

Assignment 0

Version 0.14

Cloud Computing Infrastructure Tutorial

Due Date: Friday January 31st, 2020 @ 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.

We will leverage the AWS Educate program for education credits from Amazon Web Services (AWS) to provide cloud computing resources for TCSS 558 projects. We will create virtual machines, known as elastic compute cloud (EC2) instances to host individual nodes of our distributed systems. To support working with VMs to host our distributed applications, we will harness the Docker-Machine tool to automatically create and configure VMs. We will use Docker containers then to deploy code (our nodes) onto VMs.

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.

Use of a Linux environment is recommended for assignment 0.

For Windows 10 users, there is a Ubuntu “App” that can be installed onto Windows 10 directly. This provides a Ubuntu Linux environment without the use of Oracle Virtualbox. Alternatively, Windows users can install Oracle Virtual Box to create virtual machines under Windows 10, and then install a Ubuntu 18.04 virtual machine.

Windows 10 Ubuntu “App” instructions:

https://msdn.microsoft.com/en-us/commandline/wsl/install_guide

Windows Oracle Virtual Box & Ubuntu VM instructions:

There are a number of blogs and YouTube videos that walk through installing Oracle VirtualBox on Windows 10, and how to then install Ubuntu 18.04 LTS on Virtual Box. Search using google.com or video.google.com to find blogs and/or videos to help.

Oracle VirtualBox can be downloaded from:

<https://www.virtualbox.org/wiki/Downloads>

Task 0 - Creating an AWS account

If you do not presently have an AWS account, as of Winter 2020 there are two options that provide up to \$100 in usage credits each.

Before applying, please note your UW email ID has up to ~3 domain name variants that can be used to apply to create a secondary or replacement account. If cloud credits are exhausted on an account with the original email ID, it is possible to apply again with with an alternate email with an “.edu” domain.

UW “.edu” domains include: uw.edu, u.washington.edu, and washington.edu. Occasionally GitHub student developer pack and AWS Educate applications are denied because student status can not be confirmed. In some cases reapplying using an alternate ID has been sufficient to resolve the issue.

The GitHub Student Developer pack provides a AWS Starter account with \$100 in AWS credits. In addition, the GitHub Student Developer pack provides unlimited free public and private repositories on GitHub, and also a \$100 credit to the Azure cloud.

<https://education.github.com/pack>

Once applying, account information will arrive via email in a few minutes to a few days. The user interface is provided by Vocareum, a third party provider. On the UI, there will be a button to access the AWS console:

Vocareum

Welcome to your AWS Educate Account

AWS Educate provides you with access to a wide variety of AWS Services for you to get your hands on and build on AWS! To get started, click on the AWS Console button to log in to your AWS console.

Please read the FAQ below to help you get started on your Starter Account.

- What are the list of services supported?
- What regions are supported with Starter Accounts or Classroom Accounts?
- I can't start any resources. What happened?
- Can I create users within my Starter or Classroom Account for others to access?
- Can I create my own IAM policy within Starter Account or Classroom?
- Can I use marketplace software with my Starter Account or Classrooms?
- Are there any restrictions on AWS services in my AWS Educate Account?

Your AWS Account Status

Active	full access (sxu253@uw.edu)
\$99.98	remaining credits (estimated)
2:60	session time

Account Details AWS Console

Please use AWS Educate Account responsibly. Remember to shut down your instances when not in use to make the best use of your credits. And, don't forget to logout once you are done with your work!

If you already have an AWS account created on your own, not using a UW email, then it should be possible to create a **new** account by applying for the GitHub program under your UW email.

If denied AWS credits through the GitHub program, alternatively you can apply directly to AWS educate using your UW email. This program provides up to ~\$100 in credits. These accounts also appear to be “starter” accounts.

<http://awseducate.com>

Here are the account limitations for starter accounts as of November 2019:

https://s3.amazonaws.com/awseducate-starter-account-services/AWS_Educate_Starter_Accounts_and_AWS_Services.pdf

A key limitation is that spot instances do not appear to be supported.

If you already have a UW AWS account through AWS Educate or Git Hub, but have exhausted credits, try to create a support request. Indicate that you’re a student, and that you’ve exhausted credits from an AWS Educate account from projects in other classes and academic projects. Include a detailed description of classes and projects where credits were spent, and state that you’re requesting credits for the Winter 2020 TCSS 558 Applied Distributed Computing course. **Please note it may take a few days to receive a response from Amazon so start early.**

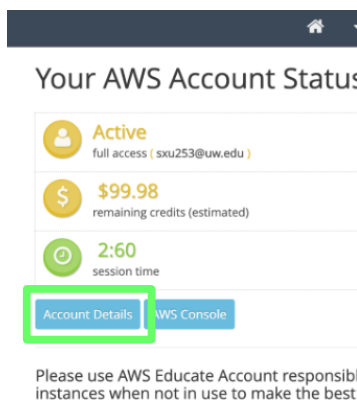
If this doesn’t work, contact the instructor. Provide your AWS account ID (if available), and your UW email address the AWS account was created with. The instructor will follow-up with the UW AWS Educate representative.

Note it is possible to complete assignment 0 using only free tier resources (e.g. t2.micro instances, *no spot instance needed*).

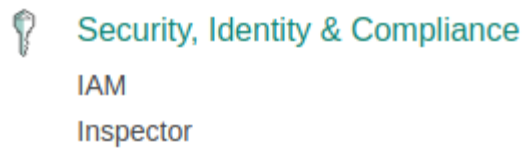
Task 1 - AWS account setup

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

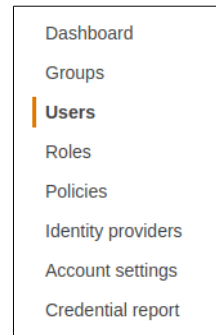
If you’re using an AWS starter account: Obtain account credentials for your AWS starter account by going to the Vocareum user interface in your web browser, and click the **blue button** labeled **“Account Details”**. This should provide access the account credentials including an **access key** and 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:

The image shows the "Add user" form in the AWS IAM console. The form is titled "Add user" and has a "Details" button in the top right corner. It is divided into two sections: "Set user details" and "Select AWS access type". In the "Set user details" section, there is a text input field for "User name*" containing the text "tcss558" and a blue button with a plus sign and the text "Add another user". In the "Select AWS access type" section, there are two radio button options: "Programmatic access" (which is selected) and "AWS Management Console access". The "Programmatic access" option includes the text "Enables an access key ID and secret access key for the AWS API, CLI, SDK, and other development tools." The "AWS Management Console access" option includes the text "Enables a password that allows users to sign-in to the AWS Management Console." There are also "Learn more" links in both sections.

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.

User	Access key ID	Secret access key
▶ <input checked="" type="checkbox"/> tcsc558	AKIAI44QH8D8DFK1H5G5	***** Show

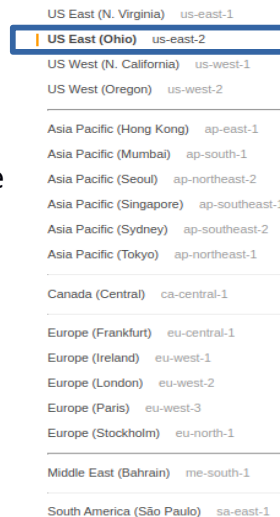
Once you've downloaded these keys, be sure to **never** publish these key values in a source code repository such as github where your account credentials could be exposed. **Protect these keys as if they were your credit card or wallet!**

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.

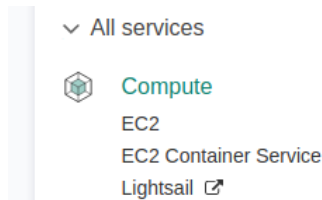
Choose the "region" that you'll work in. Recommended options are "US East (Ohio)" known as "us-east-2" via the CLI. The Ohio region is newer, has less traffic, and has been seen as **less expensive** than others for EC2 spot instances.

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 use “t2.micro” instances. 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”:



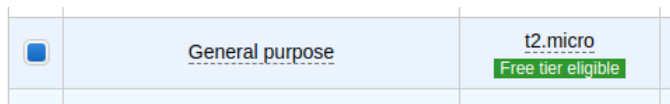
Next, click “Instances” on the left-hand side menu. Then click the blue “Launch Instance” button:



Select Ubuntu:



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



Next, specify the following instance details:

Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of th

Number of instances [Launch into Auto Scaling Group](#)

Purchasing option Request Spot instances

Network [Create new VPC](#)

Subnet [Create new subnet](#)

Auto-assign Public IP

Placement group Add instance to placement group

Capacity Reservation [Create new Capacity Reservation](#)

IAM role [Create new IAM role](#)

Shutdown behavior

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:

 Select an existing security group

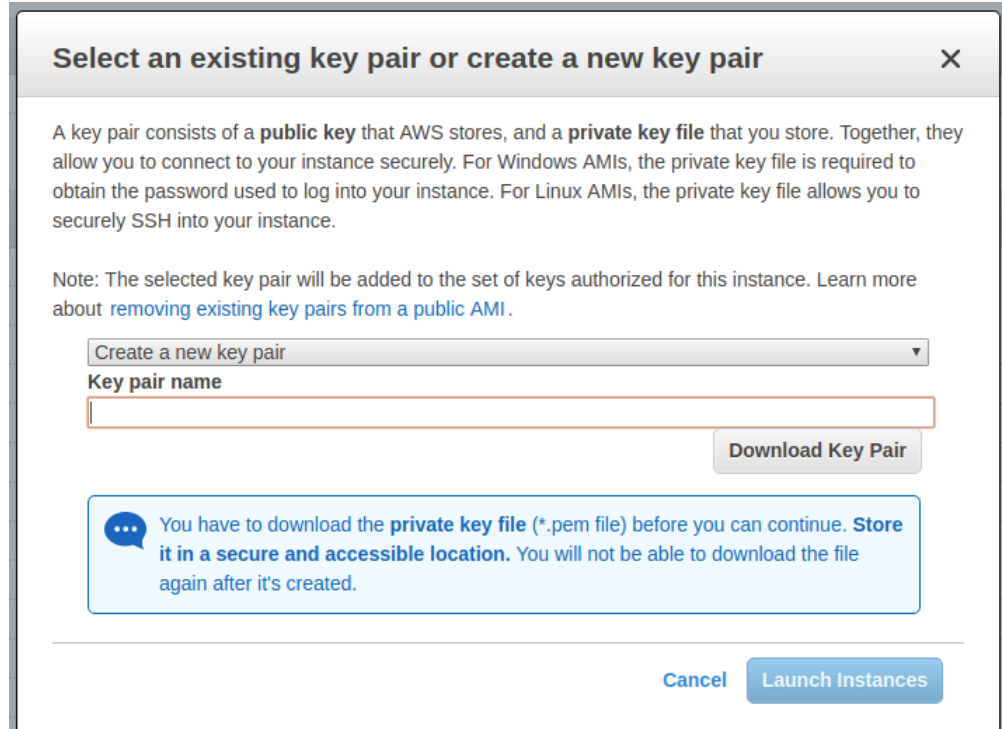
And then mark the option for “default VPC security group”.

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 screenshot shows a dialog box titled "Select an existing key pair or create a new key pair". It contains the following elements:

- A close button (X) in the top right corner.
- Explanatory text: "A key pair consists of a **public key** that AWS stores, and a **private key file** that you store. Together, they allow you to connect to your instance securely. For Windows AMIs, the private key file is required to obtain the password used to log into your instance. For Linux AMIs, the private key file allows you to securely SSH into your instance."
- Note: "Note: The selected key pair will be added to the set of keys authorized for this instance. Learn more about [removing existing key pairs from a public AMI](#)."
- A dropdown menu with "Create a new key pair" selected.
- A text input field labeled "Key pair name" with a red border, currently empty.
- A "Download Key Pair" button.
- A blue information box with a speech bubble icon: "You have to download the **private key file** (*.pem file) before you can continue. **Store it in a secure and accessible location.** You will not be able to download the file again after it's created."
- At the bottom, "Cancel" and "Launch Instances" buttons.

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
```

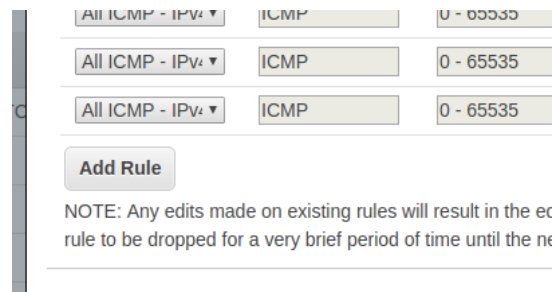
Before you can SSH into the instance, the default security group used by your instance must be modified to allow SSH (port 22) access from your computer.

In the Amazon management console., under instances, look at the detailed instance information and click on “**default**” next to “Security groups”:

Availability zone	us-east-
Security groups	default.
Scheduled events	No sche

Click the “Inbound” tab, and then the “Edit” 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

Then “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 SHA256:0cy2eP8Q15zmBThAqTq9z1Tw00+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

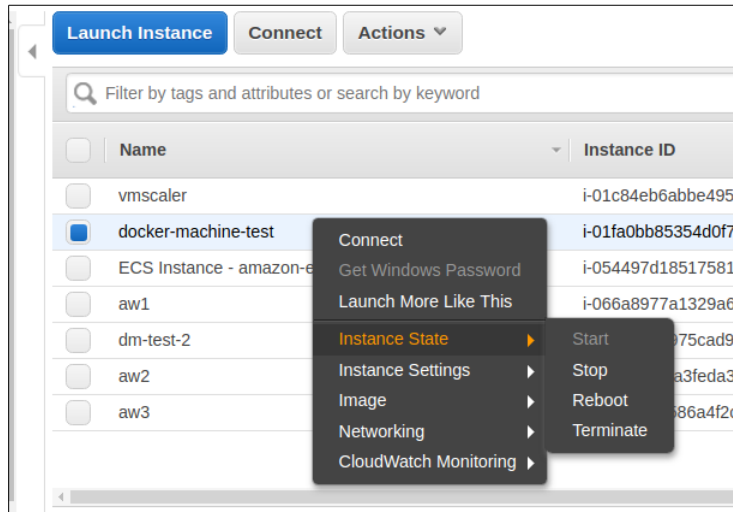
The idea is when you reconnect to the VM again, there is a possibility that someone masquerades as the VM. To prevent someone from masquerading as the VM you're trying to connect to, ssh tracks the identity of each host and alerts the user **every time** there is a change. Sometimes the changes are expected, 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.

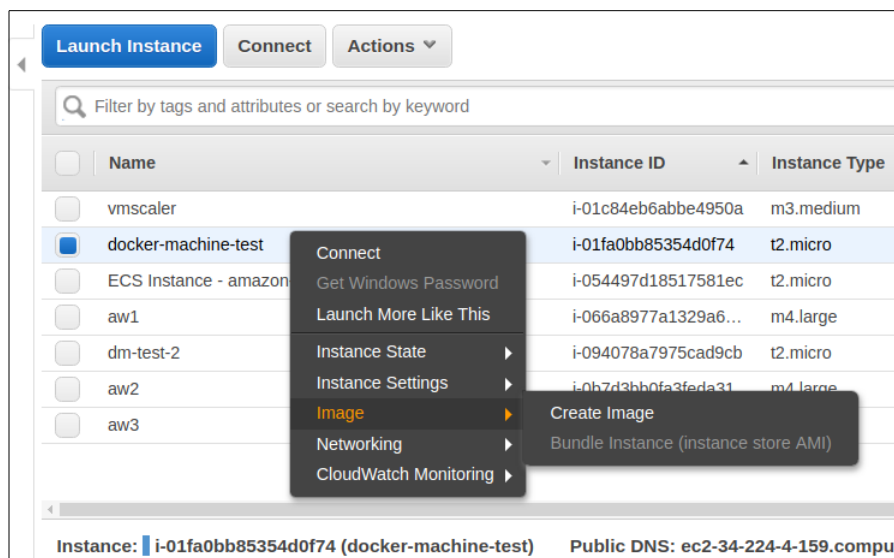
Name	Volume ID	Size	Volume Type	IOPS	Snapshot	Created	Availability Zone
	vol-046f0855...	16 GiB	gp2	100 / 3000	snap-0548d02...	October 4, 2017 at 7:59:58 PM ...	us-east-1e
	vol-0666ab9...	16 GiB	gp2	100 / 3000	snap-0548d02...	October 4, 2017 at 7:58:07 PM ...	us-east-1e
	vol-048394b...	16 GiB	gp2	100 / 3000	snap-0548d02...	October 4, 2017 at 5:46:22 PM ...	us-east-1e
	vol-06c5e58...	8 GiB	gp2	100 / 3000	snap-0cfc17b0...	October 2, 2017 at 9:01:27 PM ...	us-east-1a
	vol-026d9ed...	22 GiB	gp2	100 / 3000		October 2, 2017 at 8:41:10 PM ...	us-east-1a
	vol-070d24a...	8 GiB	gp2	100 / 3000	snap-0c39fbce...	October 2, 2017 at 8:41:10 PM ...	us-east-1a
	vol-03cba5c...	8 GiB	gp2	100 / 3000	snap-0cfc17b0...	October 2, 2017 at 8:01:57 PM ...	us-east-1e
	vol-0cc7d3e...	10 GiB	gp2	100 / 3000	snap-ff270c9f	September 11, 2017 at 1:17:19 ...	us-east-1e
	vol-052ea2a...	10 GiB	gp2	100 / 3000	snap-ff270c9f	September 11, 2017 at 12:40:55 ...	us-east-1e
	vol-088b1f0c...	10 GiB	gp2	100 / 3000	snap-ff270c9f	August 30, 2017 at 3:53:15 PM ...	us-east-1e
	vol-0228356...	10 GiB	gp2	100 / 3000	snap-1bcbf463	June 9, 2017 at 5:25:22 PM UTC-7	us-east-1e
	vol-09230ce...	10 GiB	gp2	100 / 3000	snap-1bcbf463	June 9, 2017 at 5:25:22 PM UTC-7	us-east-1e
	vol-02731e8...	10 GiB	gp2	100 / 3000	snap-1bcbf463	June 9, 2017 at 5:25:22 PM UTC-7	us-east-1e

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-hand side 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
```

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 tomcat8
COPY fibo.war /usr/share/tomcat8/webapps/
COPY entrypoint_tomcat.sh /
RUN mkdir /usr/share/tomcat8/logs
RUN mkdir /usr/share/tomcat8/temp
RUN ln -s /var/lib/tomcat8/conf /usr/share/tomcat8
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/tomcat8/bin/startup.sh
echo "tomcat daemon up..."
sleep 3
while :
do
tomcatstatus=`ps aux | grep tomcat8 | 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
```

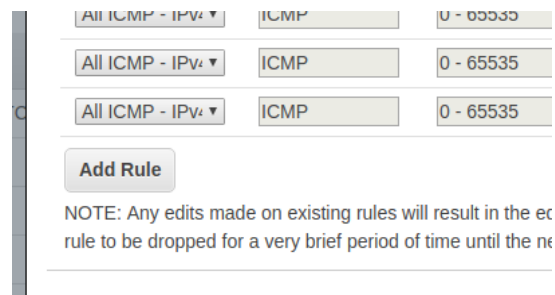
Now, you'll need to open port 8080 in the default security group in the Amazon management console.

Under instances, look at the detailed instance information and click on "default" next to "Security groups":

Availability zone us-east-1
Security groups default
Scheduled events No scheduled events

Click the "Inbound" tab, and then the "Edit" 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

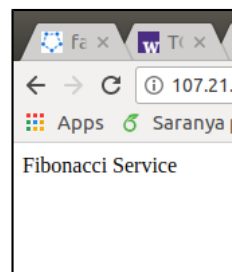
Source = My IP

Then "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.

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

```
sudo apt-get 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
```

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 !!!

Below, I've specified "c5.large" an EC2 instance with 2 virtual CPUs. **Note this is not free. See discussion below on using the free t2.micro if needed.** We will launch this instance as a "spot" instance with a maximum bid of 10 cents per hour:

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/awse educate-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.

Alternatively, you could use "FREE tier t2.micro" instances for your docker host(s). These instances will spend on your 750-hours/month FREE credits for t2.micro instances. These single-CPU instances are limited to an initial 30-minute burst of 100% CPU utilization. Once CPU credits are exhausted, the instance is down-throttled to 10% of one CPU core until credits are earned at a rate of 6-minutes @100% utilization per hour:

Instance type	Initial CPU credit*	CPU credits earned per hour	vCPUs	Base performance (CPU utilization)	Maximum earned CPU credit balance***
t2.nano	30	3	1	5%	72
t2.micro	30	6	1	10%	144
t2.small	30	12	1	20%	288
t2.medium	60	24	2	40% (of 200% max)**	576
t2.large	60	36	2	60% (of 200% max)**	864
t2.xlarge	120	54	4	90% (of 400% max)**	1296
t2.2xlarge	240	81	8	135% (of 800% max)**	1944

From: <http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/t2-instances.html>

These t2 instances are not spot instances. They are considered full price where the first 750 hours is free. To create a t2.micro docker host:

```
docker-machine create --driver amazonec2 --amazonec2-region us-east-2
--amazonec2-instance-type "t2.micro" --amazonec2-zone "b" --amazonec2-open-
port 8080 --amazonec2-open-port 8081 --amazonec2-open-port 8082
--amazonec2-open-port 8083 aw1
```

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 referring 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.

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, t2.micro) 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
1 st place	Configuration 2	300ms	100%
2 nd place	Configuration 1	400ms	133%
3 rd 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.

For t2.micro VMs, deploy only one apache-tomcat container on the Docker host.

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
docker run -p 8081:8080 -d --rm tomcat1
docker run -p 8082:8080 -d --rm tomcat1
docker run -p 8083:8080 -d --rm tomcat1
```

```
# launch only 1 container for t2.micro
docker run -p 8081:8080 -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>

For t2.micro, the haproxy.cfg will point to one Apache tomcat container only.

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.

If using t2.micro, use only one apache-tomcat container on the Docker host.

```
# launch 3 containers - c5.large weights
docker run -p 8081:8080 -d --rm --cpus .66 tomcat1
docker run -p 8082:8080 -d --rm --cpus .66 tomcat1
docker run -p 8083:8080 -d --rm --cpus .66 tomcat1

# launch only 1 container for t2.micro
docker run -p 8081:8080 -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>

For t2.micro, the haproxy.cfg will point to one Apache tomcat container only.

Configuration #3 - Separate Servers 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
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 (.xlsx 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)

Optionally, 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

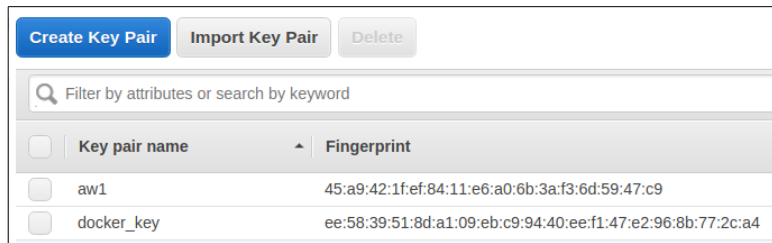
When creating multiple docker VM hosts on amazon, each host is referred to by name. To see your hosts use the command:

```
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:



<input type="checkbox"/>	Key pair name	Fingerprint
<input type="checkbox"/>	aw1	45:a9:42:1fef:84:11:e6:a0:6b:3a:f3:6d:59:47:c9
<input type="checkbox"/>	docker_key	ee:58:39:51:8d:a1:09:eb:c9:94:40:ee:f1:47:e2:96:8b:77:2c:a4

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 AWS Educate starter account is now recommended optional

v.12 Corrected quotes on docker-machine command around region name

v.14 Changes regarding use of t2.micro instances, and obtaining account credentials for an AWS starter account