School of Engineering and Technology
University of Washington – Tacoma
Instructor: Wes Lloyd

Assignment 0

Version 0.12

Cloud Computing Infrastructure Tutorial

Due Date: Sunday January 31st, 2021 @ 11:59 pm, tentative

Objective

The purpose of assignment 0 is to establish AWS accounts, and gain experience with technologies used to provide distributed computing infrastructure to support future TCSS 558 programming assignments.

For information regarding creating an AWS account for use in TCSS 558 course projects and beyond, please refer to Tutorial 0 in the TCSS 562 Cloud Computing course:

http://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/tcss562_f2020_aws_getting_started.pdf

Where possible, it is recommended to create a standard AWS account. This account does not have restrictions on the use of EC2 spot instances which are virtual machines (VMs) available at a reduced cost, approximately ~ 1/4 of full price. These VMs can be used for TCSS 558 projects.

FOR STANDARD AWS ACCOUNTS, EMAIL THE INSTRUCTOR FOR AWS CREDITS:

If creating a standard AWS account backed by a credit card, once the account is created, contact the instructor by email to request a credit coupon code. Use **"AWS CREDIT REQUEST"** as the subject. Identify the email address you've used to create the AWS account.

The instructor will email a credit code which can be used to provide AWS Cloud Credits.

Assignment 0 provides a tutorial on the use of Cloud Computing Infrastructure. Specifically, assignment 0 walks through the use of EC2 instances, docker, docker-machine, and haproxy for load balancing.

We will create virtual machines, known as elastic compute cloud (EC2) instances to host individual nodes of our distributed systems. Docker containers are used to deploy our application code to create the nodes of our distributed system. These containers are created on VMs. To support working with VMs to host our distributed applications, students may optionally harness tools such as Docker-Machine and/or Kubernetes to automatically create and configure containers and/or VMs.

Use of a Linux environment is recommended for assignment 0.

Windows and Mac users can install Oracle Virtual Box to create virtual machines under Windows 10, and then install an Ubuntu 20.04 virtual machine.

For Windows 10 users, there is an Ubuntu "App" that can be installed onto Windows 10 directly. The use of the Ubuntu App is not recommended due to potential compatibility issues or lack of features.

Windows Oracle Virtual Box & Ubuntu VM instructions:

Oracle VirtualBox can be downloaded here:
https://www.virtualbox.org/

The Ubuntu 20.04 LTS ISO file can be downloaded here:
https://releases.ubuntu.com/20.04/ubuntu-20.04.1-desktop-amd64.iso (2.6 GB)

The LIVE installation media is smaller. This ISO file is "bootable" so the system will start up using this image. You can then install Ubuntu over the internet using this ISO file where only the necessary components you select will be downloaded:
https://releases.ubuntu.com/20.04/ubuntu-20.04.1-live-server-amd64.iso (914 MB)

Another approach is to obtain Ubuntu using a bit torrent stream by downloading:
https://releases.ubuntu.com/20.04/ubuntu-20.04.1-desktop-amd64.iso.torrent (208K)

If unfamiliar with BitTorrent, a popular mechanism for downloading and sharing of large files read the wikipedia page here:
https://en.wikipedia.org/wiki/BitTorrent

If wanting additional help on how to install Oracle Virtual Box, try searching the internet for installation instructions using search engines such as bing or google, or try finding a video at video.google.com.

Here are two helpful videos:

Introduction to Oracle VirtualBox for creating Virtual Machines:
https://www.youtube.com/watch?v=sB_5fqiysi4

Installing Ubuntu 20.04 on Windows 10 Oracle VirtualBox
https://youtu.be/x3Zpe1rIPFE

And here are instructions for installing Ubuntu 20.04 on Oracle VirtualBox:

Generic instructions for a Windows/Mac/Linux Host:
https://fossbytes.com/how-to-install-ubuntu-20-04-lts-virtualbox-windows-mac-linux/

Linux-specific instructions for a Linux Host:
https://linuxconfig.org/how-to-install-ubuntu-20-04-on-virtualbox

Instructions including how to install Guest Additions:
https://linuxhint.com/install_ubuntu_virtualbox_2004/

When working with Virtual Machines, the base operating system (e.g. Windows 10) on your laptop that hosts the virtual machine is called the host operating system. The operating system used by the VM is called the guest operating system.

A common configuration used in TCSS 558 is:
Guest operating system (VM) = Ubuntu 20.04 LTS
Host operating system = Windows 10 or Mac OSX

Once your VM is installed, it is highly recommended to install the Ubuntu 20.04 VirtualBox "Guest Additions". These guest additions provide important features such as ***sharing clipboards*** between the host and the guest, as well as ***file system sharing***, and ***mouse pointer integration***. For instructions on installing the guest additions here is an article:
https://linuxconfig.org/virtualbox-install-guest-additions-on-ubuntu-20-04-lts-focal-fossa

Please do yourself a favor, and do not go the entire quarter without installing the guest additions.

Task 1 – AWS account setup

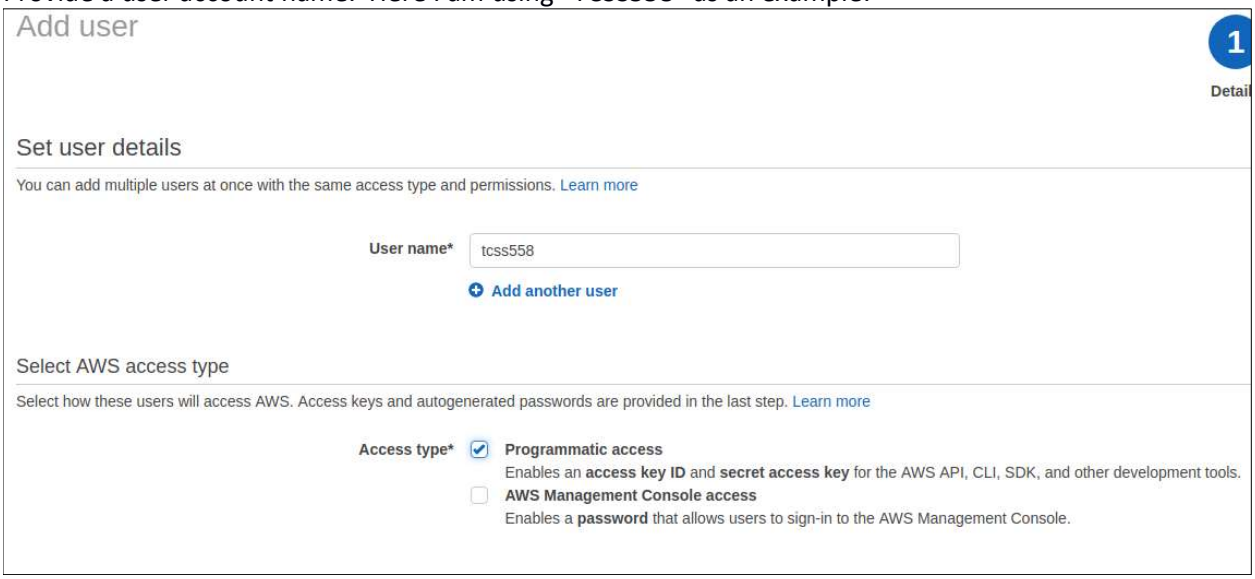
Once having access to AWS, task #1 involves creating AWS account credentials to work with Docker-Machine, if you have not already done so.

If you're using an AWS Educate starter account: Obtain account credentials for your AWS starter account by going to the Vocareum user interface in your web browser, and click the **blue button** labeled **"Account Details".** This should provide access the account credentials including an ***access key*** and a ***secret key***. Copy these and store in a safe and secure place, and then **SKIP to Task 2 in the assignment!**

If not using an AWS starter account, from the AWS services home page, locate the "IAM" Identity Access Management link, and select it:

Once in the IAM dashboard, on the left hand-side select "Users" → :

Provide a user account name. Here I am using “TCSS558” as an example:



Be sure to select the “Programmatic access” checkbox.

Then click the “Next: Permissions” button…

For simplicity, you can simply select the button:

Using the search box, search, find, and select using the checkbox the following policy:

* AmazonEC2FullAccess

If you plan to use this user account to explore additional Amazon’s services, then admin access can be added (not required):

* AdministratorAccess

This will allow you, via the CLI, to explore and do just about everything with this AWS account.

Now click the “Next: Review” button, and then select “Create user”.

You'll now see a screen with an Access key ID (grayed out below), and a Secret access key. You can copy both the Access key, and the secret access key to a safe place, or alternatively, click the "Download .csv" button to download a file containing this information.

Once you've downloaded these keys, be sure to **never** publish these key values in a source code repository such as github where your account credentials could be exposed. **Protect these keys as if they were your credit card or wallet!**

Task 2 – Working with Docker, creating Dockerfile for Apache Tomcat

Next, launch a virtual machine on Amazon to support working with Docker/Docker-Machine. You will want to have access to a computer with the ssh/sftp tools. It is best to have access to a local computer with Ubuntu installed either natively, or on Oracle Virtualbox. It is possible to download putty, an "SSH" client and also an "SFTP" client, for Windows, but not recommended.

Recommend use of us-east-2 Ohio AWS Region:

Choose the AWS "region" that you'll work in. The recommended option is "US East (Ohio)" known as "us-east-2" via the CLI. The Ohio region is newer, has less traffic, and has been seen as ***less expensive*** than others for EC2 spot instances.

The region can be set using the dropdown in the upper-right hand corner. Selecting the region configures the entire AWS console to operate in that region as shown to the right –---------->

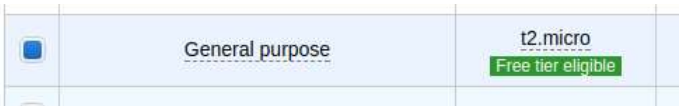
For assignment 0, we will begin testing by using a "t2.micro" instance. Users are allowed up to 750-hours each month of instance time for FREE using the t2.micro type.

From the AWS menu, under Compute services, select "EC2":

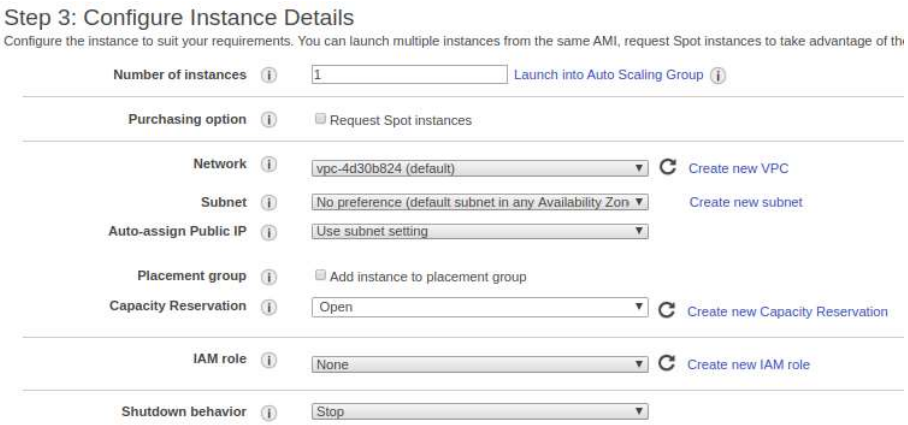
Next, click "Instances" on the left-hand side menu.
Then click the blue "Launch Instance" button:

Select Ubuntu:

Specify t2.micro as the instance type, and click the “Next: Configure Instance Details” button,



Next, specify the following instance details:



Network: choose “(default)” for the Virtual Private Cloud (VPC).
Subnet: choose an availability zone such as us-east-2b
Auto-assign Public IP: choose “Use subnet setting (Enable)”. This will provide a public IP address to enable connecting to your instance.
Shutdown behavior: Choose “Stop”

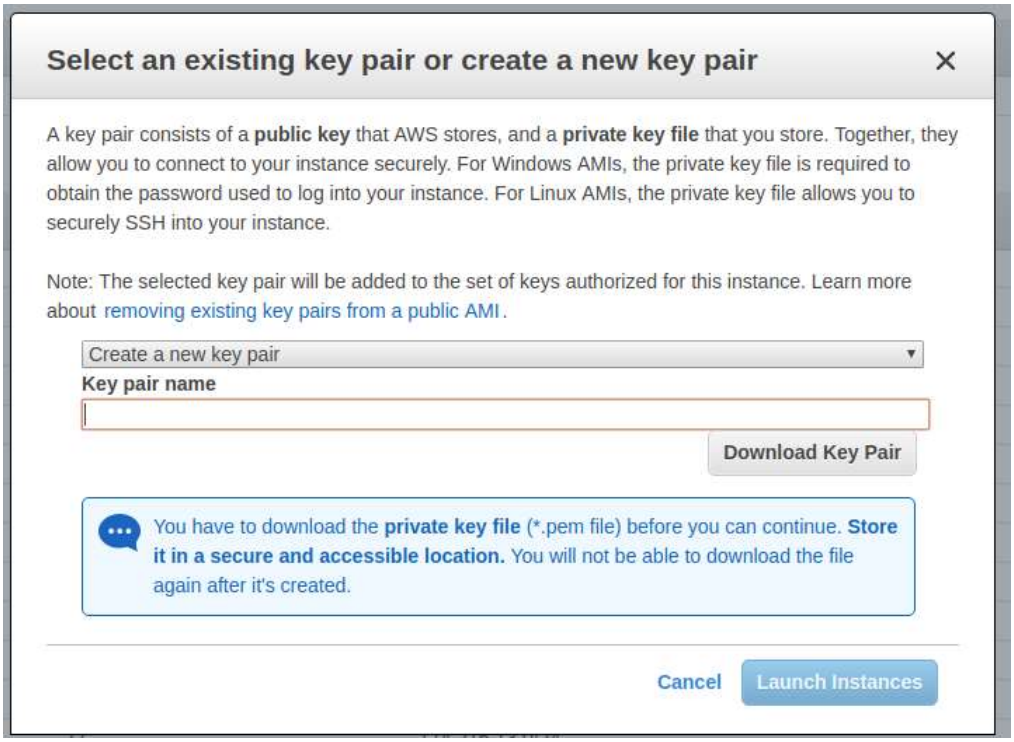
Next, click “Next: Add Storage”.
Then, keep defaults and click “Next: Add Tags”.
Then, keep defaults and click “Next: Configure Security Group”.
Choose the option:

And then mark the option for “default VPC security group”.

As we go along, apply all security changes to the default security group for your default VPC. This way rule changes will persist as you come back to AWS for future work sessions.
Then click “Review and Launch”.

Review the details and if everything looks ok, click “Launch”.

The very first time you'll be prompted to create a new RSA private/public keypair to enable logging into your instance.



The instance should launch and be visible by clicking "Instances" on the left-hand side of the EC2 Dashboard. Locate the IPv4 Public IP:

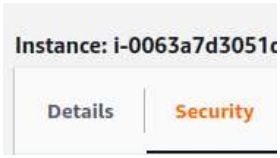
Throughout the tutorial, Linux commands are prefaced with the "$". **Comments are prefaced with a "#".**

First, from the Linux CLI change permissions on your keyfile:

```
$chmod 0600 <key_file_name>.pem
```

Before you can SSH into the instance, the default security group used by your instance must be modified to allow SSH (port 22) access from your computer.

In the Amazon management console., under ec2 instances, click on your instance, and click on the **security tab:**



In the details section, click on your security group ID to edit the settings:

Click the “Inbound rules” tab, and then the “Edit inbound rules” button.
Scroll down and click the “Add Rule” button at the bottom of the dialog box:

Add a “SSH” Rule with the following settings:
Protocol = TCP
Port Range = *will automatically be set to 22*
Source = My IP

Add a “ICMP” Rule with the following settings (to support “ping”):
Protocol = All ICMP – IPv4
Port Range = *will automatically be set to All*
Source = My IP

Then click the “Save rules” button to save the security change.

Then connect using ssh:

```
$ssh -i <key_file_name>.pem ubuntu@<IPv4 Public IP>
```

Say yes, when the following message is displayed:

```
The authenticity of host '107.21.193.159 (107.21.193.159)' can't be
established.
ECDSA key fingerprint is SHA256:0cy2eP8Q15zmBThAqTq9z1TwO0+MS0ldKi1SmPZhkE0.
Are you sure you want to continue connecting (yes/no)? Yes
```

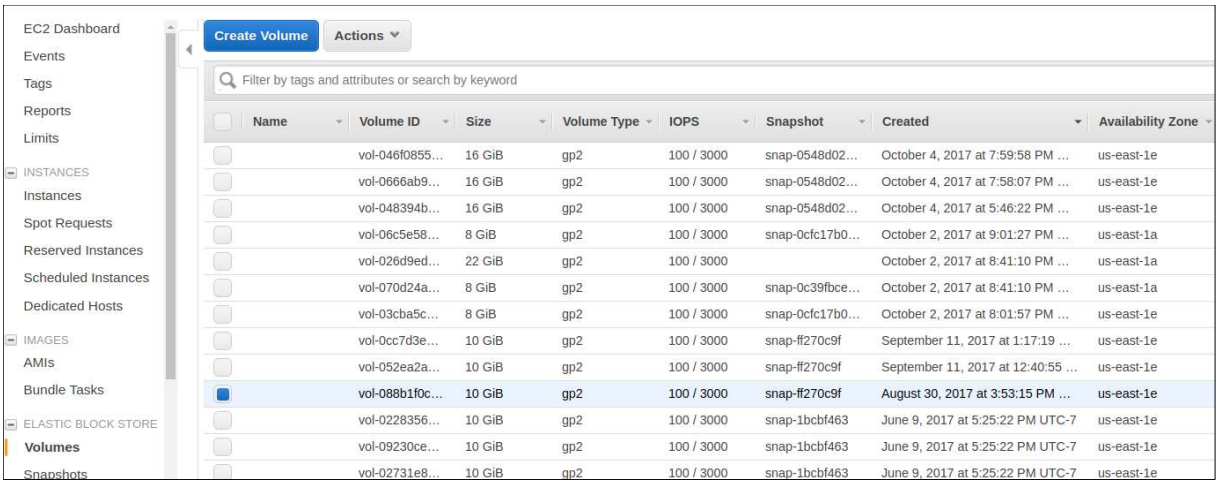
Note, the actual IP address will be different than “107.21.193.159”.

Linux tracks every machine you ever ssh to. The very first time it hashes the public key and places it into a file at /home/<user_user_id>/.ssh/known_hosts

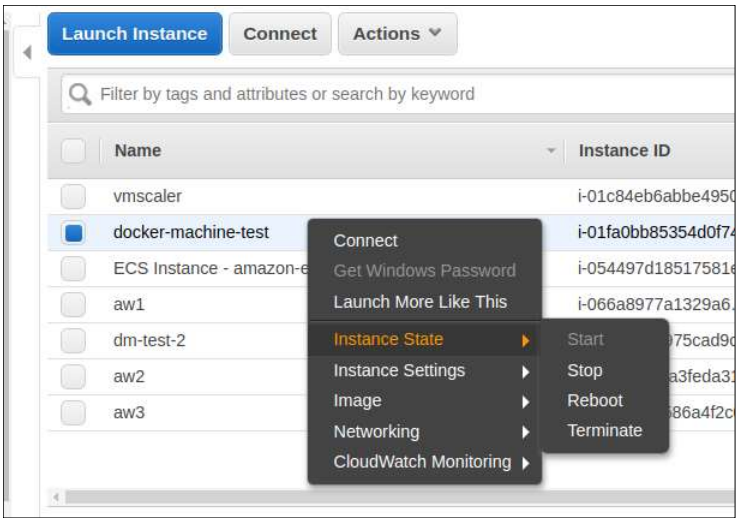
The idea is when you reconnect to the VM again, there is a possibility that someone masquerades as the VM. To prevent someone from masquerading as the VM you’re trying to connect to, ssh tracks the identity of each host and alerts the user **every time** there is a change. Sometimes the changes are expected, such as when you launch a new VM to replace an old one. The idea is to help notify the user if the VM’s identity changes unexpectedly.

Stopping, and backing up your VM on Amazon:

By default, the t2.micro is an “EBS-backed” instance. The t2.micro instances make use of remotely hosted elastic blockstore virtual hard disks for their “/root” volume. “EBS-backed” instances can be paused at any time. This allows the VM to be stopped, and resumed later. Billing is paused, but storage charges for the EBS disk are ongoing 24/7. New AWS users are allowed 30GB of free EBS disk space in the 1st year. Beyond this, the price for storage is 10 cents per GB, per month, for standard “GP2-General Purpose 2” EBS storage. A second 30GB (total of 60GB) will cost $36/year in credits. In the console, any volumes listed under “Elastic Block Store | Volumes” will count towards this 30GB quota.

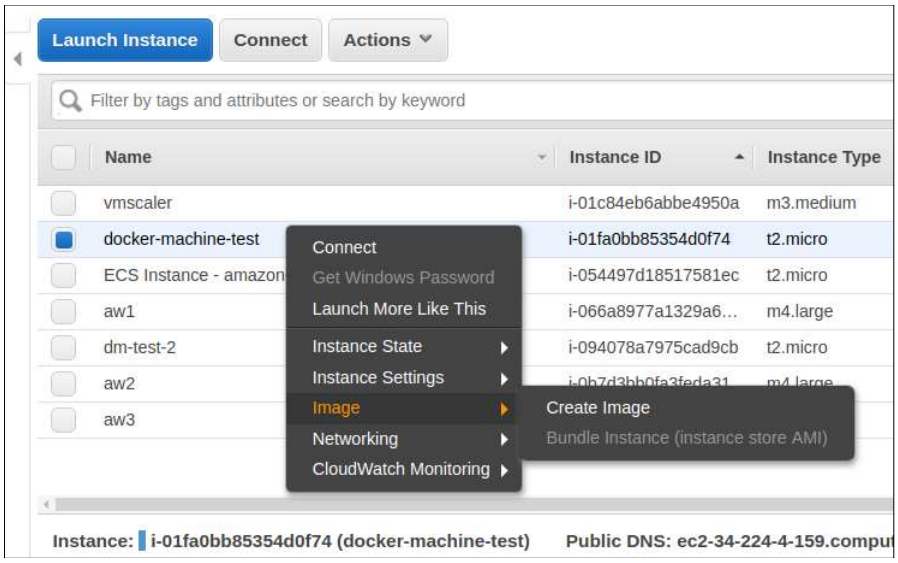


Snapshots, under “Elastic Block Store” represent compressed copies of EBS volumes that are stored using Amazon Simple Storage Service (S3), aka blob storage. Standard pricing for S3 storage is 2.3 cents per month per GB. If not using a VM for a considerable time, a cost effective way to preserve the data is to “**snapshot**” the EBS volume and create an AMI. Then delete the VM and live EBS volume.



To “stop” your instance right-click on the row in the “Instances” view, select “Instance state”, and then “stop”. You may later resume the instance by selecting “start”. When restarting your instance, your public IPv4 address may be reassigned.

An image can be created by right-clicking on the instance row, and selecting “Image” and “Create Image”. This will temporarily shutdown your instance to create the image. Once the image has been created, the instance is restored to its online state. New images will be listed under “Images | AMIs” on the left-handside of the EC2 console. Sorting by Creation Date makes it easy to locate newly created images.



As you work throughout the course projects in TCSS 558, starting from the virtual machine image can help jump start development and testing of future projects.

Next, let's install Docker on this VM.

highlight the full text below including all spaces and linefeeds/newlines, then copy-and-paste directly to the VM. You may break this into separate commands by copying-and-pasting individual command separated by spaces to more carefully see what is happening:

IT IS BEST TO USE THE WORD DOC FOR COPY & PASTE, NOT THE PDF!

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key
add -
sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# refresh sources
sudo apt-get update

# install packages
apt-cache policy docker-ce

sudo apt-get install -y docker-ce

#verify that docker is running
sudo systemctl status docker
#this is an equivalent command which is easier to remember
sudo service docker status
```

The "Docker Application Container Engine" should show as running.

When working with Docker directly on your local VM, we will preface docker commands with "sudo", so the commands run as the superuser.

Create a docker image for Apache Tomcat

The "Docker Hub" is a public repository of docker images. Many public images are provided which include installations of many different software packages.
The "sudo docker search" command enables searching the repository to look for images.

Let's start by downloading the "ubuntu" docker container image:
Note that docker commands are prefaced as "sudo".
They must be run as superuser.

```
sudo docker pull ubuntu
```

Verify that the image was downloaded by viewing local images:

```
sudo docker images -a
```

Next, make a local directory to store files which describe a new docker image.

```
mkdir docker_tomcat
cd docker_tomcat
```

Now, download the Java application that we will deploy into the Docker container:

```
wget http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a0/fibo.war
```

Using a text editor such as vi, vim, pico, or nano, edit the file "Dockerfile" to describe a new Docker image based on ubuntu that will install the Apache tomcat webserver:

```
nano Dockerfile
```

```
# Apache Tomcat Dockerfile contents:
FROM ubuntu
RUN apt-get update
RUN apt-get install -y tomcat9
COPY fibo.war /usr/share/tomcat9/webapps/
COPY entrypoint_tomcat.sh /
RUN mkdir /usr/share/tomcat9/logs
RUN mkdir /usr/share/tomcat9/temp
RUN ln -s /var/lib/tomcat9/conf /usr/share/tomcat9
ENTRYPOINT ["/entrypoint_tomcat.sh"]
```

Next, create a script called "entrypoint_tomcat.sh" under your docker_tomcat directory as follows:

```
#!/bin/bash
# tomcat daemon - runs container continually until tomcat exits
/usr/share/tomcat9/bin/startup.sh
echo "tomcat daemon up..."
sleep 3
while :
do
```

```
tomcatstatus=`ps aux | grep tomcat9 | grep java`
if [ -z "$tomcatstatus" ]
then
  #exit
  echo "tomcat down"
fi
 sleep 1
done
```

You'll need to change permissions on this file.
Give the owner execute permission:
`chmod u+x entrypoint_tomcat.sh`

Next, build the docker container:
`sudo docker build -t tomcat1 .`

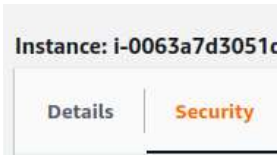
Check that the docker image was build locally:
`sudo docker images`

Next launch the container as follows:
`sudo docker run -p 8080:8080 -d --rm tomcat1`

Check that the container is up
`sudo docker ps -a`

Now, you'll need to open port 8080 in the default security group in the Amazon management console.

In the Amazon management console., under ec2 instances, click on your instance, and click on the **security tab:**



In the details section, click on your security group ID to edit the settings:

Click the "Inbound rules" tab, and then the "Edit inbound rules" button.
Scroll down and click the "Add Rule" button at the bottom of the dialog box:

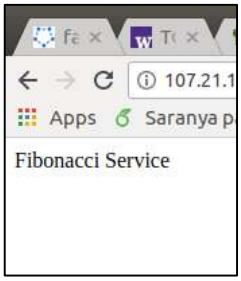
Add a "Custom TCP" rule with the following settings:
Protocol = TCP
Port Range = 8080
Source = My IP

Then click the "Save rules" button to save the security change.

Now, using your browser, point at the http GET endpoint for the web application:

http://<IPv4 Public IP of instance>:8080/fibo/fibonacci

You should see a web page as follows:



Now using your **laptop or desktop** Ubuntu environment, test the fibonacci web service deployed using the container on your EC2 instance with the testFibPar.sh script.

Download the script here using "wget":
http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a0/testFibPar.sh
This script uses a Linux utility known as GNU parallel to coordinate separate threads to support parallel client sessions with Apache Tomcat.

The testFibPar.sh script as written assumes connectivity between the client and server, and assumes everything is working correctly. In many cases, however, there are errors.

To address connectivity, security group, and application server errors, download the testFibService.sh script. The testFibService.sh script performs a NETWORK, TRANSPORT, and APPLICATION layer test confirming that the client can reach the HTTP REST Fibonacci service and successfully receive results back. **Do not submit your assignment results from testFibPar.sh before confirming connectivity to EVERY SERVER.** For multi-server deployments, each endpoint **must** be tested separately. When/if a server fails, use the "docker ps -a" command to identify containers that have failed, and restart them. Also resolve issues in your security group or haproxy.cfg file accordingly. Carefully study the testFibService.sh script if you're interested in understanding how the test script is implemented.

Download testFibService.sh using "wget":
http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a0/testFibService.sh

testFibService.sh requires several ubuntu packages be installed on the client:
 # Before running this script, be sure to install the following packages:
 # sudo apt install curl jq tidy iputils-ping telnet

If not already installed, you'll need to install GNU parallel in your Ubuntu (Linux) environment:

```
# refresh the package list first
sudo apt update

# install the parallel package
sudo apt install parallel
```

Near the top of the script, you'll see parameters for host and port:
host=34.232.53.152
port=8080

Update the host to match the public IPv4 Public IP for your EC2instance.

Now try exercising your web service using this script.
The first parameter is the total number of service requests to perform.
The second parameter is the number of concurrent threads to use.
Since we just have one docker container hosting the service, try just one thread:

```
./testFibPar.sh 10 1
```

Run this script 3 times.

The first and second runs may feature slower times reflecting "warm-up" of the infrastructure: VM, container, JVM…

```
Setting up test: runsperthread=10 threads=1 totalruns=10
run_id,thread_id,json,elapsed_time,sleep_time_ms
1,1,{"number":50000},258,.74200000000000000000
2,1,{"number":50000},300,.70000000000000000000
3,1,{"number":50000},306,.69400000000000000000
4,1,{"number":50000},390,.61000000000000000000
5,1,{"number":50000},274,.72600000000000000000
6,1,{"number":50000},288,.71200000000000000000
7,1,{"number":50000},279,.72100000000000000000
8,1,{"number":50000},356,.64400000000000000000
9,1,{"number":50000},317,.68300000000000000000
10,1,{"number":50000},328,.67200000000000000000
```

By the 3rd run, performance should be fairly consistent and stable.

Task 3 – Creating a Dockerfile for haproxy

Haproxy is a TCP load balancer that is capable of distributing client requests to a very large number of server hosts. We will next create a Docker image for our haproxy load balancer deployment.

```
mkdir docker_haproxy
cd docker_haproxy
```

First, download the sample haproxy config file:

```
wget http://faculty.washington.edu/wlloyd/courses/tcss558/assignments/a0/haproxy.cfg
```

Using a text editor such as vi, pico, or nano, edit the file "Dockerfile" to describe a new Docker image based on ubuntu that will install the Apache tomcat webserver:

```
nano Dockerfile
```

```
# haproxy Dockerfile contents:
FROM ubuntu
RUN apt-get update
RUN apt-get install -y haproxy
COPY entrypoint_haproxy.sh /
COPY haproxy.cfg /etc/haproxy/
ENTRYPOINT ["/entrypoint_haproxy.sh"]
```

Next, create a script called “entrypoint_haproxy.sh” under your docker_haproxy directory as follows:

```
#!/bin/bash
# haproxy daemon - runs container continually until haproxy exits
service haproxy start
echo "haproxy daemon up..."
sleep 3
while :
do
 haproxystatus=`ps aux | grep haproxy-systemd | grep cfg`
 if [ -z "$haproxystatus" ]
 then
  #exit
  echo "haproxy down"
 fi
 sleep 10
done
```

You’ll need to change permissions on this file.
Give the owner execute permission:

```
chmod u+x  entrypoint_haproxy.sh
```

Now, let’s update the haproxy configuration file (haproxy.cfg) using your favorite text editor. As provided the haproxy configuration file will perform round-robin load balancing against 3 nodes:

```
 server web1 54.210.51.9:8080
 server web2 54.210.51.9:8081
 server web3 54.210.51.9:8082
```

So far, we have just one Apache Tomcat server in one container, let’s comment out the bottom two entries by using the “#” character:

```
 server web1 54.210.51.9:8080
#server web2 54.210.51.9:8081
#server web3 54.210.51.9:8082
```

Now, update the IP address (here 54.210.51.9) to match your public IPv4 address of your ec2instance. Also, instead of using port 8080, change this port to 8081.

We will need to destroy your existing tomcat container which is presently using port 8080 and change this to port 8081. First destroy the old container:

```
sudo docker ps -a
```

Locate the "tomcat1" docker instance. The CONTAINER ID will be the left most column. Using this ID, stop the container:

```
sudo docker stop <CONTAINER ID>
```

Now, relaunch the Apache Tomcat container mapping container port 8080 to the host port 8081:

```
sudo docker run -p 8081:8080 -d --rm tomcat1
```

If you've used the Public IPv4 address in the **haproxy.cfg** file, you'll need to open port 8081 in the security group so that network traffic can flow through the port.

Now, we're ready to build the docker container:

```
sudo docker build -t haproxy1 .
```

Check that the haproxy docker image was built:

```
sudo docker images
```

Now let's launch the haproxy container. Haproxy will direct incoming traffic to port 8080 to port 8081 which will map to Apache Tomcat:

```
sudo docker run -p 8080:8080 -d --rm haproxy1
```

Now, using the testParFib.sh script, retest that you're still able to access your webservice, but this time through the haproxy load balancer server.

```
./testFibPar.sh 10 1
```

If this works, then all of the pieces are ready to be deployed across different Docker hosts and containers to complete assignment 0.

Task 4 – Working with Docker-Machine

We will use docker-machine to support working with multiple docker hosts and EC2 instances. Docker-machine makes it very easy to create and destroy instances, and deploy code using Docker containers to multiple VMs on Amazon.

Before we begin, please stop all containers created for Task 2 and Task 3.
Search using "sudo docker ps -a", and use the "sudo docker stop <CONTAINER ID> command to stop ALL running containers.

Sudo vs. non-sudo: When using docker-machine, docker commands run on remote hosts are

not prefaced with “sudo”.

Let’s start by installing the Amazon Web Services Command Line Interface onto your VM (AWS CLI):

```
sudo apt update
sudo apt install awscli
```

Next configure the AWS CLI using your access credentials created earlier:

```
# configure aws cli
aws configure
```

The default region name for Ohio is “us-east-2”.

Next install docker-machine onto your EC2 instance:

```
#to install Docker-Machine:

# Download the application
curl -L https://github.com/docker/machine/releases/download/v0.16.2/docker-machine-Linux-x86_64 >/tmp/docker-machine

# Make it executable
chmod a+x /tmp/docker-machine

# Copy it into an executable location in the system PATH
sudo cp /tmp/docker-machine /usr/local/bin/docker-machine

# verify the version
docker-machine version
```

Check and note that the Docker machine version matches as above.
For further information on Docker Machine see documentation here:
https://docs.docker.com/machine/overview/

Now, let’s create a virtual machine to serve as a docker host.
A single command creates the EC2 instance of the specified type, installs the latest version of docker, and prepares the instance for hosting docker containers !!!

For this step of the tutorial, we specify the use of a “**c5.large**” EC2 instance with 2 virtual CPUs. We will launch this instance as a “spot” instance with a maximum bid price of 10 cents per hour. If using an AWS Educate Starter account, “spot” instances are not supported. When using an AWS Educate Starter account, please remove the two command line attributes relating to spot instances “**`--amazonec2-spot-price ".10" --amazonec2-request-spot-instance`**”. The c5.large will then cost approximately 8.5 cents in the us-east-2 Ohio region *(as of 1/14/2021)*.

Type this command, but don’t execute yet:

```
docker-machine create --driver amazonec2 --amazonec2-region "us-east-2" --
amazonec2-instance-type "c5.large" --amazonec2-spot-price ".10" --amazonec2-
request-spot-instance --amazonec2-zone "b" --amazonec2-open-port 8080 --
amazonec2-open-port 8081  --amazonec2-open-port 8082  --amazonec2-open-port
8083  aw1
```

Note that I've specified availability zone "b". Please set your availability zone accordingly. It will be best to consolidate your instances into the same availability zone for project work in TCSS 558. Set availability zone zone to match your first VM.

Starter accounts may not support spot instances according to this November 2019 AWS document:
https://s3.amazonaws.com/awseducate-starter-account-services/AWS_Educate_Starter_Accounts_and_AWS_Services.pdf

If you receive a spot instance error, retry the command without the spot command-line options:

```
docker-machine create --driver amazonec2 --amazonec2-region us-east-2 --
amazonec2-instance-type "c5.large" --amazonec2-zone "b" --amazonec2-open-port
8080 --amazonec2-open-port 8081  --amazonec2-open-port 8082  --amazonec2-
open-port 8083  aw1
```

Other notes about the docker-machine create command:
The "aw1" refers to the name of the instance. This is the name that you'll use to interact with the VM using the docker-machine CLI. You can use any name desired.

Also please note that docker-machine automatically opens ports using "--amazonec2-open-port <port number>". This automatically adjusts the security-group to provide WORLD access to these ports. ***This is not secure!****, but ok, assuming your instances will not stay up for long.

Try listing docker-machine hosts:

```
docker-machine ls
```

```
NAME   ACTIVE   DRIVER      STATE     URL                       SWARM   DOCKER     ERRORS
aw1    -        amazonec2   Running   tcp://3.16.90.207:2376            v19.03.5
```

You should see something similar to the listing above, 1 remote docker host.

Now change your docker CLI to work against the remote host.

```
eval $(docker-machine env aw1)
```
Check "docker-machine ls" again. The host should be marked "ACTIVE" with a "*".
The following command can also be used to show the active host:

```
docker-machine active
```

Next, we need provide the docker_tomcat and docker_haproxy containers locally on each host. While it is possible to use the "docker save" and "docker load" commands in conjunction with docker-machine to accomplish this, for simplicity we will simply rebuild the images on each host for assignment 0.

Try listing the container images known to this docker host:

```
docker images
```

There aren't any!!! Now, go back into your docker_tomcat directory on your local instance:

```
cd docker_tomcat
```

Rebuild the tomcat container, but this time because we ran the "eval" command above, the build occurs on the remote server:

```
docker build -t tomcat1 .
```

Now check the list of images:

```
docker images
```

Next rebuild the haproxy image on this remote host.

```
cd docker_haproxy
```

Before rebuilding, update the haproxy.cfg file.
Please specify the IP address of the new docker-machine host that is listed using "docker-machine ls". Specify port 8081.

After making these changes, build the haproxy image on the remote host:

```
docker build -t haproxy1 .
```

Now, create an apache tomcat docker container on the remote host.
We will map apache-tomcat's port 8080 to 8081 on the Docker Host.

```
docker run -p 8081:8080 -d --rm tomcat1
```

Next, create the haproxy docker container on the remote host.
We will map haproxy's port 8080 to 8080 on the Docker host.

```
docker run -p 8080:8080 -d --rm haproxy1
```

Now, by refering again to the IP address obtained from "docker-machine ls".
Using the testFibPar.sh script, update the host IP and test the service:

```
./testFibPar.sh 10 1
```

If your service works, then this certifies you've been able to deploy the service onto a docker host using both an apache-tomcat and apache-haproxy container. You're now ready to tackle assignment 0's deliverable (task 5).

Task 5 – For Submission: Testing Alternate Server Configurations

The objective for assignment 0 is to compare performance of running the Fibonacci web service using three different configurations created using Docker and Docker-Machine. To submit assignment 0, create and submit a report using a spreadsheet in Excel, LibreOffice/OpenOffice Calc, or Google Sheets. Optionally the report may be created as a document in Word, LibreOffice/OpenOffice Writer, or Google Docs.

For each configuration, adjust the host and port in the testFibPar.sh script to point to the haproxy container which is set to load balance the containers. Please run testFibPar.sh 3 times, and copy/import the CSV output of the **last, third run** into the report.

Be sure to test the client's ability to run the Fibonacci HTTP REST service on every server **first** using the testFibService.sh script described earlier. **DO NOT SUBMIT RESULTS OF TESTFIBPAR.SH IF ANY SERVER IS FAILING OR THEY WILL BE INCORRECT !**

Run the test script to perform 3 concurrent threads with 10 requests per thread:

```
./testFibPar.sh 30 3
```

In the report, label the testFibPar.sh output for each configuration clearly by name. Please indicate the instance type used (e.g. c5.large) for the docker host(s) for the tests of each test using a table of cells as described below. IT is REQUIRED to use the **same** instance type for all VMs in the configurations.

At the bottom of the report include a summary table of cells. Include a ranking with place, average performance in ms, and % equivalence as follows:

RESULTS SUMMARY:

Performance Ranking	Configuration Name	Average Runtime	Performance Equivalence
1st place	Configuration 2	300ms	100%
2nd place	Configuration 1	400ms	133%
3rd place	Configuration 3	500ms	166%

Create and test the following configurations using docker and/or docker-machine:

Configuration #1 – Co-Located Servers Default CPU Thresholds:

For c5.large VMs, deploy three apache-tomcat containers on one Docker host Virtual Machine.

Map the tomcat containers to use successive port numbers, and update the haproxy configuration accordingly to use these ports:

```
# launch 3 containers on c5.large
# use of sudo assumes not using docker-machine to launch containers
# on remote docker host
sudo docker run -p 8081:8080 -d --rm tomcat1
```

```
sudo docker run -p 8082:8080 -d --rm tomcat1
sudo docker run -p 8083:8080 -d --rm tomcat1
```

Describe configuration 1's VM in the report as follows:

CONFIGURATION 1 VM:

VM instance-id	VM instance type	VM Public IP	Type
i-02021976eb21f2660	c5.large	3.16.90.207	spot instance

Include the text of the haproxy.cfg file at the bottom of the report section.

CONFIGURATION 1 HAPROXY CONFIG:
<text of haproxy.cfg>

Configuration #2 – Co-Located Servers With CPU Thresholds:

For c5.large, deploy three apache-tomcat containers on one Docker host Virtual Machine, with 66% CPU allocation.

```
# launch 3 containers – c5.large weights
# use of sudo assumes not using docker-machine to launch containers
# on remote docker host
sudo docker run -p 8081:8080 -d --rm --cpus .66 tomcat1
sudo docker run -p 8082:8080 -d --rm --cpus .66 tomcat1
sudo docker run -p 8083:8080 -d --rm --cpus .66 tomcat1
```

Describe configuration 2's VM in the report as follows:

CONFIGURATION 2 VM:

VM instance-id	VM instance type	VM Public IP	Type
i-02021976eb21f2660	c5.large	3.16.90.207	spot instance

Include text of the haproxy.cfg file at the bottom of the section for configuration 2.

CONFIGURATION 2 HAPROXY CONFIG:
<text of haproxy.cfg>

Configuration #3 – Separate Servers with No CPU Thresholds:

Deploy three apache-tomcat containers on three separate Docker host Virtual Machines. This will require launching an additional two docker hosts using docker-machine. Map haproxy accordingly on the first host to load balance against the apache-tomcat containers running on the other remote hosts.

To describe the configuration VMs, include a TABLE describing VMs created for configuration 3 as follows:

CONFIGURATION 3 VMs:

VM instance-id	VM instance type	VM Public IP	Type
i-02021976eb21f2660	c5.large	3.16.90.207	spot instance
i-0e8fb81e56cabfedb	c5.large	3.16.44.102	spot instance
i-03c9e2eb76c4364c8	c5.large	3.16.97.137	spot instance

Include text of the haproxy.cfg file at the bottom of the section for configuration 3.

CONFIGURATION 3 HAPROXY CONFIG:
<text of haproxy.cfg>

Use "docker-machine ls" or the AWS web console to find the IP address of each host.

You will need to build the tomcat container separately for each new host.

```
# On each host, launch one apache-tomcat container
# no use of "sudo" below assumes use of docker-machine to launch containers
# on remote docker host
docker run -p 8081:8080 -d --rm tomcat1
```

The expected behavior is that each of these three configurations will perform differently. If this is not the case, please check your configuration to be sure haproxy has been reconfigured correctly each time for the appropriate hosts.

What to Submit

To complete the assignment, upload your report (.xslx spreadsheet file, docx document, or PDF file) into Canvas under assignment 0.

Grading

Each cell in the RESULTS SUMMARY table is worth 2 points. (24 total)
Each cell in the VM descriptions is worth 1 point. (20 total)
Including haproxy.cfg for each configuration is worth 3 points each. (9 total)

This assignment will be scored out of 53 points. (53/53)=100%

Teams (optional)

O*ptionally*, this programming assignment can be completed with **two** person teams.

If choosing to work in pairs, ***only one*** person should submit the team's report to Canvas.

Additionally, ***EACH*** team member should submit an **effort report** to score team participation. **Effort reports** are submitted INDEPENDENTLY and in confidence (i.e. not shared) by each team member.

Effort reports are not used to directly numerically weight assignment grades.

Effort reports should be submitted as a PDF file named: "effort_report.pdf". Google Docs and recent versions of MS Word provide the ability to save or export a document in PDF format.

For assignment 0, the effort report should consist of a one-third to one-half page narrative description describing how the team members worked together to complete the assignment. The description should include the following:

1. Describe the key contributions made by each team member.
2. Describe how working together was beneficial for completing the assignment. This may include how the learning objectives of using EC2, Docker, Docker-machine, and haproxy were supported by the team effort.
3. Comment on disadvantages and/or challenges for working together on the assignment. This could be anything from group dynamics, to commute challenges, to faulty technology.
4. At the bottom of the write-up provide an effort ranking from 0 to 100 for each team member. Distribute a total of 100 points among both team members. Identify team members using first and last name. For example:

   John Doe
   Effort 43

   Jane Smith
   Effort 57

Team members may not share their **effort reports**, but should submit them independently in Canvas as a PDF file. Failure of one or both members to submit the **effort report** will result in both members receiving NO GRADE on the assignment…

Disclaimer regarding pair programming:
The purpose of TCSS 558 is for everyone to gain experience developing and working with distributed systems and requisite compute infrastructure. Pair programming is provided as an opportunity to harness teamwork to tackle assignment challenges. But this does not mean that teams consist of one champion programmer, and a second observer simply watching the champion! The tasks and challenges should be shared as equally as possible.

Docker - Helpful Hints

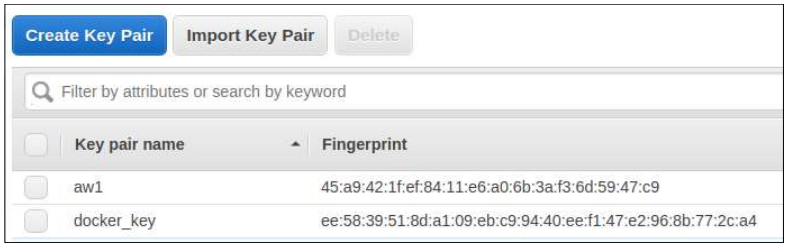
To display all containers running on a given docker node:
```
docker ps -a
```

To stop a container:
```
docker stop <container-id>
```
For example:
```
docker stop cd5a89bb7a98
```

Multiple docker hosts
When creating multiple docker VM hosts on amazon, each host is referred to by name. To see your hosts use the command:
```
docker-machine ls
```

The active host will be shown with a '*'.

The hostname conveniently synced with the AWS keypair name, which is the SSH key used to interact with the virtual machine. If you should need to manually remove keys, this can be done via the EC2 console. On the left-hand side, see "Key Pairs" under "Network & Security". Keys can be deleted if need be using the UI:



To use a specified remote docker host created by docker-machine:

```
eval $(docker-machine env <host-name>)
```

To unset the remote docker host, and work with your local docker:

```
# set docker back to the localhost
eval $(docker-machine env -u)
```

Remove a docker host

Once a host created by docker-machine is no longer needed it can be removed by name. This will destroy the VM and stop any associated charges.

```
$docker-machine rm aw2
```

Shell into a docker container

Obtain the container id with docker ps -a.

```
$sudo docker exec -it <container-id> bash
```

Document History:

v.10 Initial version
v.11 move to tomcat9 for ubuntu 20.04
V.12 added testFibService.sh script. Updated security group screen captures