Assignment 1 - A simple service the hard way

The task is to create a simple web service that display the number of visitors since deployment. It need not be unique visits, but rather a counter that is increased every time the page is loaded. To achieve this, you need to learn how to launch virtual machines, create virtual networks and routers, configure storage, and manage credentials. The assignment consists of two steps. First you should do this using graphical user interface Horizon and then do the same using the Terraform orchestration tool.

By hand using Horizon
You should create a simple virtual infrastructure that consists of one or more virtual machines that are connected to the same subnet and that access internet via a virtual router.

- Create a virtual network and a subnet.
- Create a virtual router.
- Upload a set of SSH keys.
- Create a virtual machine, connect it to the network you created above, and use the SSH keys you just added.
- Associate a floating IP to the virtual machine.
- Configure the security groups (firewall) to allow for ssh access (port 22) and www access (use the port that is exposed by your server).
- login to the virtual machine and make sure you can access internet.
- Finally, create a storage container in object store

Now that we have created the infrastructure up and running, it's time to write an application. You are tasked to write a web service that keeps count of visits. The front-end should be scalable, i.e. it should be possible to add and remove servers without losing count. Therefore, the counter should be stored in persistent storage and in this assignment, use the object store. The Swift object store allows you to upload a binary object, for example a file that contains a single number. This may not be the most efficient way to store a counter, but hey nobody really does visitor counters anymore so don’t worry too much about efficiency.
However, what you should worry about is how you handle credentials. Accessing the object store will require the calling VM to authenticate itself. How do you get the password there in a secure way?

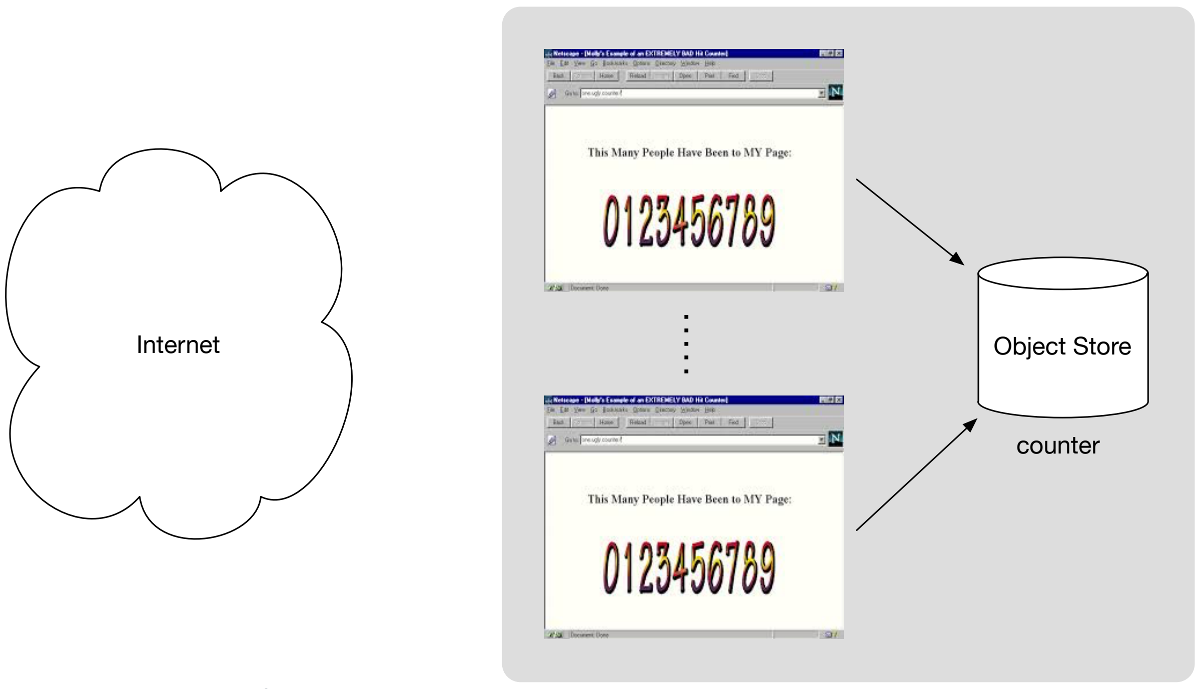
Internet
This Many People Have Been to MY Page:
0123456789
This Many People Have Been to MY Page:
0123456789
Object Store
counter

Figure 1. A high-level view of the visitor count service.

Start by developing a very simple web server on your local machine. We suggest you write it in Python and use the Flask library, but to each her own. The web server should consist of a static part, which displays the message "You are visitor:", followed by a dynamic part which is the visit counter. This means that the web server needs to access the object store to read and update the counter.

There exist a set of Python APIs for accessing OpenStack and they are automatically installed when you install the OpenStack command-line tools. The first step is then to do just that. We suggest you use virtualenv to simplify installation of the OpenStack tools. This will help you recreate the same environment on a server in the cloud later.

Once you have your service working locally, you should transfer it to a server in the cloud. When you got a working service, you can save that as an image (do a snapshot). Now, should be able to create new VMs from that same server image and they should connect to the same counter object. The counter value displayed should work as expected, monotonically increasing (don't worry about race conditions). You can use the `cloud-init` feature to run a script when a VM is booted. This can be used to start the web server and pass parameters.

Scripting using Terraform

At this point you have created the infrastructure by hand using Horizon. Now, you shall do the same using Terraform. The only things that should be kept from the first part is the server image.

Start by installing Terraform and work through the getting started guide found here https://learn.hashicorp.com/terraform/getting-started/install.html. You will find an interactive tutorial at katakoda (don't worry if you don't understand the Docker part. We will come to that later.).

Write your Terraform application as a set of *.tf-files. Demonstrate how you can add and remove web servers and that the counter works as expected. Provide an output-file that contains the public IP-addresses of the current front-ends.
Also provide a way to clean your account.

Example code for simple server

```
from swiftclient.service import SwiftService, SwiftError,
SwiftUploadObject
from flask import Flask, Response
import os

def download_counter(swift, container, object_name='counter'):
    ...

def update_counter(object_name='counter'):
    ...

def upload_counter(swift, container, object_name='counter'):
    ...

def increment_counter(swift, container, object_name='counter'):
    download_counter(swift, container, object_name)
    value = update_counter(object_name)
    upload_counter(swift, container, object_name)
    return value

container = ...
app = Flask(__name__)

@app.route('/')
def handle_get():
    with SwiftService() as swift:
        try:
            return Response('{"value":%d}' % increment_counter(swift,
container), mimetype='application/json')
        except SwiftError as e:
            logger.error(e.value)


if __name__ == "__main__":
    app.run()
```

The above application assumes that the proper environment variables are set, i.e. the ones from the .rc-file that is downloaded from your project.

The cloud servers have no passwords so you need to login using the ssh-key you provided when creating the VMs.

```
      ssh -i mykey.key ubuntu@123.123.123.123
```

It's good practice when you login in to a machine for the first time, to make sure to update it (otherwise the repositories are not correct and you will not be able to install packages). For the server above, you should need to do the following:

```
sudo apt-get update
sudo apt-get install git
sudo apt install python-pip
```

ADD A NOTE ON VIRTUALENV?

```
pip install Flask==1.1.1
pip install python-swiftclient==3.8.0
pip install python-keystoneclient
```