



VICTORIA UNIVERSITY OF
WELLINGTON
TE HERENGA WAKA

NWEN 243: CLOUDS AND NETWORKING

2024 – T2

Dr Arman Khouzani

Project 1 Specification: Introduction to Networks

Read the instructions on the wiki about what/how to submit, and **the deadline**.

- There are of 15 questions, each carrying 1 point, for a total of 15 points. Since this project had 15% contribution to your overall grade, you can think of each question having 1% contribution to your overall grade, simple and straightforward!
- Note that each question may have multiple sub-parts (each carrying equal weight). So if a question has 4 parts, each part will have 0.25 point. Again, simple!
- Each of your answers must be accompanied by two things: **(a)** a brief but clear explanation **strictly within the word limit**; and **(b)** a screenshot. That said, one screenshot may be sufficient for one question: for each of the parts, you may choose to refer to the same screenshot if it is made clear where the answer lies (either in writing or by highlighting on the image).
- The screenshots should include the prompt, as described in the “Preparation section”. Also do not use a phone to snap a shot, use **PrintScreen**!
- You must answer the questions (and their parts) in order (sequentially) **in a single pdf file** (which is all you will submit). If you decide to skip a question or a part, you must still label it on your answer sheet and leave the answer blank.
- Include your name and student ID number on the pdf (at least on the front page).
- The answer to each question **must start on a new page**, properly marked showing which question. Parts of the same question can follow on the same page. **Do not include the question statement in your answer sheet.**
- Recall that you can ask for help from tutors during lab times. However, never, under any circumstances, share your answers. **There is zero tolerance toward plagiarism.** If you are in doubt what constitutes plagiarism, ask us!

1 Preparation

We need a “machine” (a computer) to run our tasks. While we could have asked you to run them directly on a physical machine in one of our labs, or your own machine, we are instead going to launch a “virtual machine” (VM) on the *Amazon Web Services (AWS)* cloud. This way, we can guarantee uniformity of hardware/OS/software. This will also meld well with your next projects, where you will see more features that cloud computing can offer, beyond just simple VMs. In particular, for now, we will just be using the EC2 service of AWS, standing for *Elastic Compute Cloud*.

1.1 Using AWS Learning system for the first time

- You should have received an AWS Academy invitations (from `notifications@instructure` to your ecs accounts (`your_ecs_login_name@ecs.vuw.ac.nz`). This is the email you will need to use to sign in to AWS. This email is automatically forwarded to your `myvwu` email by default – which you access through the university email (unless you have changed your forwarding.) Note: If you don’t see the email, check your junk, spam, etc, folders.

Please check out the announcement in NUKU for more details.

- AWS does not work with Safari. Please use an alternative browser. Chrome works. Firefox too. Edge seems fine too (because it is basically chromium under the hood). Try others at your peril.

Set up your AWS Academy account:

To set up your AWS Academy account, open your email invitation from AWS Academy. Choose *Get Started* (Figure 1).

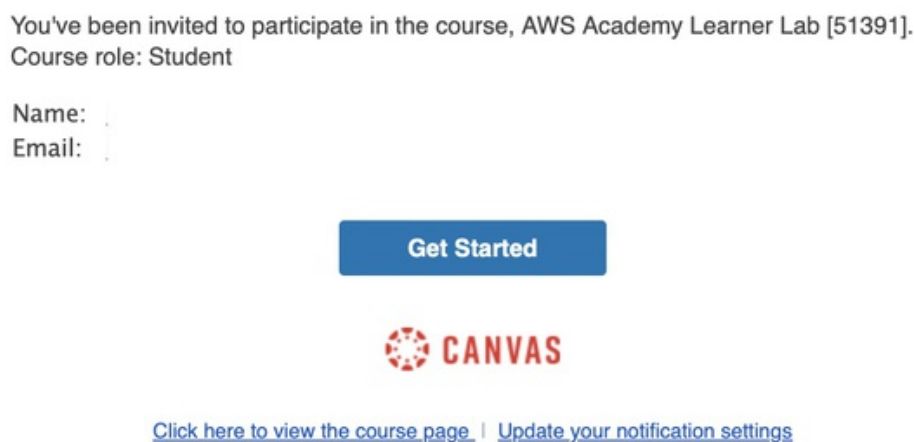


Figure 1: “Get started” page. Note that you will see a different lab number.

You will be taken to a *Canvas* screen like in Figure 2.

Choose the create option to create a canvas account. You may get an offer to transfer your email if you are currently using another, e.g., gmail in chrome. Don’t do that!

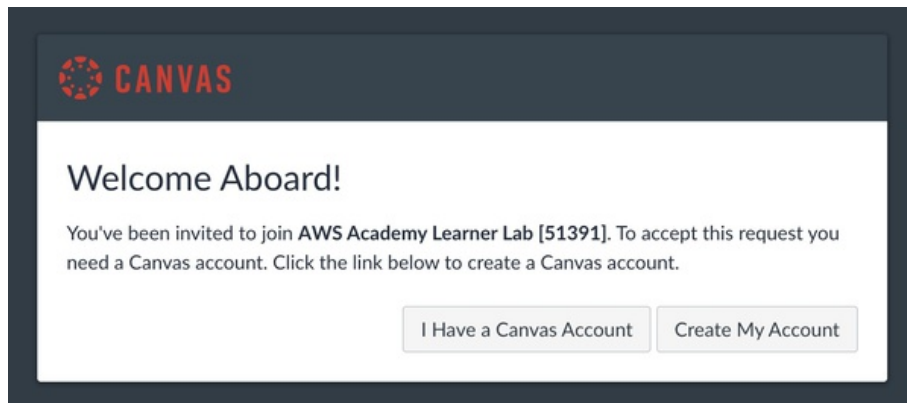


Figure 2: Choose “Create My Account”.

Sometimes, perhaps if you have previously had an account, you may not see this screen. Try using this link: <https://canvas.instructure.com/login/canvas>. You will be taken to the screen shown in Figure 3. Use the email address that you received the invitation from AWS Academy.

Figure 3: Do NOT set the password as your ECS or your VUW password.

Do not use your ECS or any of your uni passwords (or any of your passwords from anywhere) for your account. Just choose a fresh unique safe password. You will then be logged in and can go on to the course.

Once you are logged in for the first time:

Go to courses on the left menu bar and select the lab (Figure 4).

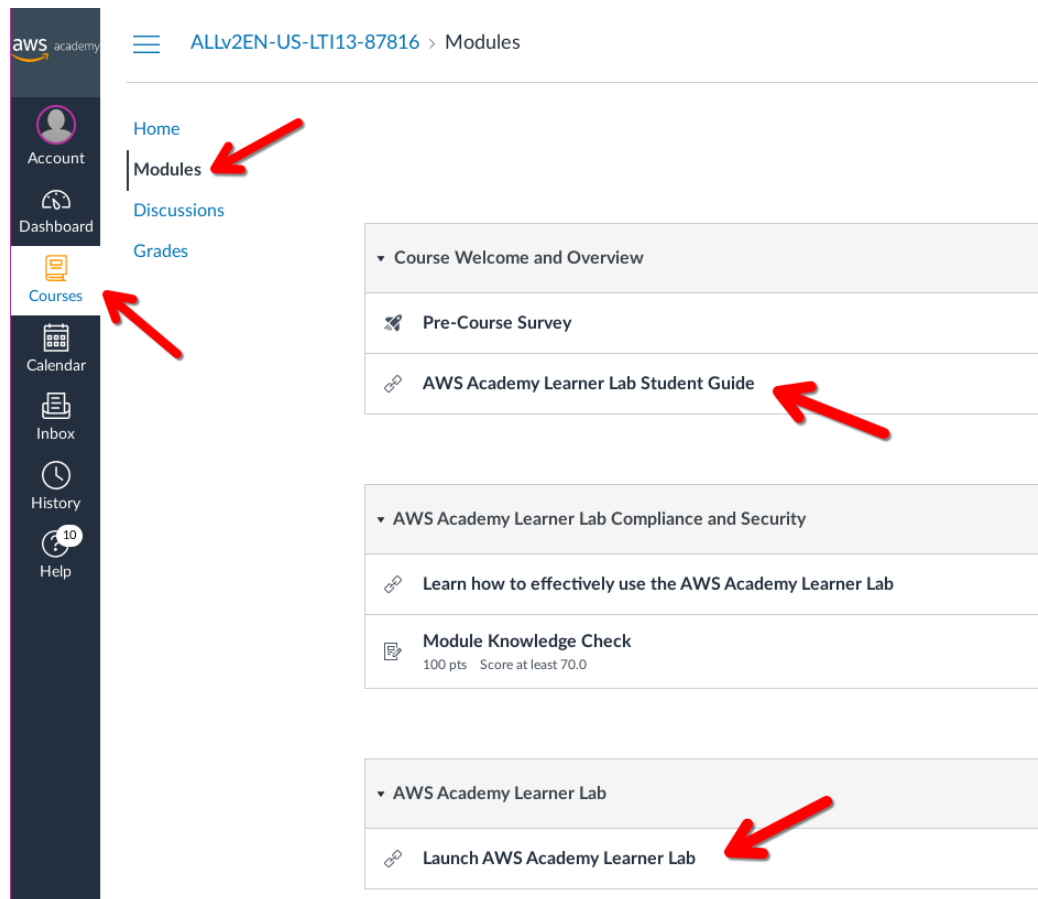


Figure 4: Courses → Modules → Launch AWS Academy Learner Lab.

Click on modules, and then read the student guide. Promise not to use Safari. Promise yourself again! Click on learner lab - there may be some delay here. Read and agree to the agreement!

The AWS Console:

See this red dot on the top left next to AWS (Figure 5)? That means we are not in the lab yet!

Click on *Start Lab* (the play button on the top right). The red dot turns yellow. AWS is “provisioning” and creating stuff for us.

Wait. Be patient. Wait a bit more! After what feels like an eternity, the yellow light should turn green, indicating that the AWS console is now ready.

Click on the AWS which now has the green light. A new pop-up tab should open (if not, check the top of the page to see if the pop-up is blocked and allow it).

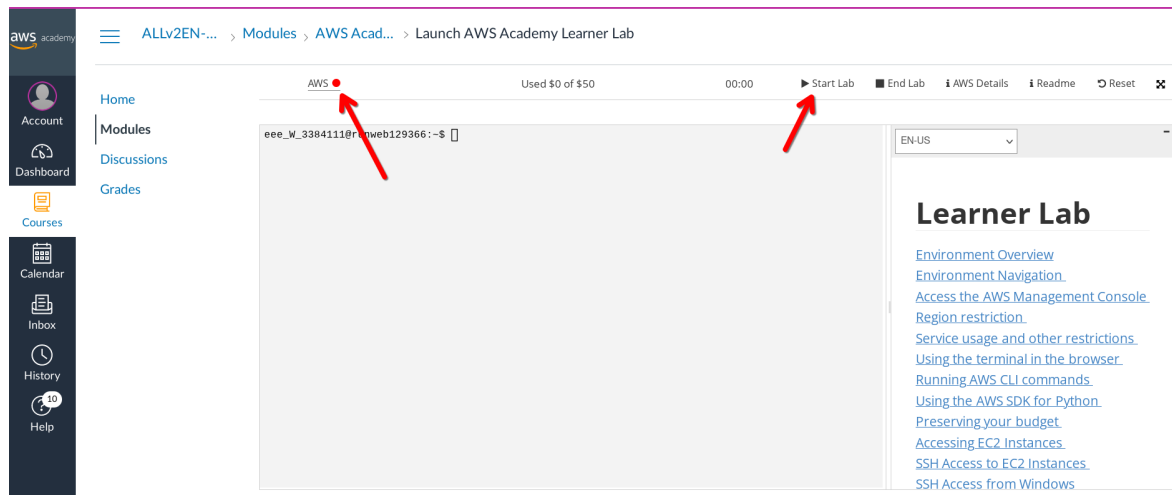


Figure 5: Click on the “Start Lab” button. The red dot turns yellow. Wait (for a loooooong time) until the yellow dot turns green.

You will be taken to the AWS’s *Console Home*, as depicted in Figure 6. This is where we will do all our NWEN243 labs (projects). Feel free to have a look around and read guides. Please do not create infrastructure until we begin the labs – that may have unintended consequences.

For subsequent logins, use this link: <https://canvas.instructure.com/login/canvas>. You may wish to bookmark this page!

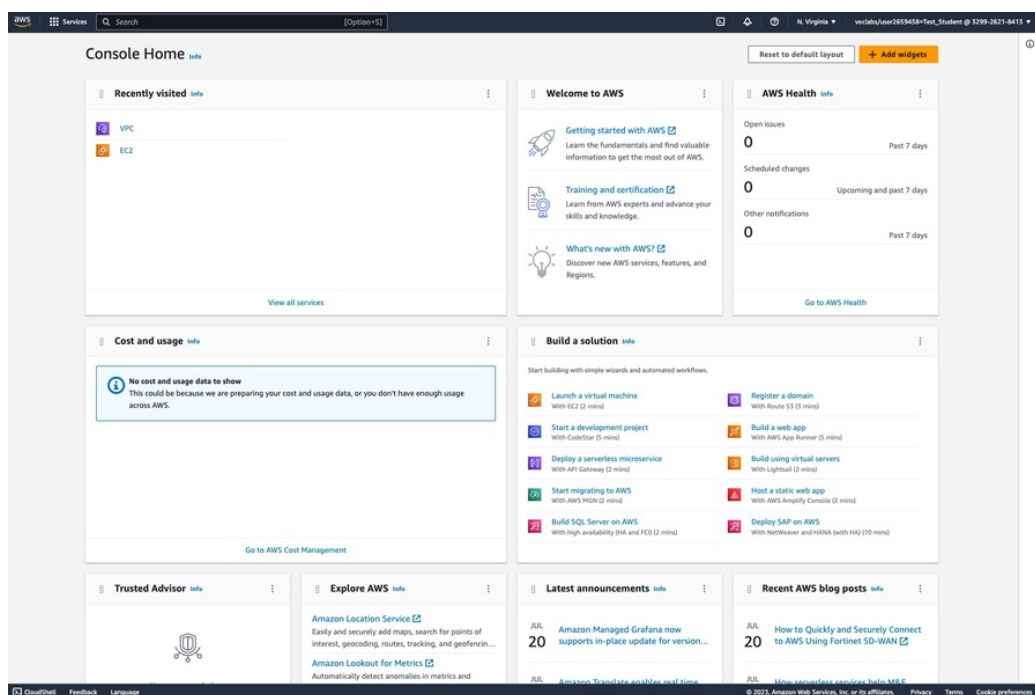


Figure 6: The AWS console home, where we start all our NWEN243 projects from. Your console page may look different from this, as the AWS console’s look gets updated over time, and this is a dated screenshot!

1.2 Launch an EC2 VM, Configure the Prompt

1. Follow the steps in the previous section to get to the AWS Console Home.
2. Click on **EC2** (this is to create a virtual machine in the AWS Cloud).
3. From the page that loads, click on the button for **Launch instance** (Figure 7).

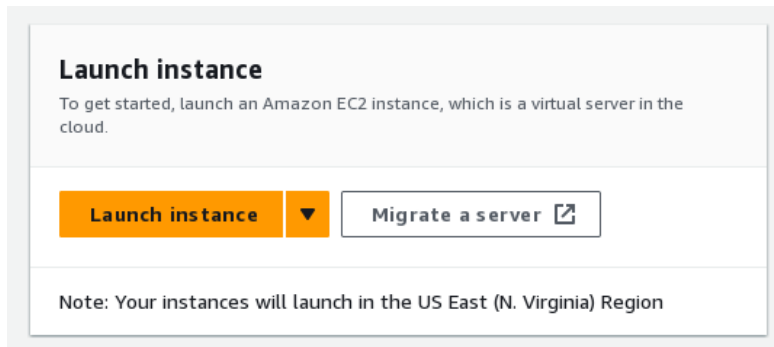


Figure 7: The button for launching an EC2 VM instance.

4. Optionally, choose a nice name for your VM instance (in the *Name and tags* filed). I (unimaginatively) went with **NWEN243_P1**.
5. For the choice of OS (Operating System), let's go with (the popular debian-based Linux distro) *ubuntu* (Figure 8). *Amazon Linux* is a good choice too, just note that it is based on Red-Hat Enterprise Linux, so the commands might be slightly different (for instance, its package manager is **yum** instead of **apt**).

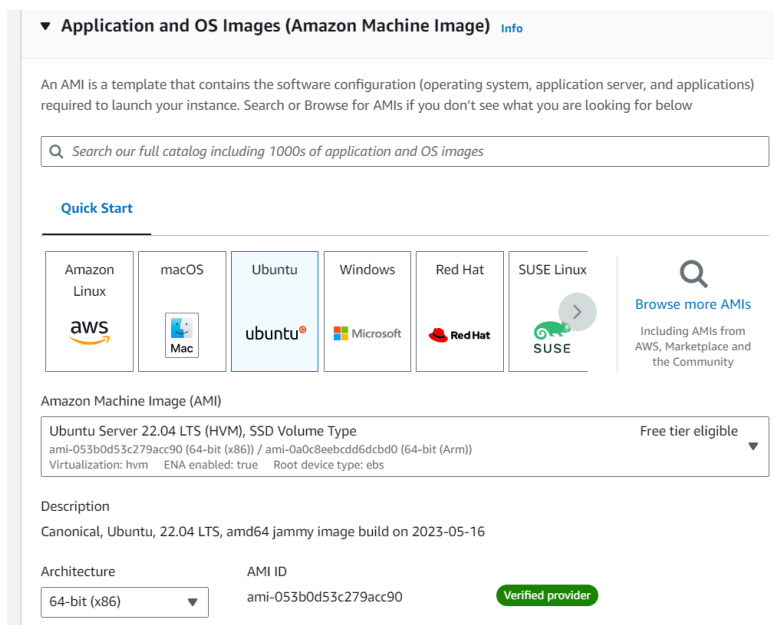


Figure 8: Choose ubuntu.

6. Under **Key pair (login)**, create new key-pair and save it somewhere safe on your local machine (Figure 9). To be exact, this is the “private” part of the key-“pair” but detail has to wait until CYBER372! You can use this ssh key to log in

to your VM from your local machine, like a professional would do! That said, AWS allows amateurs like us to connect to the VM instances (albeit laggily – is this a word?) through its own web interface as well, as we will see in the next steps.

▼ Key pair (login) [Info](#)

You can use a key pair to securely connect to your instance. Ensure that you have access to the selected key pair before you launch the instance.

Key pair name - required

Select [Create new key pair](#)

Create key pair

Key pair name

Key pairs allow you to connect to your instance securely.

aws_nwen243

The name can include up to 255 ASCII characters. It can't include leading or trailing spaces.

Key pair type

☒ RSA
RSA encrypted private and public key pair

☐ ED25519
ED25519 encrypted private and public key pair

Private key file format

☒ .pem
For use with OpenSSH

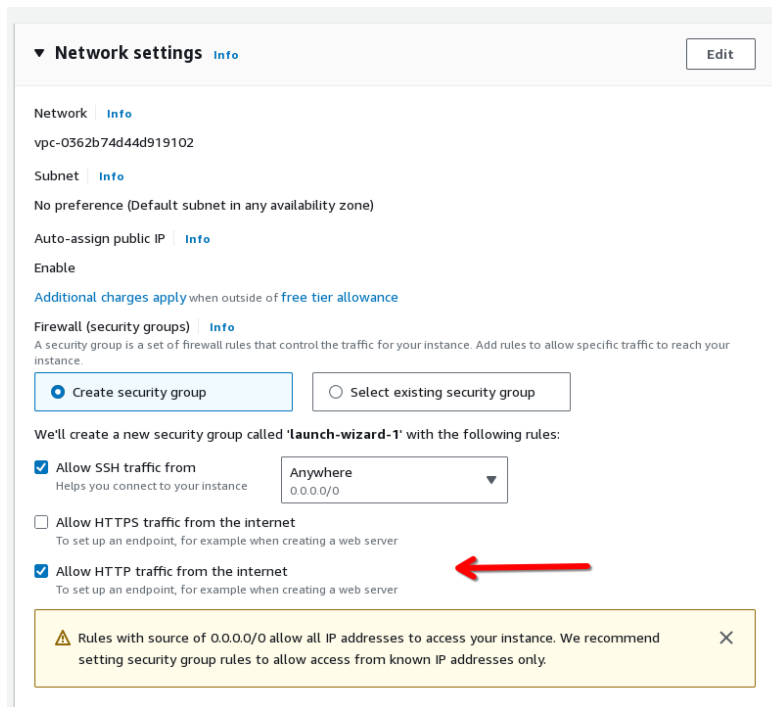
☐ .ppk
For use with PuTTY

⚠ When prompted, store the private key in a secure and accessible location on your computer. You will need it later to connect to your instance. [Learn more](#)

Cancel Create key pair

Figure 9: Create a new ssh key and save it locally with a suitable name. You can use this to ssh log-in to your instance from your local machine, or alternatively, use the slow web interface of AWS itself to connect to your VM.

7. Under **Network Settings**, check (enable) the **Allow HTTP traffic from the internet** option to allow externally initiated http requests to reach through to our VM (Figure 10). This is only relevant to last few questions, where we set up a simple web-server.
8. By default, the ubuntu VM will be created with a default username (with admin privileges) called **ubuntu**. Although we can change this, let's leave it as is.



▼ Network settings [Info](#) Edit

Network [Info](#)
vpc-0362b74d44d919102

Subnet [Info](#)
No preference (Default subnet in any availability zone)

Auto-assign public IP [Info](#)
Enable
[Additional charges apply](#) when outside of [free tier allowance](#)

Firewall (security groups) [Info](#)
A security group is a set of firewall rules that control the traffic for your instance. Add rules to allow specific traffic to reach your instance.

☒ Create security group ☐ Select existing security group

We'll create a new security group called 'launch-wizard-1' with the following rules:

☒ Allow SSH traffic from Anywhere
0.0.0.0/0
Helps you connect to your instance

☐ Allow HTTPS traffic from the internet
To set up an endpoint, for example when creating a web server

☒ Allow HTTP traffic from the internet ←
To set up an endpoint, for example when creating a web server

⚠ Rules with source of 0.0.0.0/0 allow all IP addresses to access your instance. We recommend setting security group rules to allow access from known IP addresses only. ✕

Figure 10: Allow incoming http requests to your VM. This is relevant to Q13 – Q15.

9. Make sure you have not touched any other settings. Ready? Launch!

10. Congratulations, we just got our first VM in the cloud (Figure 11).

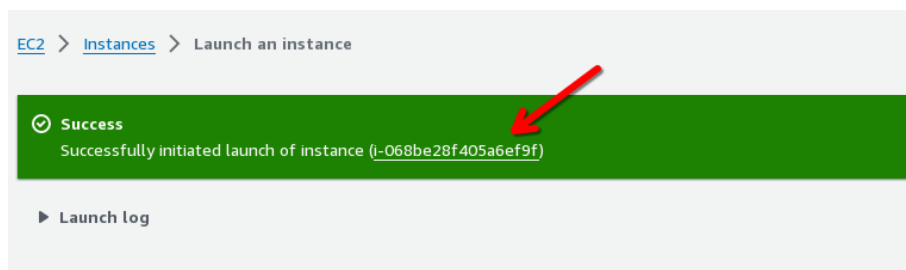
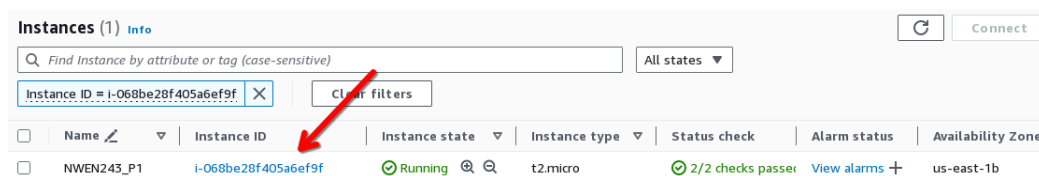


Figure 11: Successful launch of our Ubuntu VM.

11. Now let's connect to it: click on the ID of the instance. You will be taken to a page like shown in Figure 12. From there, click on the ID of your VM one more time.



Instances (1) [Info](#) Refresh Connect

Find Instance by attribute or tag (case-sensitive) All states ▼

Instance ID = [i-068be28f405a6ef9f](#) ✕ Clear filters

	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone
<input type="checkbox"/>	NWEN243_P1	i-068be28f405a6ef9f	Running	t2.micro	2/2 checks passed	View alarms	us-east-1b

Figure 12: The list of our VM instances – which should so far show only one VM!

12. We are taken to the page for controlling our VM instance. From here, simply click on **Connect** (Figure 13). If the connect button is not available, do the following: under **Instance state**, choose **Start instance**.

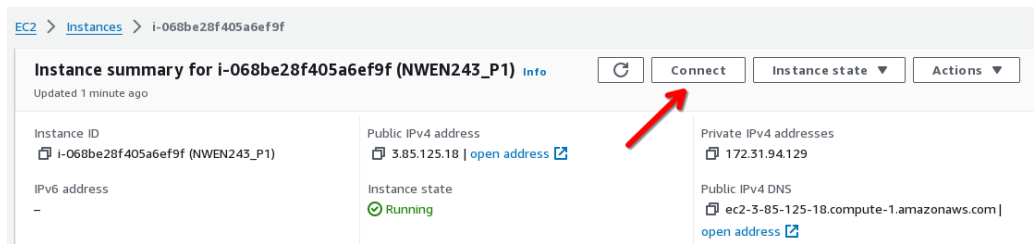


Figure 13: Click on **Connect**.

13. The page that loads gives us multiple options. Leave it be the default choice (**Connect using EC2 Instance Connect**). Finally, click **Connect** (Figure 14).

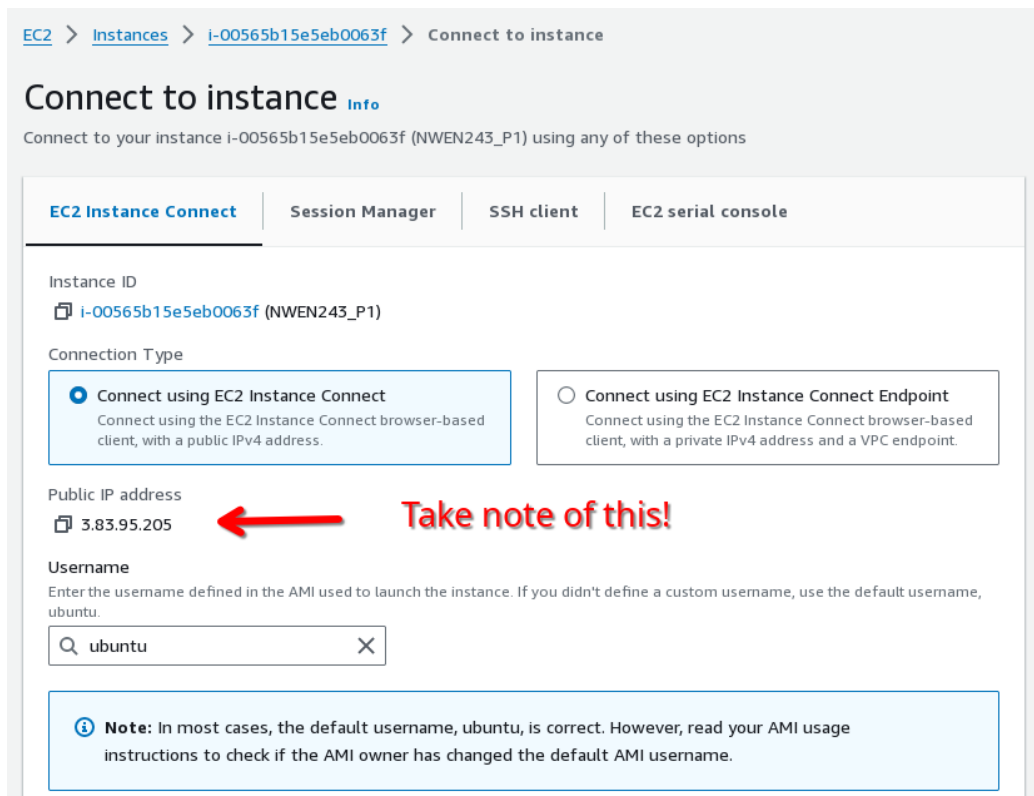


Figure 14: Take note, and **Connect**!

14. On a new tab, you should be taken to your VM (finally), a nice bash terminal.
Before we get to the questions though, let's do an **IMPORTANT** configuration (for grading purposes!). Run the following command in the terminal to customise your prompt (replace `arman` with your own ECS username). This is so that your screenshots will be unique.

Throughout this manual, instead of copy/pasting instructions or code, you should do the safer option of typing it out instead – because sometimes characters get copied as non-ascii, which look deceptively like their ascii counterparts, leading to obscure errors!

```
export PS1="[\d \t] arman@\h: \w\$ "
```

Explanation:

- \d: date
- \t: time
- arman: replace this with your ecs username!
- \h: hostname
- \w: working directory
- \\$: just a dollar sign character

Figure 15 shows the result of this command in my terminal, as an example.

```
Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-80-16:~$ export PS1="[\d \t] arman@\h: \w\$ "
[Sun Jul 21 05:07:46] arman@ip-172-31-80-16: ~$ █
```

Figure 15: For your screenshots in your answers, you **MUST** have customised your prompt to show date, time, YOUR username and the host machine (the VM name), similar to the above.

Note that this affects only the current terminal session. Next time you log in, you need to re-issue the above command. Or better yet, do the next step, so you don't have to.

15. We can put our prompt customisation into **bashrc** file, which gets automatically executed at each startup of ubuntu, so that we will get our requested prompt customisation upon all future logins. We can use a simple terminal-based text-editor called **nano** (or **vim**, if you are either a professional, a geek, or a masochist!):

```
nano ~/.bashrc
```

Scroll down to the end. Then on a new line, type (replacing **arman** with your own ECS username):

```
export PS1="[\d \t] arman@\h: \w\$ "
```

Press **Ctrl+S** to save, then **Ctrl+X** to exit. Done!

Some general hints about our Linux terminal:

- By pressing the up-arrow key you can go through your previous commands.

- Hitting the **tab** key in the middle of a command auto-completes; where there can be multiple options, hit the **tab** key twice to be shown a list of suggestions.
- You can access the manual of each command using the **man** command. For instance, **man man** will display the manual for the **man** command! You can then use the up and down arrow keys, forward slash “/” for keyword search (and then **n** and **Shift+n** for going next and previous among the found incidents). Finally, you can hit **q** to quit, and get back to the command prompt.
- In a Linux terminal, **Ctrl+C** does not copy! instead, it sends a **SIGINT** (interrupt signal) to the running foreground process, requesting it to be gracefully terminated. So for copy pasting, use the mouse right-click. Or if you prefer keyboard, use **Ctrl+Insert** and **Shift+Insert** instead of **Ctrl+C** and **Ctrl+V** (or just use the mouse right-click).

Alternative connection: SSH login

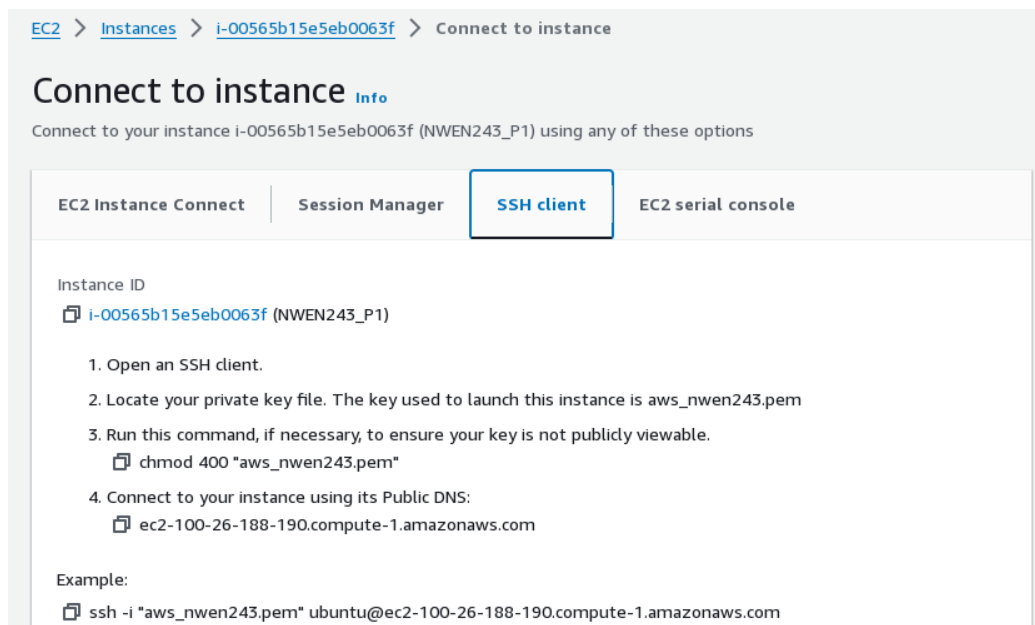


Figure 16: The *SSH client* tab.

If you find the web interface to the command line slow, you can use a standard ssh login. Just open a terminal on your local machine and follow the instructions on the *SSH client* tab (Figure 16). Of course the public DNS of your instance will be different.

Note: You are not supposed to provide any snapshots for this “preparation” part. However, in all of the snapshots of your answers to the questions, the prompt should be visible. This is one way we check this is your own work. Another way is that, in fact almost all of your answers will be unique to you since your VM will have a unique public IP address. As a reminder, your submitted work must be your own, plagiarism will not be tolerated.

Subsequent launching of your VM:

If you would like to come back to resume working on your project tasks, you can follow these steps:

1. Log in to AWS academy <https://canvas.instructure.com/login/canvas> using the credentials you created when you signed up (again, as was detailed before).
2. Click on **Courses** (on the left panel), then **Modules**, then **Learner Lab**.
3. Then **Agree**, and **Start Lab** (with the “play” button on the top right) – Figure 17.

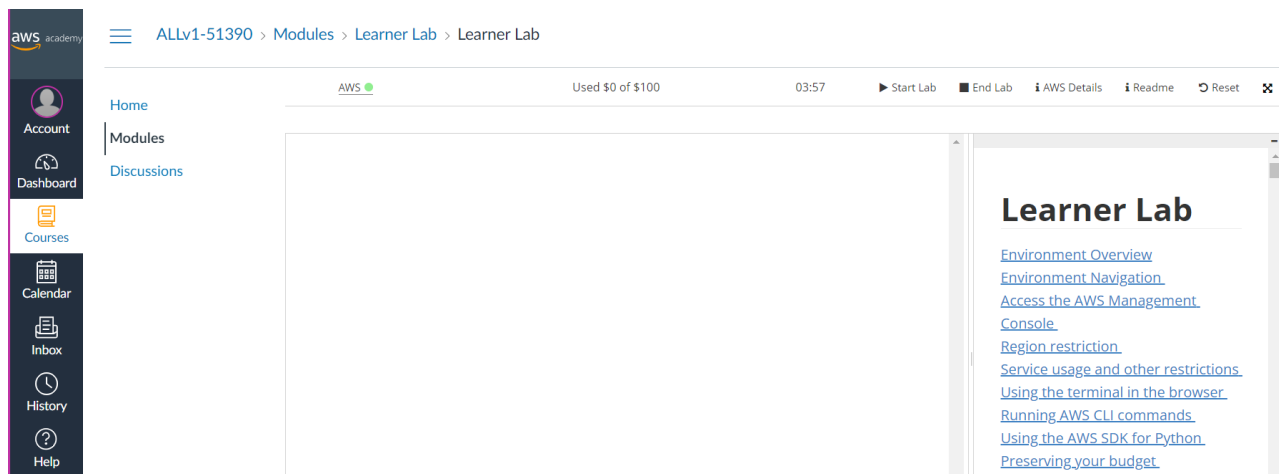


Figure 17: Click on **Start Lab** button – not **Reset**!

4. Wait for the provisioning (the red **AWS** icon on the top left to go green).
5. An eternity later, when the icon eventually turns green, click on it!
6. A new tab opens, on which, find the link to **EC2**. Click.
7. Don't launch a new instance. Look for your previous VM instance under “**Instances (running)**”. Once you find it, click, and Bob's your uncle.

If your VM terminal is no longer responsive.

After some inactivity time, you may notice that your web-based access to your VM terminal is non-responsive. Just try to refresh, or reconnect to it as described before (Figure 13).

Just note that if the “Connect” button in Figure 13 is disabled, it could be that the machine state is “stopped”. All you need to do is from the button next to it labelled “Instance state” choose “start” from its drop-down menu.

However, also recall that each session lasts for 4 hours. After which, you need to **Start Lab** again (Figure 17) for a new 4-hour session. Your VM instance will still be there (unless you click on **Reset** by mistake).

2 Questions

Q1. [20 words max] Issue the following command in your VM's terminal:

```
ip address show
```

Interpret the output and answer the following questions:

- (a) What is the name of the network interface controller of your VM?¹
- (b) What is its MAC address expressed in hex?
- (c) What is the MAC address expressed in binary (actual bits)?
- (d) What is the length of the MAC address (in number of bits)?
- (e) Recall that the network interface controller keeps listening to the wire, identifies the beginning of each dataframe, checks the destination MAC addresses in its header, and if it matches the MAC address of itself (so it knows this dataframe was meant for it), then it removes its layer 2 header and passes the payload up to the OS. However, there is another special purpose MAC address that is always picked up by any node as well, known as *broadcast* MAC address. From the output, identify what that is.

Q2. [20 words max]

- (a) Find out the private and public IPv4 addresses of your VM.
- (b) Find out the private IPv6 of your VM (note: unless you had requested at launch time, your instance is not assigned a public IPv6 by default.)
- (c) How long is your public IPv4 (in number of bits)?
- (d) Express your public IPv4 address in binary (in bits).
- (e) How long is your private IPv6 address (in number of bits)?

Q3. [20 words max] Let's look at the output of

```
ip address show
```

one more time.

- (a) Looking at the private IPv4 address, what part of it is the network portion, and which part is the host portion?
- (b) What is the range of private IPv4 addresses that belong to the same LAN?
- (c) How many distinct IPv4 addresses can there be in this LAN?

Q4. [30 words max]

- (a) What is the netmask for the (private) IPv4 address?

¹Ignore the interface whose name starts with "lo": that is a special purpose dummy interface called *loopback*, which your computer uses to send packets to itself(!), e.g., when debugging programmes that use the internet.

- (b) From the output of the `ip address show`, extract the (private) broadcast IPv4 address.
- (c) Explain how the broadcast IPv4 address could be obtained from your VM's IPv4 address and the netmask.

Q5. [40 words max]

- (a) Issue the following command in the VM terminal:

```
ip route list
```

Interpret the output, especially the line starting with the keyword *default via*.

- (b) Now issue the following command:

```
ip neighbour show
```

In the output of the command, you should be able to identify an IP and a MAC address. Whose IP and whose MAC address are these? You should in part describe what the purpose/necessity of this information is. That is, what does our VM use the knowledge of this MAC and IP address for?

Q6. [50 words max]

- (a) Ping **youtube** from your VM. That is, issue the following command from your VM terminal:

```
ping www.youtube.com
```

From the result, identify what destination IP address is chosen for `www.youtube.com`. Let's call this **IP1**.

- (b) Let's repeat the experiment, but this time from a "different" machine. This could be your "local" machine if you know how to ping from it. Or, for convenience, you can use a web-service that provides **ping** service like <https://ping.eu/ping/>. Note that the ping messages are sent from their server, which is located in Germany. What IP address do you see for **youtube** when you ping from this different machine than the VM (either your local machine or from the <https://ping.eu/ping/> server)? Let's call this **IP2**.
- (c) Most likely, you did not get the same IP addresses. That is, **IP1** \neq **IP2**. Using **ping**, estimate four round trip latency (RTL) values:
 - **RTL_VM_IP1**: from your VM to **IP1**
 - **RTL_VM_IP2**: from your VM to **IP2**
 - **RTL_local_IP1**: from the "different" machine to **IP1**
 - **RTL_local_IP2**: from the "different" machine to **IP2**.
- (d) Use these estimated RTLs to throw some light on why **IP1** is different from **IP2**.

Q7. [50 words max] The **tracert** programme is a network diagnostic tool used to track the path packets take to a destination. It works by sending a series of packets with increasing values of Time-to-Live (TTL) in their IP header, which causes the packets to expire at successive network hops along the route to the destination (genius!). Each router or network device that the packets encounter along the path should normally send back an ICMP (Internet Control Message Protocol) “Time Exceeded” message to the source. This message includes information about the router’s IP address. Traceroute uses this information to display the IP addresses of the routers (or network hops) in the path to the destination. While we can use the **tracert** programme from the terminal of our VM, let’s use a web service that provides some visuals too: <https://www.ip2location.com/free/tracert>. It does not matter which server you choose.

- (a) Find a path from a server of [ip2location.com/free/tracert](https://www.ip2location.com/free/tracert) to **IP1** and **IP2** (found in the previous question). Comment on the location of **IP1** and **IP2**.
- (b) Find a path from a server of [ip2location.com/free/tracert](https://www.ip2location.com/free/tracert) to **wgtn.ac.nz** and **barretts.ecs.vuw.ac.nz**. Describe your findings.

Q8. [50 words max] Let’s investigate some ARP! Remember that ARP (Address Resolution Protocol) was a protocol that helps translate layer 3 addresses to layer 2 addresses within a local network segment. But what on earth does that mean in human language?! That’s the topic of this question. From the terminal (of your VM), run the following command to capture the ARP packets:

```
sudo tcpdump -v -nn arp
```

Note that we are not running the process in the background (no ampersand at the end). So while this process is running, the command line is not accessible.

Wait for a couple of minutes, you should see some packets, wait till you see at least 6, then press **Ctrl+C** to terminate the **tcpdump** process. Now answer the following questions:

- (a) Identify a request-reply pair, and in plain English(!), explain what their meaning are. Specially specify who the sender is and what is being asked/answered (i.e., the purpose of each packet).
- (b) These pairs of request-replies seem to be repeating periodically. Explain what is the sense of asking the same ARP request periodically.
- (c) By looking at the timestamps (reported at the beginning of each line of the output of **tcpdump**), estimate how often these ARP requests are being sent (just make sure they are sent from the same device!).

Q9. [60 words max] Let’s investigate DHCP (Dynamic Host Configuration Protocol). As we all remember(!) the main purpose of DHCP is to give layer 3 address to hosts that join a local network.

- (a) The fact that our VM has a local IP address means that the DHCP has done its job already! But we can access the history of system interaction logs. In modern Linux, it is by calling **journalctl** (or alternatively, seeing the content

of the log file at `/var/log/syslog`). So in order to investigate previous DHCP interactions, we can search for a relevant keyword. A powerful programme in Linux for searching for text patterns is **grep**.

Run the following command from your VM terminal:

```
journalctl | grep -i 'dhcp'
```

The `|` (pipeline) is saying pipe (i.e., pass) the output of the first command/programme to another command/programme and the `-i` parameter (or equivalently `--ignore-case` means ignore-case (capital or small letters).

Then identify the 4-way communication of DHCP interaction between our VM and the DHCP server. Make sure they belong to the same session (by checking the timestamp+date of the log messages). Identify the IP address of our DHCP server and the duration of the lease.

- (b) The result of DHCP interaction is a “lease”, the detail of which can be accessed by issuing the following command:

```
netplan ip leases enX0
```

where you should replace **enX0** with the name of the network interface you found for your machine in question 1.

There you should see that DHCP (“optionally”) provides more than just leasing you an IP address for a certain duration. Briefly describe the main and optional information that is provided in your DHCP lease.

Q10. [60 words max] This experiment takes a small setup:

- First, let’s have our packet capturing program, **tcpdump**, running in the background, set to capture packets whose either source or destination port number is 53:

```
sudo tcpdump -nn -v port 53 > tcpdump.out 2>&1 &
```

Make sure to not to omit:

- nn**: (single dash, no space after, two small ‘n’s): this is for no-names, just the IP addresses are enough.
 - v**: (single dash, no space after, small ‘v’): for verbose, so we get more information about the captured packets;
 - > **tcpdump.out 2>&1**: this is redirecting the output of the command (as well as any error or information) to an output file which I named **tcpdump.out** (choose whatever name you like, even the file extensions are arbitrary in Linux!).
 - &**: the ampersand runs the programme in the background, returning the prompt back to us to execute more commands, which we need for the next step!
- Now let’s visit a webpage. But we only have a text-only terminal. Worry not! We can use the versatile **curl** programme (it stands for Client for URL). Let’s fetch the home-page of the uni:

```
curl --silent https://www.wgtn.ac.nz/ > /dev/null
```

That **--silent** parameter (with two dashes) suppresses the download status message. We are also sending what we are downloading to a special place called **/dev/null**, which discards all data written to it! So we are essentially discarding the downloaded page.

- We should now terminate **tcpdump** to make it stop capturing packets (and not continue running in the background):

```
sudo killall tcpdump
```

- Now open the output file to inspect what packets were captured as a result of our **curl** command:

```
cat tcpdump.out
```

You should see that at least 2 and potentially 4 packets captured. (Well, there could be more packets captured, as our OS might also be generating this kind of traffic in the background – naughty OS! In such a case, either run the experiment again, or just identify and ignore the packets that are not relevant to us!).

- (a) By looking at the contents of the packets (expressed in a summarised human-readable way), identify each packet. In particular, for each one, specify who is sending it to whom, and what is its purpose.
- (b) What transport layer protocol is used by them? (this information is displayed if you have not forgotten to add the verbose option (**-v**) to your **tcpdump** command. Comment on why, amongst our two main options for the transport layer, this is chosen, i.e., why is it the appropriate choice for their purpose?

Q11. [60 words max] Let's continue having fun (learning) with **tcpdump**!

- First, let's make sure our previous **tcpdump** is indeed terminated:

```
sudo killall tcpdump
```

- Also let's remove the previous file to capture some new packets:

```
rm tcpdump.out
```

- We are going to run a new instance of **tcpdump** in the background tuned to capture a different type of traffic, those with either source or destination port of 443:

```
sudo tcpdump -S -nn port 443 > tcpdump.out 2>&1 &
```

Note that we no longer need the “verbose” option for this question.

- Again, let's retrieve the homepage of the uni, using **curl**:

```
curl --silent https://www.wgtn.ac.nz/ > /dev/null
```

- As in the previous question, let's terminate the `tcpdump` process to stop the packet capturing:

```
sudo killall tcpdump
```

- Finally, let's inspect the captured packets:

```
cat tcpdump.out
```

You should now see many more packets captured (around 40 or so).

For each of the first 3 packets, identify the following:

- (a) Source and destination entities (who is sending that packet to whom);
- (b) Source and destination port numbers;
- (c) TCP flags;
- (d) The purpose of that packet (what information does that packet deliver).

Q12. [100 words max] This is a follow-up to the previous question. Looking at the first 6 packets, explain how the sequence number and acknowledgement numbers are calculated.

Hint: The information about the size of each packet (in number of bytes) is also provided in the `tcpdump` output.

By default, `tcpdump` shows “relative” acknowledgement numbers. It is simpler to answer this question with “absolute” acknowledgement numbers. For this reason, we ran the `tcpdump` with the `-S` parameter (or equivalently `--absolute-tcp-sequence-numbers`).

Q13. The remaining questions are about socket programming. Well, a specific application of it: a simple web-server. Now you might be thinking why use socket programming where there are powerful and feature-rich libraries for web-servers handling http requests, and for web-clients dealing with sending http requests. After all, web-servers and web-clients are such standard applications of the Internet. However, we would not be learning how things work under the hood, so that we can use socket-programming for our next innovative killer-app!

The recommended language is `java`. But if you prefer a different language, like `C/C++`, `python`, or `Go`, feel free, with the following caveats:

- Support for other languages by our tutors may be limited.
- Irrespective of the choice of language, you are not allowed to use a high level library for handling or even sending http requests. You should still be using socket programming only, for any communication over the network.

This task needs a bit of setup:

1. Install **Nginx**: in the terminal of your VM, enter the following:

```
sudo apt update && sudo apt install nginx -y
```

If you are wondering why we are installing **Nginx**, it is because we do not want our web-server application (that we are going to write) to run with heightened privileges (e.g. run it as root). That is, we do not want it running on the privileged port of 80 (the default port for http), and handling all external requests directly (listening to 0.0.0.0). Instead, we want to run on localhost (127.0.0.1) and on an unprivileged port like 8080 (anything above 1024 really). While that is fine for testing and development, how can we then let outsiders access our amazing web app? But having a software like **Apache web server** or **Nginx** to sit in between, passing the client requests to port 80 of our public IP address to port 8080 of our localhost, where our programme is listening to.

Now why **Nginx** as opposed to **Apache**, because it is more lightweight and resource-efficient, and simpler to configure.

To test that you have installed **Nginx** correctly, just enter the public IP of your VM in the address field of your web-browser (on your local machine). There you should see the default **Nginx** showing, as in the following:

Welcome to nginx!

If you see this page, the nginx web server is successfully installed and working. Further configuration is required.

For online documentation and support please refer to nginx.org.
Commercial support is available at nginx.com.

Thank you for using nginx.

2. Next, we should configure **Nginx** by creating a new configuration file. In your VM terminal, issue:

```
sudo nano /etc/nginx/sites-available/my_web_server
```

which opens an empty file. Enter the following in it:

```

server {
    listen 80;
    server_name 52.203.40.84;

    location / {
        proxy_pass http://127.0.0.1:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        chunked_transfer_encoding off;
        proxy_buffering off;
    }
}

```

Just replace 52.203.40.84 with the public address of your own VM.

Finally, hit **Ctrl+S** to save and **Ctrl+X** to exit the **nano** text-editor.

Some explanations are called for:

listen 80; Nginx should listen for incoming connections on port 80 (default for HTTP).

server_name your_vm's_public_IP; Requests to this IP address will be directed to this server block.

location / specifies for which url path the rule (followed in braces) applies. In this case, this is the root url, that is, requests to **http://your_vm's_public_IP/**

proxy_pass http://127.0.0.1:8080; tells Nginx to forward incoming requests to the specified backend server at **http://127.0.0.1:8080**, which is port 8080 on localhost. As of now, there is no program listening there, but that's exactly what our own web-server will do.

Host \$host; : This sets the Host header to the original value from the client's request (otherwise, this will be that of Nginx, since the messages will be coming from it).

X-Real-IP \$remote_addr; sets the **X-Real-IP** header to the client's IP address. This will come handy later!

X-Forwarded-For \$proxy_add_x_forwarded_for; This is another header place where the real IP address of the client is appended when request go through proxies or load-balancers, etc. One difference with **X-Real-IP** is that this is a comma separated list, so if a request goes through multiple proxies, the IP of all of the intermediaries will be appended here.

chunked_transfer_encoding off; Chunked transfer encoding is a nice feature of Nginx which is enabled by default. It allows sending data in small, dynamically sized chunks, which is useful for streaming large responses or when the content length is unknown. However, since our simple webserver programme is not going to support it, we should disable it.

proxy_buffering off; By default, Nginx buffers the response from the backend server (our web-server programme) before sending it to the external client. This directive disables proxy buffering, and now Nginx will send the response from our backend webserver directly to the external client without buffering it.

3. Now enable the configuration by issuing the following in your VM's terminal:

```
sudo ln -s /etc/nginx/sites-available/my_web_server /etc/nginx/sites-enabled/
```

4. Restart **Nginx** for this configuration to take effect:

```
sudo service nginx restart
```

We are now ready to create and run our simple web-server. For your convenience, a template code is provided for you:

```
import java.io.*;
import java.net.*;

public class SimpleWebServer {
    public static void main(String[] args) {
        int port = 8080;
        try {
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server running at http://localhost:" + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Connection from " + clientSocket.
getInetAddress());

                handleRequest(clientSocket);

                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void handleRequest(Socket clientSocket) throws IOException {
        OutputStream outputStream = clientSocket.getOutputStream();
        PrintWriter out = new PrintWriter(outputStream, true);

        out.println("HTTP/1.1 200 OK");
        out.println("Content-Type: text/html");
        out.println();

        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Hello, World!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hello, World!</h1>");
    }
}
```

```

        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}

```

Or if you prefer to be more stylish(!):

```

import java.io.*;
import java.net.*;
import java.util.Scanner;

public class SimpleWebServer {
    public static void main(String[] args) {
        int port = 8080;
        try {
            ServerSocket serverSocket = new ServerSocket(port);
            System.out.println("Server running at http://localhost:" + port);

            while (true) {
                Socket clientSocket = serverSocket.accept();
                System.out.println("Connection from " + clientSocket.
getInetAddress());

                handleRequest(clientSocket);

                clientSocket.close();
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    private static void handleRequest(Socket clientSocket) throws IOException {

        // Prepare the HTTP response
        String httpResponse = "HTTP/1.1 200 OK\r\n";
        httpResponse += "Content-Type: text/html\r\n";
        httpResponse += "\r\n";
        httpResponse += "<!DOCTYPE html>\r\n";
        httpResponse += "<html>\r\n";
        httpResponse += "<head>\r\n";
        httpResponse += "<title>Hello, World!</title>\r\n";
        httpResponse += "<link rel=\"stylesheet\" href=\"https://cdn.jsdelivrivr.
net/npm/bootstrap@5.3.1/dist/css/bootstrap.min.css\">\r\n";
        httpResponse += "</head>\r\n";
        httpResponse += "<body>\r\n";
    }
}

```



```

        httpResponse += "<div class=\"container\">\r\n";
        httpResponse += "<div class=\"jumbotron mt-5\">\r\n";
        httpResponse += "<h1 class=\"display-4\">Hello, World!</h1>\r\n";
        httpResponse += "</div>\r\n";
        httpResponse += "</div>\r\n";
        httpResponse += "</body>\r\n";
        httpResponse += "</html>\r\n";

        // Send the HTTP response to the client
        OutputStream outputStream = clientSocket.getOutputStream();
        outputStream.write(httpResponse.getBytes());

        outputStream.close();
    }
}

```

1. From the terminal, create a new java file:

```
nano SimpleWebServer.java
```

In the empty file that opens, enter the code. Then save (**Ctrl+S**) and exit (**Ctrl+X**).

2. Compile your java file:

```
javac SimpleWebServer.java
```

Note: You may need to install java development kit first:

```
sudo apt update && sudo apt install openjdk-8-jdk-headless -y
```

3.

```
java SimpleWebServer
```

4. Check that your web-server is indeed working by entering the public IP address of your VM in a browser. You should see the **Hello, world!** message.

Your task is simple: edit the code so that instead of **Hello, world!**, a message containing your own name is on the page, e.g. **Hello, my name is Arman Khouzani!**

For this question, you don't need to provide your modified code, just a screenshot. Just make sure the screenshot of the web-page clearly shows your name (as asked above) as well as the public IP address of your VM (in its address bar).

- Q14. Add the following feature to our web-app: each time a client visits our site, their public IP address should be displayed to them. Note that this should be on the web-page, not printed to the terminal of the VM, as, hopefully, they do not have access to the terminal!

Note: if your first attempt only shows **127.0.0.1**, remember that we are using **nginx** as a reverse proxy, instead of directly receiving the external requests. But

recall that we are asking **nginx** to forward the IP of the requests in the header, specifically labelled **X-Real-IP**, before passing them to the **localhost**, i.e., **127.0.0.1**, which is the default IP address of the loopback dummy interface.

Provide both your code and a screenshot of the webpage as described in the previous question.

Hint: The http headers are all part of the raw http request message. So you can read the raw request message, parse it, and find the header we are interested in using standard programming. Here is a blueprint code for you:

```
private static String getHeader(Socket clientSocket, String headerName)
throws IOException {
    InputStream inputStream = clientSocket.getInputStream();
    BufferedReader in = new BufferedReader(new InputStreamReader(inputStream
));
    String line;
    while ((line = in.readLine()) != null) {
        // your code logic comes here,
        // where you find the header and return it.
    }
    return null; // if the requested header could not be found.
}
```

Note that you need to call this function (with appropriate arguments) from the code (e.g. from the **handleRequest** function).

If I were you, I would initially just print what the raw incoming http request message contains (an http get request only has a header and no body). You can do this by putting **System.out.println(line);** inside the **while** loop. Once you are comfortable what each "line" of the header contains, it should become clearer what needs to be done.

Q15. As the next step, add the following feature: based on the public IP address of the visitor, identify their (rough) location (at the level of city or just country), and display it to them. And again, this should be in the web-page, not printed to the terminal. As before, both a screenshot of the webpage and your code is needed.

Note that for this, you will need to send an http request to a web-service that provides IP geolocation. There are many of them available. Some of them are paid, some only need a free registration, and some are completely free. Even the paid examples have a free tier, although it is likely to be "rate limited", that is, they restrict how many requests in an interval of time you can send them. Do not use any paid version. As a reminder, you are still expected to use socket-programming even for sending an http request and receiving its response (like a web-client does).