TCSS 558: Applied Distributed Computing
Winter 2023
http://faculty.washington.edu/wlloyd/courses/tcss558

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

Assignment 0

Version 0.11
Cloud Computing Infrastructure Tutorial

Due Date: Tuesday January 31st, 2023 @ 11:59 pm, tentative

Objective

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

As a prerequisite to this assignment, it is assumed that you will have access to an AWS account.

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 users can install Oracle Virtual Box to create virtual machines under Windows 10/11, and then install a Ubuntu virtual machine.

For Windows users, there is an Ubuntu "App" that can be installed onto Windows 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 22.04 LTS ISO file can be downloaded here:
https://releases.ubuntu.com/22.04.1/ubuntu-22.04.1-desktop-amd64.iso (3.6 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. Note the live image is a server image:
https://cdimage.ubuntu.com/releases/22.04/release/ubuntu-22.04.1-live-server-arm64.iso (1.3 G)

Another approach is to obtain Ubuntu using a bit torrent stream by downloading:

https://cdimage.ubuntu.com/releases/22.04/release/ubuntu-22.04.1-live-server-arm64.iso.torrent (107K)

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 (2018, many views):
https://www.youtube.com/watch?v=sB_5fqiysi4

Installing Ubuntu 22.04 on Windows 11 Oracle VirtualBox
https://youtu.be/v1JVqd8M3Yc

And here are instructions for installing Ubuntu 20.04 on Oracle VirtualBox:
https://youtu.be/zHwFtyxJsog

Generic instructions for installing Ubuntu Virtual Box virtual machine:
https://ubuntu.com/tutorials/how-to-run-ubuntu-desktop-on-a-virtual-machine-using-virtualbox#1-overview

Instructions including how to install Guest Additions on the Ubuntu VM
(this enables integrated clipboard for copy & paste, and other beneficial features):
https://linuxconfig.org/virtualbox-install-guest-additions-on-ubuntu-22-04-lts-jammy-jellyfish

When working with Virtual Machines, the base operating system (e.g. Windows) 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 22.04 LTS
Host operating system = Windows 10/11 or Mac OSX

For M1 Mac (ARM CPU) users, there is only a preview version of Virtual Box available currently.
As an alternative to Virtual Box, M1 mac users are encouraged to try out:

UTM Hypervisor (for running VMs on M1 MAC)
https://mac.getutm.app/

A non-free alternative is the student edition of the Parallels Hypervisor on MAC:
https://www.parallels.com/landingpage/pd/education/

Once your VM is installed, it is highly recommended to install the VirtualBox "Guest Additions". The guest additions provide important features such as ***shared clipboards*** between the host and the guest, as well as ***file system sharing***, and ***mouse pointer integration***.

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

Task 1 – AWS account setup

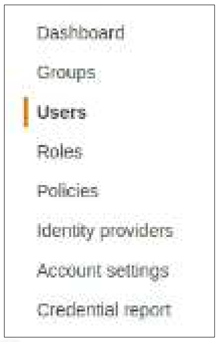
Once having access to AWS, task #1 involves creating AWS account credentials to work with Docker-Machine, if you have not already done so.

First, login to your AWS account at: **https://aws.amazon.com/**

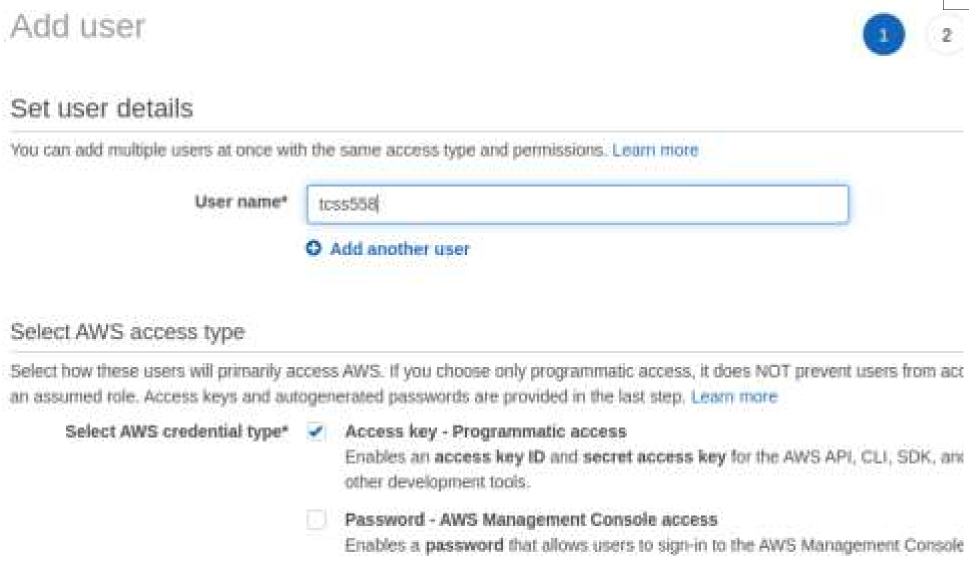
From the AWS Management Console home, locate "IAM" Identity Access Management, and select it. You can search for "IAM" using the search bar.

IAM us under the "Security, Identity & Compliance" group of features:

Next, from the IAM dashboard, on the left hand-side select "Users" → :



Provide a user account name. Here I am using "tcss558" as an example:



Add user

Set user details

You can add multiple users at once with the same access type and permissions. Learn more

User name* tcss558

Add another user

Select AWS access type

Select how these users will primarily access AWS. If you choose only programmatic access, it does NOT prevent users from acc
an assumed role. Access keys and autogenerated passwords are provided in the last step. Learn more

Select AWS credential type* ✓ **Access key - Programmatic access**
Enables an **access key ID** and **secret access key** for the AWS API, CLI, SDK, and
other development tools.

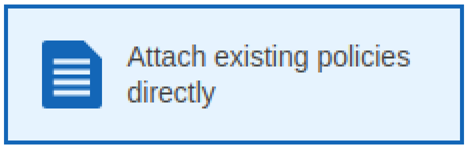
☐ **Password - AWS Management Console access**
Enables a **password** that allows users to sign-in to the AWS Management Console

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

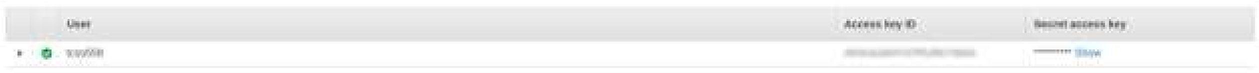
If you plan to use this user account to explore additional Amazon's services, then admin access can be added (not required):

* AdministratorAccess

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

Now click the "Next: Tags" button, it is not necessary to add any tags.
Now click the "Next: Review" button, and if the settings look correct, 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
[illegible]	[illegible]	******** Show

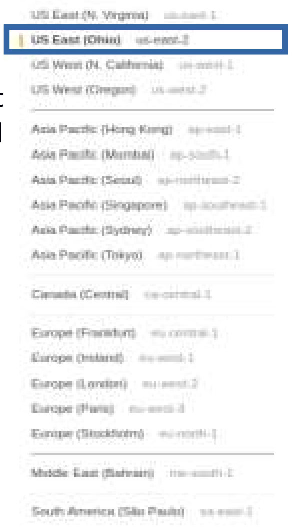
Once you've downloaded these keys, be sure to **never** publish these key values in a source code repository such as github where your account credentials could be exposed.

<u>Protect these keys as if they were your credential card or wallet!</u>

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:


US East (N. Virginia) us-east-1
US East (Ohio) us-east-2
US West (N. California) us-west-1
US West (Oregon) us-west-2
Asia Pacific (Hong Kong) ap-east-1
Asia Pacific (Mumbai) ap-south-1
Asia Pacific (Seoul) ap-northeast-2
Asia Pacific (Singapore) ap-southeast-1
Asia Pacific (Sydney) ap-southeast-2
Asia Pacific (Tokyo) ap-northeast-1
Canada (Central) ca-central-1
Europe (Frankfurt) eu-central-1
Europe (Ireland) eu-west-1
Europe (London) eu-west-2
Europe (Paris) eu-west-3
Europe (Stockholm) eu-north-1
Middle East (Bahrain) me-south-1
South America (São Paulo) sa-east-1

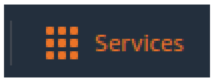
Choose the AWS "region" that you'll work in. The recommended option is "US East (Ohio)" known as "us-east-2" via the CLI. The Ohio region is newer, has less traffic, and has been seen as ***<u>less expensive</u>*** than others for EC2 spot instances.

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.

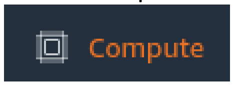
For assignment 0, we will begin testing by using a "t2.micro" instance. Users are allowed up to 750-hours each month of instance time for FREE using the t2.micro type.

With the t2.micro VM, with a new AWS account, you need not have received any AWS credits to complete assignment 0 because the t2.micro VM is essentially FREE in the first year of the account.

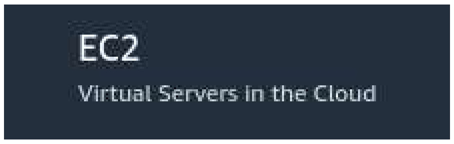
Click on the AWS services icon.



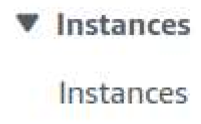
Select Compute services:



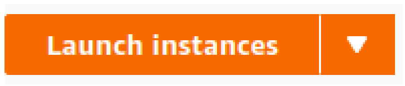
Then select “EC2”:



Next, let’s launch a t2.micro instance.
On the menu on the far-left click “**Instances**” that is under Instances:

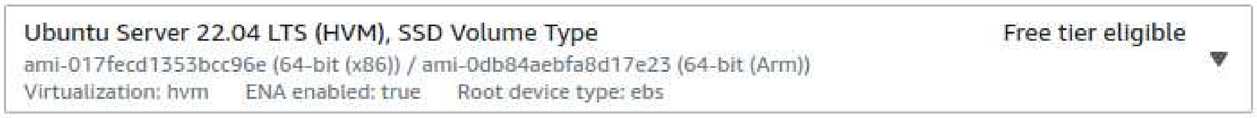


Try launching an instance using the ec2 instance launch wizard.
Press the “Launch Instances” button:



Name and tags
At the top of the user interface under **Name and** tags, a VM name and tags can be specified.
Tags allow key/value assignments to help identify VMs.
These key/value pairs can be queried using the EC2 API to programmatically identify specific VMs that have a designated purpose for your application.
We will not use this feature right now.

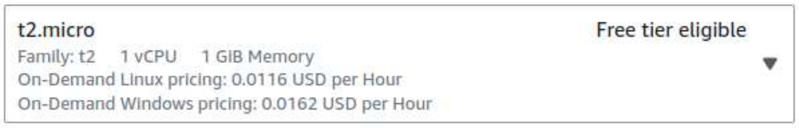
Application and OS Images (Amazon Machine Images)
For the **Application and OS images (Amazon Machine Image)** select the **Ubuntu** button.
Then from the **Amazon Machine Image (AMI)** dropdown list,
Select Ubuntu 22.04 LTS (HVM) as the Amazon Machine Image.


Ubuntu Server 22.04 LTS (HVM), SSD Volume Type
ami-017fecd1353bcc96e (64-bit (x86)) / ami-0db84aebfa8d17e23 (64-bit (Arm))
Virtualization: hvm ENA enabled: true Root device type: ebs

Amazon Machine Images are snapshots of pre-installed machines. Choosing an AMI will cause its snapshot to be replicated onto your live EBS volume. This will allow you to select an operating system for the VM.

Instance Type
Next, specify the **Instance Type**.
In the dropdown, search for and select "t2.micro". 't2' instances are a low-cost, general purpose instance type that provides a baseline level of CPU performance with the ability to burst above the baseline when needed. Upon creation, the t2.micro is provided 30-minutes of CPU credits. 1 CPU credit is equivalent to running the VM at 100% utilization for 1 minute.

https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/burstable-performance-instances-standard-mode-concepts.html


t2.micro
Family: t2 1 vCPU 1 GiB Memory
On-Demand Linux pricing: 0.0116 USD per Hour
On-Demand Windows pricing: 0.0162 USD per Hour

For every hour the t2.micro is active, an additional 6 CPU credits is earned.
A maximum of 144 earned CPU credits is allowed. (2 hours 24 minutes of 1 vCPU @ 100%)

Key pair (login)
If you already have a key pair, then select it from the dropdown list. If not select the link: "Create new key pair":

Create new key pair

When creating a key pair, using defaults is ok. Download the file to a safe location.

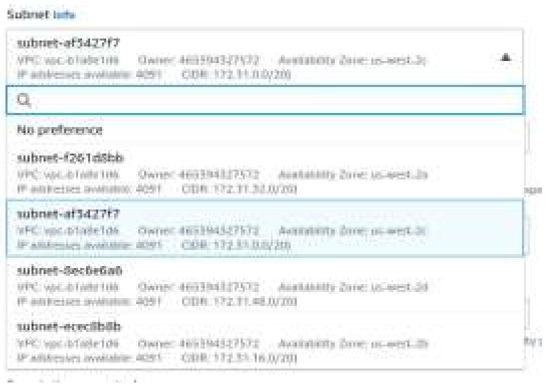
Network Settings
Make changes to the network settings for your VM launch request.
To make changes you'll need to click on the **Edit** button on the right-hand side of the Network Settings frame.

Here you can specify the Virtual Private Cloud (VPC) that the VM is created on. Every VM has a VPC. **Most of the time we use the default VPC.** Creating a custom VPC allows for specialized network configurations.

Subnet:
This is how you select a specific availability zone (AZ). AZs are specific data centers within a cloud region. Think of AZs like separate buildings. Each AZ can be physically separate from another by several miles, like different buildings.

In the future using EC2, we may want to specify a specific AZ when creating spot instances. Spot instances are low-cost VMs priced at approximately 25 to 33% of the full price. They do not provide a 100% guarantee of up-time. They are terminated automatically by Amazon when there is a spike in demand. The trade-off is receiving a discounted low-price VM. The catch is that the price will vary based on the specific AZ. As a user, it is necessary to specify which AZ to use to receive the lowest price. Prices change based on demand.

Subnets are virtual networks that are configured which then create the VM on a specific AZ.
The example shows how different subnets are mapped to different availability zones: us-west-2a, us-west-2b, us-west-2c, or us-west-2d. We are using Ohio, so it will be us-east-2X instead.



Subnet Info

subnet-af5427f7
VPC: vpc-b1a8e1d6 Owner: 465394327572 Availability Zone: us-west-2c
IP addresses available: 4091 CIDR: 172.31.0.0/20

No preference

subnet-f261d8bb
VPC: vpc-b1a8e1d6 Owner: 465394327572 Availability Zone: us-west-2a
IP addresses available: 4091 CIDR: 172.31.32.0/20

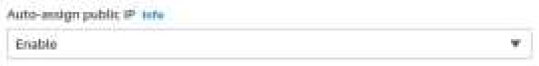
subnet-af5427f7
VPC: vpc-b1a8e1d6 Owner: 465394327572 Availability Zone: us-west-2c
IP addresses available: 4091 CIDR: 172.31.0.0/20

subnet-8ec6e6a6
VPC: vpc-b1a8e1d6 Owner: 465394327572 Availability Zone: us-west-2d
IP addresses available: 4091 CIDR: 172.31.48.0/20

subnet-ecec8b8b
VPC: vpc-b1a8e1d6 Owner: 465394327572 Availability Zone: us-west-2b
IP addresses available: 4091 CIDR: 172.31.16.0/20

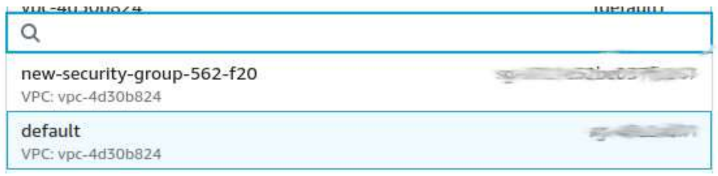
You do not need to specify an availability zone for the t2.micro VM unless you want to as the price does not vary unless creating a spot instance.

Auto-assign Public IP:
Subnets have a default setting whether a VM should receive a public IP address.
The property is called "Auto-assign public IPv4 address". Be sure this is set to Enable:



Auto-assign public IP Info
Enable

Next under **Network settings**, **Firewall (security group):**
Select the radio button: "Select existing security group".
And select the **default** security group from the list of choices.



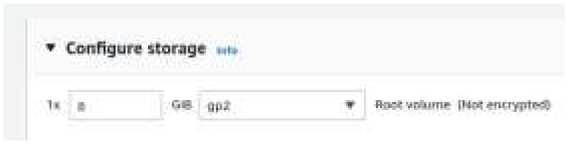
new-security-group-562-f20
VPC: vpc-4d30b824

default
VPC: vpc-4d30b824

Selecting the default security group, allows you to add common firewall access rules to enable rapid access to newly created VMs. When selecting the security group, firewall rules are displayed. They can not be changed here. They can be modified later using the Security Group GUI.

Next, configure Storage options.

By default under “Configure storage”, t2.micro will by default set the ROOT (EBS) volume to have 8GB, using the gp2 EBS volume type.



▼ Configure storage Info

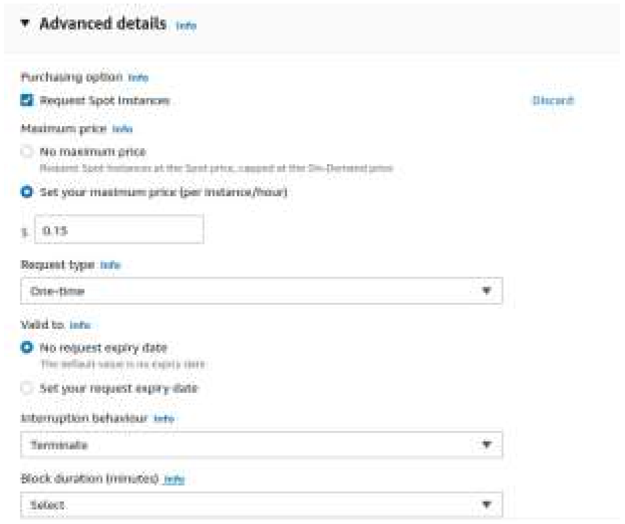
1x 8 GiB gp2 Root volume (Not encrypted)

We can identify an EBS volume based on the Volume Type. Some available types for EBS include general purpose (gp2 and gp3), provisioned SSD (io1), provisioned SSD (io2), and magnetic (which is fact no longer “standard” by the way). These can be seen in the drop down.

<u>**>>> FOR THIS TUTORIAL, PLEASE USE gp2 – General Purpose 2 as the EBS volume type.**</u>

If using a new AWS account, less than 1 year old, t2.micro is FREE.
If the account is older, you can create a t2.micro spot instance for a reduced price:

<u>Request spot instance</u>
Next, configure the instance details.
Expand “Advanced Details”, and specify the following configuration:



▼ Advanced details Info

Purchasing option Info
[✓] Request Spot Instances Discard

Maximum price Info
No maximum price
Request Spot Instances at the Spot price, capped at the On-Demand price
[✓ Set your maximum price (per instance/hour)]
$ 0.15

Request type Info
One-time

Valid to Info
[✓ No request expiry date]
The default value is no expiry date
Set your request expiry date

Interruption behaviour Info
Terminate

Block duration (minutes) Info
Select

For purchasing option, select “Request Spot Instance”, and then click “**Customize**”.

Specify a “Maximum price” of “0.02” for 2 cents.

Set a “One-time” request type, as opposed to a persistent request.

When the request type is a “persistent request”, then when the spot VM is terminated, either voluntary, or accidentally, the VM will always automatically RECREATE itself!
This leads to a cost savings measure:

Cost Saving Measure:

BE CAREFUL USING PERSISTENT REQUESTS FOR SPOT INSTANCES

When using **persistent** spot requests, when you intentionally terminate the VM, the VM will automatically be recreated within 1-2 minutes resulting in ongoing charges !!! Imagine, you shut your computer down for the weekend, only to learn that the VM came back to life, and charged for multiple days of inactivity !! When using Persistent Spot Requests, it is necessary to physically **DELETE THE SPOT REQUEST** in addition to terminating the instance. Failure to do so, will result in ADDITIONAL CHARGES ! **BE WARNED !**

For the Interruption behavior it is typical to select "Terminate". To select "Stop" for a spot instance you must specify a persistent request type.

Next, Review the details of the configured instance in the Summary window pane to the right.
This allows a check to see that all settings are acceptable prior to creating the VM.
If everything looks okay, press the "Launch Instance" button:

Throughout the tutorial, Linux commands are prefaced with the "$".

Comments are prefaced with a "#".

Log into your AWS EC2 t2.micro instance:

If this is the first time you've created a VM, you should have created a new key pair which is used to establish a secure shell (ssh) connection to your new VM.

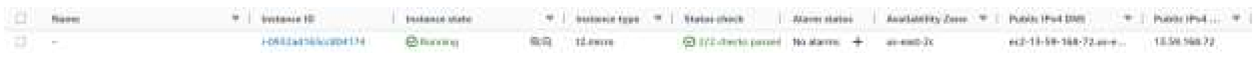
From your Linux VM's command line, change permissions on the keyfile you downloaded:

```
$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.

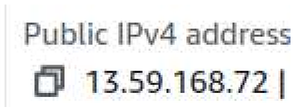
Select "Instances" on the left-hand side of the screen.

Locate your newly launched VM. It should be the only one!
Select your VM on the left:



Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Public IPv4 DNS	Public IPv4 ...
–	[illegible]	Running	t2.micro	2/2 checks passed	No alarms	us-east-2c	ec2-13-59-168-72.us-e...	13.59.168.72

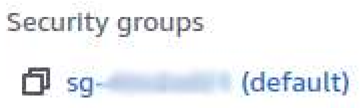
Then, look for "Public IPv4 address" and select the copy-icon just to the LEFT of the IP address:


Public IPv4 address
13.59.168.72 |

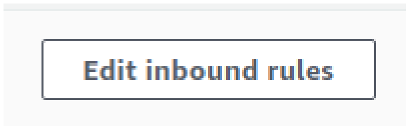
This copies the IP address to the clipboard.
Before connecting to the VM, configure the security group to allow SSH access.

Click on the "Security" tab of the instance.

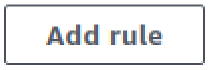
Click on the security group ID:


Security groups
sg-[redacted] (default)

On the list of Inbound Rules, press the button:

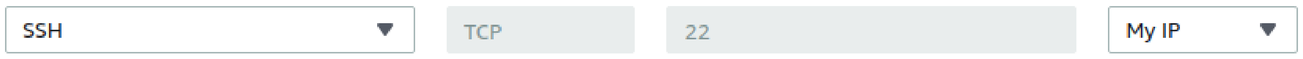


At the bottom of the list, press the button:

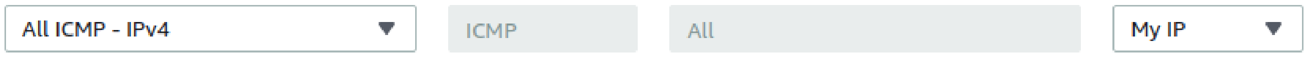


Configure a new rule as follows:

Type: "SSH", Source: "My IP"


SSH | TCP | 22 | My IP

Another Security Group option is to add the "All ICMP – IPv4" permission. This enables you to "ping" your VM.


All ICMP - IPv4 | ICMP | All | My IP

For every new VM that is launched, when you associate the default security group, these security group settings will be applied. Overtime you can accumulate rules for popular places where you use your laptop. This could be places like your home, the local Starbucks, and the UW-Tacoma campus.

As an alternative to selecting "MyIP" you can specify a wider subnetwork address range. This way if your compute receives different IP addresses within the same CIDR (address) block, you don't have to update the security group as often. (e.g. every day) CIDR block stands for classless-inter-domain routing. A CIDR block is a group of IP addresses the form a "sub-network". Within your house, all of the devices sharing your WiFi connection usually receive private IP addresses sharing the same CIDR-block, where all addresses are private (not public IPs).

Find out your IP address.
In your web browser, open Google search, and type in “What is my IP?”.

Your IP address should appear.
Note the first 3 numbers.
Let’s add SSH permission for your CIDR network block.

If your IP is for example **120.118.53.**108, then your 24-bit CIDR block which would include all 255 addresses on the local subnet will be 120.118.53.0/24.

As an alternative to MyIP, in the security group, add your entire CIDR block, e.g. 120.118.53.0/24

The motivation for adding your CIDR block is that if you commonly use a buliding’s WiFi network you may receive various addresses in the same block from day-to-day. If you add ranges vs. individual IPs you’ll hopefully not need to update the security as often.

An even better way to set the network address range is to ask your network administrator what the wireless address range is. Do accept any dynamically assigned IP from a given wireless network, you may need to actually specify ‘/21’ which would allow up to 4096 unique addresses and not the 256 associated with ‘/24’.

Once complete, save changes:

Return to the list of Instances, by clicking the “Instances” option on the left-hand menu.

Now navigate to a Linux terminal.
If using Windows without a Linux environment, a 3rd-party program like PuTTY is required.
To learn more, see this tutorial:
https://www.ssh.com/ssh/putty/windows/install

Otherwise, open a Linux terminal, and navigate to the subdirectory using “cd” where you have stored your key using the absolute path. For example: “/home/username/awskeys/”

First try pinging your VM

```
ping 13.59.168.72
```

Use CTRL-C to stop the ping:

```
PING 13.59.168.72 (13.59.168.72) 56(84) bytes of data.
64 bytes from 13.59.168.72: icmp_seq=1 ttl=33 time=70.3 ms
64 bytes from 13.59.168.72: icmp_seq=2 ttl=33 time=70.7 ms
64 bytes from 13.59.168.72: icmp_seq=3 ttl=33 time=72.3 ms
64 bytes from 13.59.168.72: icmp_seq=4 ttl=33 time=70.5 ms
```

Pinging provides a rough estimate of the network latency between your computer, and the VM. Here the ping sends a 64-byte packet to the cloud VM. The cloud responds and the client measures the round-trip response time.

Ping-time to the VM is a good way to test your network's ability to access cloud resources. A VM in Oregon should have a lower ping time than Ohio, Virginia, or a VM in a different continent.

Now launch an ssh session as follows:

<replace with your VM's IP>
```
ssh -i mykey.pem ubuntu@13.59.168.72
```

or if not in the same directory as the key:

<replace with your VM's IP>
```
ssh -i /home/username/awskeys/mykey.pem ubuntu@13.59.168.72
```

You will then be prompted, because your machine doesn't recognize the MAC address of the host, say 'YES'; to save the HW address and proceed with the connection:

```
The authenticity of host '13.59.168.72 (13.59.168.72)' can't be established.
ECDSA key fingerprint is SHA256:dhfCG+k/Zz7p13d39cAIiGfTCKW0zTHwtLTdoJQspp4.
Are you sure you want to continue connecting (yes/no/[fingerprint])? Yes
```

You may receive the following error:

```
Warning: Permanently added '13.59.168.72' (ECDSA) to the list of known hosts.
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@         WARNING: UNPROTECTED PRIVATE KEY FILE!          @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Permissions 0664 for 'mykey.pem' are too open.
It is required that your private key files are NOT accessible by others.
This private key will be ignored.
Load key "mykey.pem": bad permissions
Permission denied (publickey).
```

This indicates that the ssh key permissions are too open.
Adjust as described earlier with the command:

```
chmod 0600 mykey.pem
```

or provide an absolute path to the key file:

```
chmod 0600 /home/username/awskeys/mykey.pm
```

After adjusting the key permissions, log into the VM again.

Once logged into the VM, inspect various aspects of your newly launched VM with the following commands:

Operation	Command
Inspect CPU	**`lscpu`**
Inspect memory	**`free -h`**
Inspect the disks	**`df -Th`**
Inspect the OS kernel	**`uname -a`**
Inspect the OS version	**`cat /etc/os-release`**

If you are new to Linux, you are encouraged to complete tutorials 1 and 2 from Fall 2022 TCSS 562:

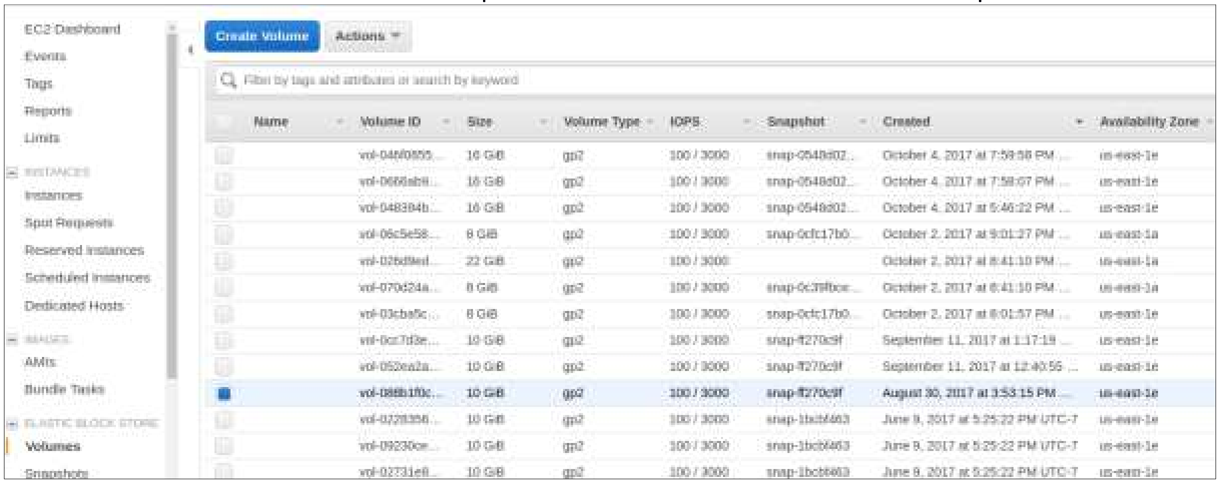
https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_1.pdf
https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_2.pdf

If wanting to learn more about ec2, you are encouraged to review/complete tutorial 3 from Fall 2022 TCSS 562:

https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_3.pdf

Stopping, and backing up your VM on Amazon:

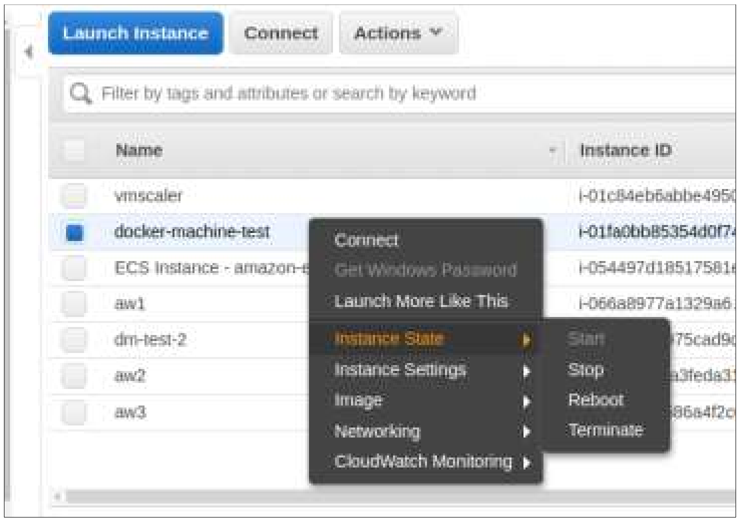
By default, the t2.micro is an "EBS-backed" instance. The t2.micro instances make use of remotely hosted elastic blockstore virtual hard disks for their "/root" volume. "EBS-backed" instances can be paused at any time. This allows the VM to be stopped, and resumed later. Billing is paused, but storage 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-04f0855...	16 GiB	gp2	100 / 3000	snap-0548d02...	October 4, 2017 at 7:59:58 PM ...	us-east-1e
	vol-0666ab8...	16 GiB	gp2	100 / 3000	snap-0548d02...	October 4, 2017 at 7:59:07 PM ...	us-east-1e
	vol-048384b...	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-f270c9f	September 11, 2017 at 1:17:19 ...	us-east-1e
	vol-052ea2a...	10 GiB	gp2	100 / 3000	snap-f270c9f	September 11, 2017 at 12:40:56 ...	us-east-1e
	vol-088b1f0c...	10 GiB	gp2	100 / 3000	snap-f270c9f	August 30, 2017 at 3:53:15 PM ...	us-east-1e
	vol-0728356...	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

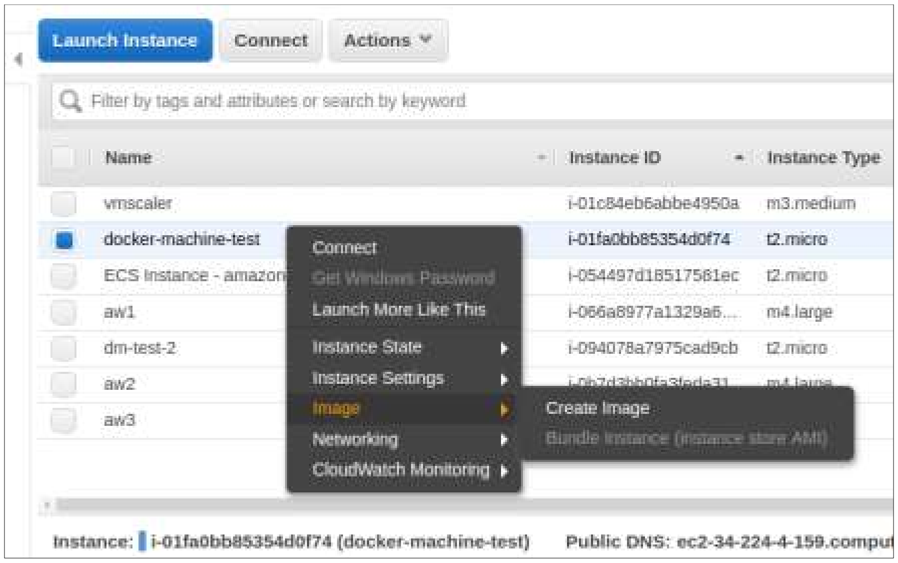
Snapshots, under "Elastic Block Store" are stored at a cost of 5 cents per month per GB. Snapshots, however, are incremental. If you snapshot a VM, you will pay for the initial storage. If you snapshot the same VM again, you will only pay for incremental differences. If a VM uses 5 GB of storage, the initial snapshot will backup 5 GB. If the VM then uses 5.1 GB of storage, when you snapshot again, you will only pay for the 0.1 GB of additional storage. This saves time and money. If you are not using a VM for a considerable time, but you would like to SAVE the data, a cost effective approach is to "**snapshot**" the root (boot) EBS volume and create an AMI. Then terminate the VM which will automatically delete the live EBS volume.

Stopping a VM when not in use



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 the public IPv4 address is new/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. (There is an option to make the image with a running VM which may be error prone.) 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 the VM from your the saved virtual machine image (called an Amazon Machine Image – AMI) can help jump start development and testing of future projects and also save cost.

THE KEY IS TO NOT LEAVE THE VM RUNNING CONSTANLY WHEN IT IS NOT IN USE…
Although, in the first year, for t2.micro, you receive a monthly 750-hour credit for running t2.micro VMs. If you have multiple active t2.micro VMs, you can exceed the FREE credit and be charged.

Next, let's install Docker on this VM.

Execute the following commands below, one by one.
If copying and pasting, it is recommended to use the MS Word document version as special characters may not copy properly from the PDF file.

Make special note of spaces, linefeed, and newline characters to ensure they appear correctly. Note that some commands are a single line, even though they appear to wrap over multiple lines. If these commands are broken over multiple lines they will fail.

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

By checking the architecture and using the **arch** variable below, Intel and Apple users obtain the correct architecture 'amd64' (for Intel) or 'arm64' (for M1 Mac).

```
sudo apt update

# this command is one line
sudo apt install apt-transport-https ca-certificates curl software-
properties-common

# this command is one line
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor
-o /usr/share/keyrings/docker-archive-keyring.gpg

# this command is one line
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo
tee /etc/apt/sources.list.d/docker.list > /dev/null

# refresh sources
sudo apt update

apt-cache policy docker-ce

# install packages
sudo apt install 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 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 script
  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:

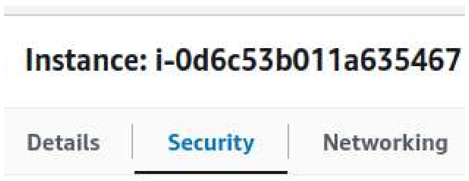
```
sudo docker run -p 8080:8080 -d --rm tomcat1
```

Check that the container is up

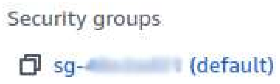
```
sudo docker ps -a
```

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:**


Instance: i-0d6c53b011a635467

Under Security details, click on your security groups ID to edit the settings:



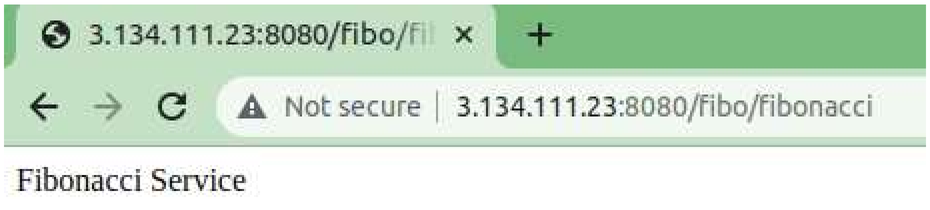
On the “Inbound rules” tab (which will already be selected), click on the “**Edit inbound rules**” button. Scroll down and click the “Add Rule” button at the very bottom LEFT of the screen:

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:



Fibonacci Service

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.

Before using testFibPar.sh, 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 0 results before testing that your client (i.e. laptop) has connectivity to EVERY SERVER using the testFibPar.sh script to confirm network connectivity.

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

Edit testFibService.sh using a text editor.
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 running testFibService.sh on your t2.micro ec2instance running Apache Tomcat:

```
./testFibService.sh
```

If your VM fails to pass the NETWORK, TRANSPORT, and APPLICATION layer tests, please revisit and resolve any issues before proceeding.

Every server should be tested in this tutorial.
Later, when setting up the load balancer, testFibService.sh should be run on the ec2 instance (VM) with the load balancer to ensure there is connectivity to each Apache Tomcat server.

Now, let’s try calling the Fibonacci service in parallel with concurrent calls from your local client (i.e. laptop):

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

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

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

Edit testFibPar.sh using a text editor.
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},234,.76600000000000000000
2,1,{"number":50000},268,.73200000000000000000
3,1,{"number":50000},266,.73400000000000000000
4,1,{"number":50000},252,.74800000000000000000
5,1,{"number":50000},240,.76000000000000000000
6,1,{"number":50000},261,.73900000000000000000
7,1,{"number":50000},270,.73000000000000000000
8,1,{"number":50000},231,.76900000000000000000
9,1,{"number":50000},264,.73600000000000000000
10,1,{"number":50000},264,.73600000000000000000
```

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 script
    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 private or 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. If using the private IPv4 address, no update should be required.

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 from your Client (i.e. laptop), use the testFibService.sh script to test that you're still able to access the Fibonacci web service.

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.

<u>**Sudo vs. non-sudo:**</u> 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 t2.micro 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 t2.micro 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 (v0.16.2).
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 !!!

- - - * * * STARTING HERE AWS CREDITS ARE REQUIRED * * * - - -

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. The c5.large spot instances will typically cost from 2 to 4 cents per hour in the us-east-2 Ohio region *(as of winter 2023)*.

Type this command, but don’t execute yet:

```
docker-machine create --driver amazonec2 --amazonec2-region “us-east-2” --
amazonec2-instance-type "c5.large" --amazonec2-ami ami-0ff39345bd62c82a5 --
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 availability zone “b” has been specified. 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. Check the availability zone by inspecting the Subnet ID under Networking. The availability zone will be in parentheses.

If you receive a spot instance error, the VM can be created as a full-price VM. This is approximately 8.5 cents/hour by issuing the command without the spot command-line options:

```
docker-machine create --driver amazonec2 --amazonec2-region us-east-2 --
amazonec2-instance-type "c5.large" --amazonec2-ami ami-0ff39345bd62c82a5 --
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

IMPORTANT: Follow the instructions carefully below to ensure proper configuration and test runs. Please ask questions if anything is not clear.

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.

It is suggested to use 'testFibService.sh' to validate that tomcat servers are working properly.
It is also suggested to tail (trace) the log files of the Apache Tomcat servers to verify that the Fibonacci service runs correctly.

To trace (tail) a log file, on the docker host, identify the container ID of the tomcat container to trace the log with '**docker ps -a**'.

Then launch a bash shell into this container:

docker exec -it <container-id> bash

Then:

tail -fn 100 /usr/share/tomcat9/logs/catalina.out

You should see success information written to the log, and not errors when the Fibonacci service is called. Success log messages look like this:

```
<6>Fibonacci POST
<6>The JSON obj:{"number":50000}
Request for fibonacci number for:50000
```

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 THE TIMING RESULTS WILL BE INCORRECT !**

Run the test script **FROM THE t2.micro VM** to perform 3 concurrent threads with 10 requests per thread:

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

Run testFibPar three times, and submit results from the third run.
You should have made 90 total runs with each configuration.

DO NOT USE YOUR PERSONAL LAPTOP OR COMPUTER TO RUN testFibPar.sh for the assignment.
Running the test client from your personal computer will involve increased latency to the cloud which may confuse the results of the performance test. It may be difficult to tell the difference the performance of different configurations because the round-trip latency from your personal computer to the cloud WILL BE VERY LONG.

In the **haproxy.cfg** file, **be sure that each server has a unique name:** like "web1", "web2", and "web3".

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 file accordingly to map to these tomcat servers on the ports:

```
# launch 3 containers on the c5.large docker host
# if you use sudo the containers will be created on the t2.micro docker host
# if not using sudo, they are created on the c5.large docker-machine host
docker run -p 8081:8080 -d --rm tomcat1
docker run -p 8082:8080 -d --rm tomcat1
docker run -p 8083:8080 -d --rm tomcat1
```

When you edit haproxy.cfg, you will need to rebuild the haproxy container, destroy the old one, and launch a new one.

In the report, describe configuration 1's VM 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 your haproxy.cfg file at the bottom of the report section.

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

<u>Configuration #2 – Co-Located Servers With CPU Thresholds:</u>
For c5.large, deploy three apache-tomcat containers on one Docker host Virtual Machine, with 66% CPU allocation.

```
# launch 3 containers on the c5.large docker host with weights
# if you use sudo the containers will be created on the t2.micro docker host
# if not using sudo, they are created on the c5.large docker-machine host
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
```

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>

<u>Configuration #3 – Separate Servers with No CPU Thresholds:</u>
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 docker-machine docker host, launch one apache-tomcat container
# no use of "sudo" below assumes use of docker-machine to launch containers
# on remote c5.large docker hosts
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.

Document History:

v.10	Initial version
v.11	clarified non-use of sudo for configs 1 and 2

AT THE CONCLUSION OF THIS ASSIGNMENT, BE SURE TO TERMINATE ALL VIRTUAL MACHINES / SPOT INSTANCES

FOR MORE INFORMATION REGARDING WORKING WITH Amazon EC2, refer to Tutorial #3 in TCSS 462/562

In particular, Tutorial 3 describes how to SAVE amazon machine images (AMI) as well as several cost savings measures:

https://faculty.washington.edu/wlloyd/courses/tcss562/tutorials/TCSS462_562_f2022_tutorial_3.pdf

<u>Docker - Helpful Hints</u>

<u>To display all containers running on a given docker node:</u>

```
docker ps -a
```

<u>To stop a container:</u>

```
docker stop <container-id>
```

For example:

```
docker stop cd5a89bb7a98
```

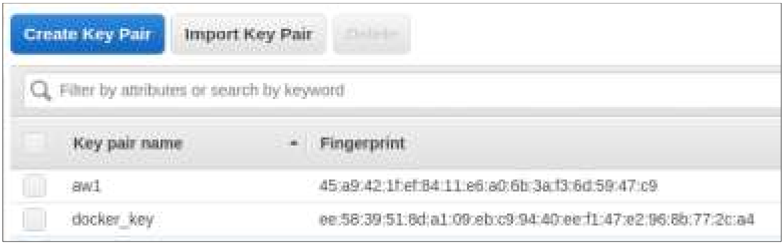
<u>Multiple docker hosts</u>

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:



Key pair name	Fingerprint
aw1	45:a9:42:1f:ef:84:11:e6:a0:6b:3a:f3:6d:59:47:c9
docker_key	ee:58:39:51:8d:a1:09:eb:c9:94:40:ee:f1:47:e2:96:8b:77:2c:a4

<u>To use a specified remote docker host created by docker-machine:</u>

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

<u>To unset the remote docker host, and work with your local docker:</u>

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

<u>Remove a docker host</u>

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
```

<u>Shell into a docker container</u>

Obtain the container id with docker ps -a.

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