference)[1]:
    conn.delete_object(bucket_reference, stuff['name'])

for stuff in conn.get_container(bucket_name)[1]:
    conn.delete_object(bucket_name, stuff['name'])
#print conn.delete_container(bucket_reference)
print conn.delete_container(bucket_name)
```

```
lab2_492efbef-b7ef-43e2-a598-d258123fbb26
lab2Popcorn
<swiftclient.client.Connection object at 0x7faa88a8bf10>
None
```

In [205]:
```
# Terminate all your running instances
instance.delete()
```

In [ ]:
```

```