

LAB 5

AWS, SSH Clients, Client Server Programming

This lab assumes that you have already created an account on AWS, Amazon's cloud computing platform. Everything we cover in this lab and the next uses the free tier of AWS, i.e. you won't need any credit in your account to use these services.

1. Agenda of the lab

The goal of this lab is to learn how to:

- Set up and instantiate an EC2 instance under your AWS account
 - Paying attention to various security groups, making various selections of security groups and understanding the consequences of these selections.
- Communicating with the EC2 instance using SSH clients
 - Using PuTTY to install Python on your EC2 instance
 - Using FileZilla to transfer files – including, for example, python code – from the local machine to the server and vice versa
- TCP client and server
 - Writing and running a TCP server on the EC2 instance
 - Writing and running a TCP client on the local machine
 - Communicating between client and server

2. Lab Tasks

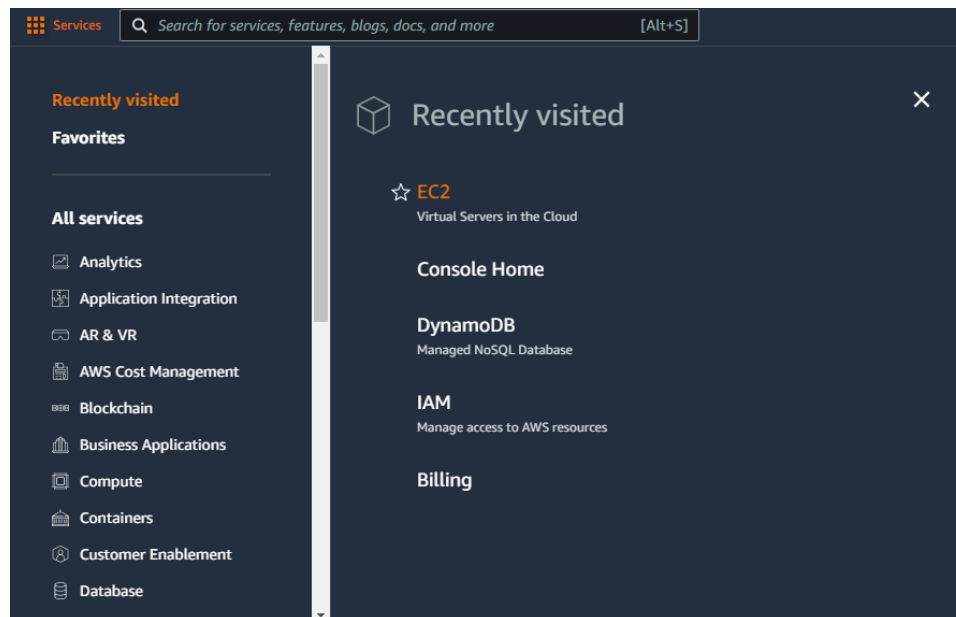
We will cover this agenda in a series of tasks described below.

2.1 Instantiating an EC2 instance

We begin by instantiating an EC2 instance.

Perform the following series of steps:

- i. **Sign-in to the AWS console as a *root* user.**
The option of IAM (Identity and Access Manager) user is for the purpose of creating an entity (person or application) that can interact with AWS services under specified rights and permissions. We will use this option later, when using a database service in AWS, so it is useful to note its existence.
- ii. **From the Services menu (top left of the page), select the EC2 option:**

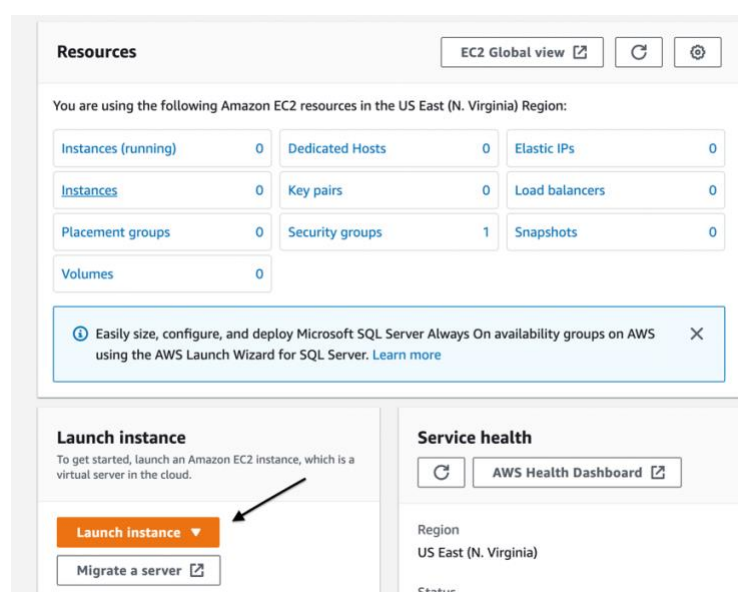


You can use the search bar on top to search for EC2 if it's not directly visible

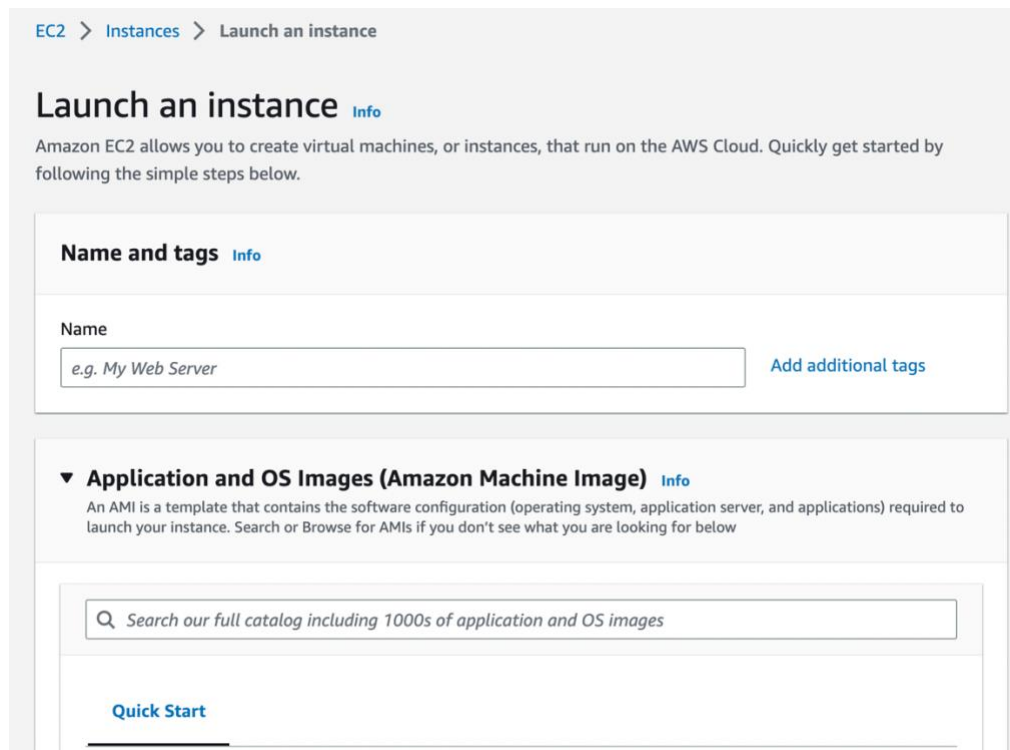
EC2 stands for Elastic Compute Cloud. It gives you access to a virtual computer on the cloud. This is a full-fledged machine with an operating system of your choice and considerable processing power. You can run many useful applications, including your own programs, on your EC2 instance. For example, one of the programs we will be running on our EC2 instance is a TCP server written in Python. *Essentially the same TCP server we wrote in the Software Systems module, but this time on a machine on the cloud.*

iii. Launch an EC2 instance.

Do this by clicking the Launch Instance button:



Next, we need to choose various specifications for the new instance. This is a multi-step process. The steps are divided into various sections on a scrollable page (partially shown below).



Specifications:

1. Name your instance and choose a Machine Image

After giving a suitable name to your instance, you need to specify the type of Processor, Storage and Operating System. As you'll see, some of the available options are eligible for the free tier. We'll be selecting one of these options, which should be sufficient for your projects.

In the search bar, type in Ubuntu , hit enter, and select the following option from the results (this is usually the second option from the top in the results):

	Ubuntu Server 20.04 LTS (HVM), SSD Volume Type	Select
Ubuntu	ami-0b93ce03dcbcb10f6 (64-bit (x86)) / ami-0f972b3903f44791c (64-bit (Arm))	<input checked="" type="radio"/> 64-bit (x86)
Free tier eligible	Ubuntu Server 20.04 LTS (HVM),EBS General Purpose (SSD) Volume Type. Support available from Canonical (http://www.ubuntu.com/cloud/services).	<input type="radio"/> 64-bit (Arm)
Verified provider	Platform: ubuntu Root device type: ebs Virtualization: hvm ENA enabled: Yes	

The difference between the first two options is among the Ubuntu versions. So, 22.04 runs a different version of the Linux Kernel. However, for our purposes it wouldn't make a real difference if you choose 22.04 LTS or 20.04 LTS.

We are choosing a 64 bit x86 processor. For operating system, we are choosing Ubuntu, which is a Linux distribution. We will be using this operating system via a command line interface allowing us to type and execute Linux commands remotely. Don't worry if you do not have prior experience with Linux. We will usually need very basic commands. I will provide a link below that can serve as a reference. You will find the system to be very intuitive and easy to understand.

If you scroll down the list of available images, you'll notice that outside the free tier you have options available for platforms suitable for various purpose. For example, some of them are good for machine learning applications and provide GPU support.

2. Choose Instance Type

Specify the number of processors, amount of memory and the networking capacity of your EC2 instance.

Staying within the free tier, we pick the default choice:

Instance type

t2.micro

Free tier eligible

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



[T2 instances](#) are designed to be suitable for applications that require occasional (burst) spikes of computation or full core utilizations. For our purpose, t2.micro is more than sufficient. Click on the [Compare Instance Types](#) link to see the details.

3. Create a key pair

This is a public/private key pair which is used by an encryption protocol, such as RSA. When connecting to our server through an SSH client (a client that lets us remotely log into the instance) we will need to use this key pair because SSH uses an encryption protocol when connecting to the server.

Click on *Create a new key pair*

Create key pair

×

Key pairs allow you to connect to your instance securely.

Enter the name of the key pair below. 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](#)

Key pair name

ec1-15-01-2022

The name can include upto 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 (Not supported for Windows instances)

Private key file format

☐ .pem

For use with OpenSSH

☒ .ppk

For use with PuTTY

Cancel

Create key pair

As shown above, we create a new key value pair, give it a useful name, and **Download Key Pair as a .ppk file**. (Not .pem because we'll be using PuTTY and it needs a .ppk file)

The .ppk file will download to your machine. Keep it safe for now. (this file is a one-time download).

4. Network Settings

This is an important part of the specifications as it directly effects the communication between our TCP client and the TCP server on the EC2 instance.

Click the Edit link to the right of Network Settings, to see the more detailed box below (shown partially here):

▼ Network settings [Info](#)

VPC - *required* [Info](#)

vpc-0ffff78eeebfa9780
172.31.0.0/16

(default) ▼



Subnet [Info](#)

No preference ▼



[Create new subnet](#)

Auto-assign public IP [Info](#)

Enable ▼

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

Security group name - *required*

launch-wizard-1

This security group will be added to all network interfaces. The name can't be edited after the security group is created. Max length is 255 characters. Valid characters: a-z, A-Z, 0-9, spaces, and ._-:/()#,@[]+=&;{}!\$*

Description - *required* [Info](#)

launch-wizard-1 created 2023-01-15T17:52:22.404Z

Inbound security groups rules

Under the inbound security groups rules, click on the Type drop box (whose default value is SSH), and select **All TCP**, as show below:

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

work interfaces. The name can't be edited after the security group is created. Max length is 255 characters, and ._-:/()#,@[]+=&;{}!\$*

launch-wizard-1 created 2023-01-15T17:52:22.404Z

0.0.0.0/0) [Remove](#)

Protocol [Info](#) **Port range [Info](#)**

Custom TCP
Custom UDP
Custom ICMP - IPv4
Custom ICMP - IPv6
Custom Protocol
All TCP
All UDP
All ICMP - IPv4
All ICMP - IPv6
All traffic
ssh ✓
ssh ▲

Source type [Info](#) **Source [Info](#)** **Description - optional [Info](#)**

Anywhere ▼

0.0.0.0/0 ✕

This makes the instance less secure. However, it allows our simple TCP client to communicate with the server without adding a layer of security.

5. Configure Storage

Choose the default values for this option.

6. Advanced details / IAM instance profile

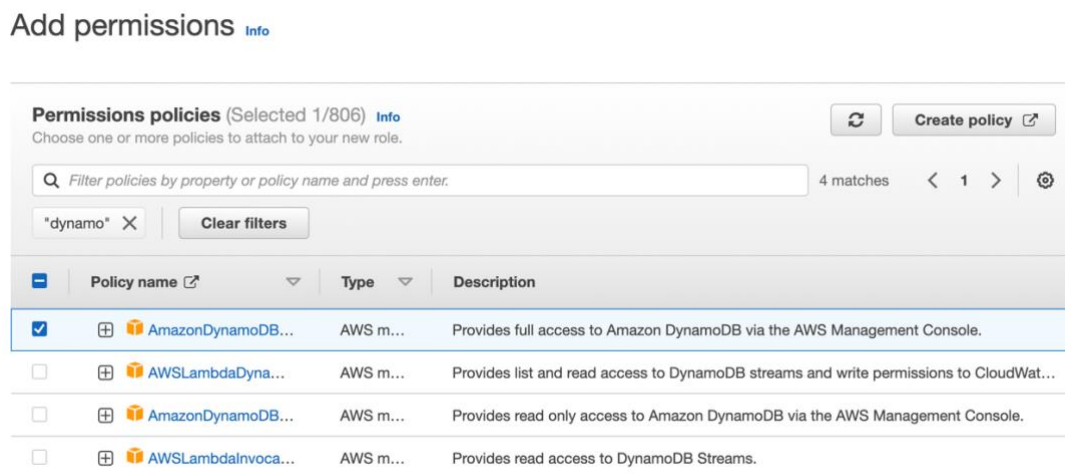
We need to create an IAM role for the next lab, in which we will be using DynamoDB, Amazon's distributed Non-relational database.

We will need to create an IAM profile first. So click on, Create new IAM profile.

From the screen that appears, click Create Role.

In the Use Cases, select EC2.

On the Add permissions screen, search for DynamoDB and make the following selection from the options that appear:



Essentially, this allows your EC2 instance to communicate with DynmoDB using the role which you just created. IAM roles are something that AWS uses to structure the communication between its difference services.

On the next screen, give this role a suitable name, such as, DynamoDBAccessEC2.

Click Create Role without making any more changes.

Your newly created role will now appear in the list of available roles, as show below.

Roles (3) [Info](#)

Delete
Create role

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

< 1 >

<input type="checkbox"/>	Role name	Trusted entities	Last activity
<input type="checkbox"/>	AWSServiceRoleForSupport	AWS Service: support (Service-Linked Role)	-
<input type="checkbox"/>	AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor (Service-Linked Role)	-
<input type="checkbox"/>	DynamoDBAccessEC2	AWS Service: ec2	-

7. View Summary and Launch Instance

Finally, back on the Launch an instance page, review the summary, which should look something like the following, and click Launch instance.

▼ Summary

Number of instances [Info](#)

Software Image (AMI)
Ubuntu Server 20.04 LTS (HVM),...[read more](#)
ami-0b93ce03dcbcb10f6

Virtual server type (instance type)
t2.micro

Firewall (security group)
New security group

Storage (volumes)
1 volume(s) - 8 GiB

i
Free tier: In your first year includes 750 hours of t2.micro (or t3.micro in the Regions in which t2.micro is unavailable)

×

Cancel

Launch instance

After the instance has been launched you can view it on your instances page:

Instances (1) Info								
<input type="text" value="Find instance by attribute or tag (case-sensitive)"/> < 1 > 								
<input type="checkbox"/>	Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability Zone	Pl
<input type="checkbox"/>	myinstance1	i-0a0e5de2fe1cf38ec	Running	t2.micro	-	No alarms	us-east-1a	ec

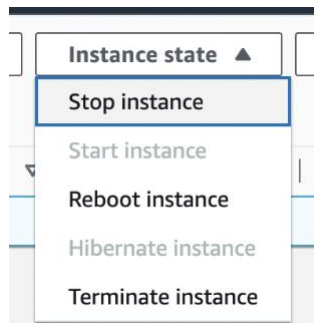
Clicking the check box next to the instance's name, you can see many useful details about it, as shown below:

Instance: i-0a0e5de2fe1cf38ec (myinstance1)		
▼ Instance summary Info		
Instance ID i-0a0e5de2fe1cf38ec (myinstance1)	Public IPv4 address 54.208.110.223 open address	Private IPv4 addresses 172.31.86.168
IPv6 address -	Instance state Running	Public IPv4 DNS ec2-54-208-110-223.compute-1.amazonaws.com open address
Hostname type IP name: ip-172-31-86-168.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-86-168.ec2.internal	Elastic IP addresses -
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	AWS Compute Optimizer finding
Auto-assigned IP address	VPC ID	

You may recall the difference between a public and private IP address from our discussion of NAT in Software Systems. For our local machine to connect to our EC2 instance, it will need to use its public IPv4 address – in this case, 54.208.110.223.

IMPORTANT note:

Please remember to stop an instance after you have done your work. When you start work next time, you can restart this instance. Remember that each time you restart the instance, it gets assigned a new public IPv4 address.

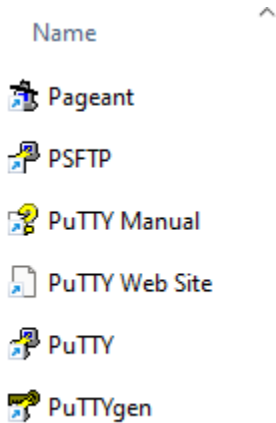


2.2 Communicating with the EC2 instance

Now we'll use **PuTTY**, an open-source terminal emulator operating over SSH, to communicate with our newly launched EC2 instance.

PuTTY is needed on the windows systems. MAC users can work directly from the terminal. A separate (short) document has been added for MAC users.(You may still use FileZilla as described below).

For windows users, download and install [PuTTY](#). Following components will be installed on your system:

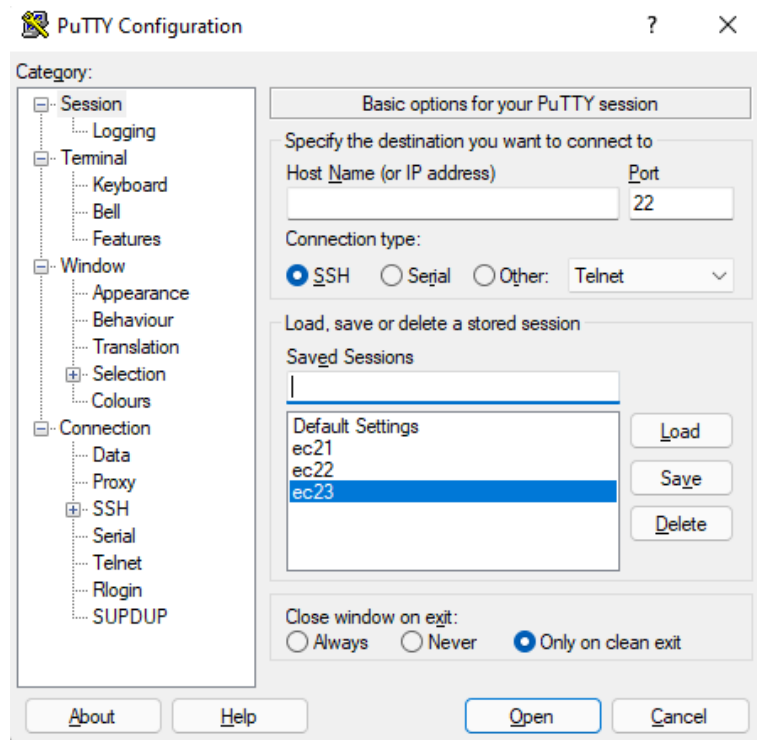


The **PuTTY** application (second from the bottom) is the terminal emulator that lets us run Linux commands on our remote Ubuntu based EC2 instance.

sThe **PSFTP** application is a secure SSH based file transfer tool. It is a command line tool which you can use to transfer files between your local computer and the server. However, we'll use another tool called **FileZilla** for this purpose. **FileZilla** provides a GUI for file transfer, making the process much simpler and speedier. (FileZilla is available for both MAC and Windows).

2.2.1 Connect PuTTY to the EC2 instance

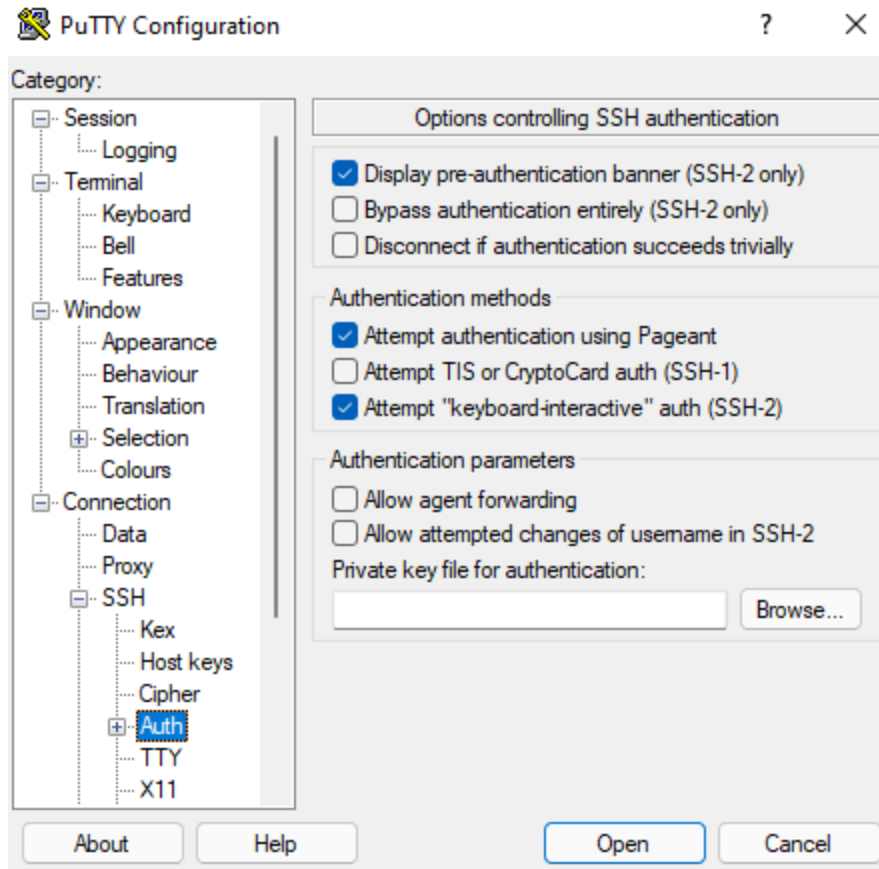
- Run the PuTTY app.



In the picture above, you see that I already have some sessions saved in PuTTY from previous use. You will save your first session shortly.

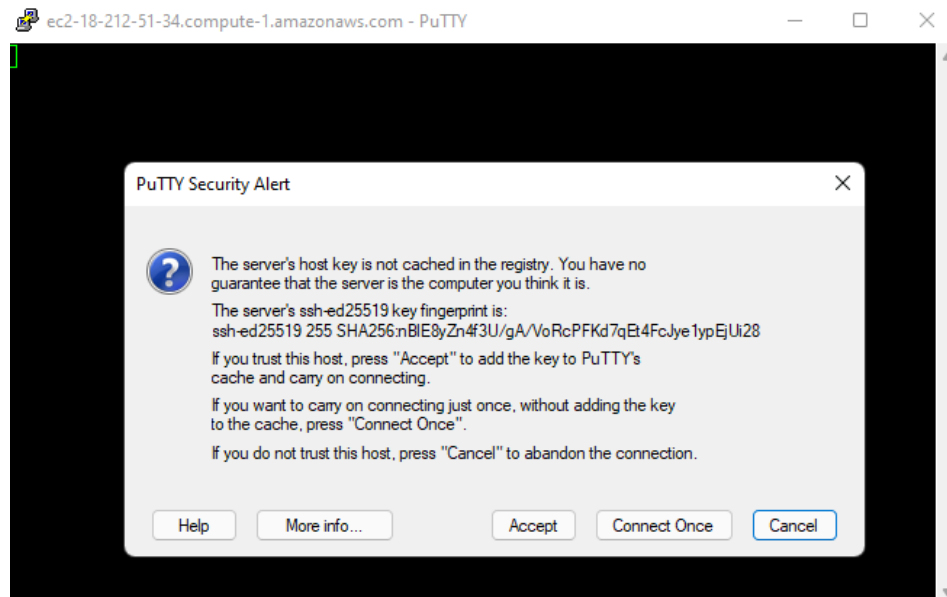
Notice the Port 22. This is port number used by the SSH protocol (the port number it inserts into the TCP socket).

- In the “Host Name (or IP address)” edit box, enter the public IP address of your EC2 instance
- From the Category pane on the left of the window, under the Connection node, expand SSH and select Auth



This is where you supply the private key .ppk file created during the EC2 instantiation.

- Click Browse and select the .ppk file.
- At this point, if we Click Open, PuTTY will be connected to your EC2 instance. However, it is good idea to save our current connection settings (i.e. the current session) so that we can reconnect later using the same settings.
 - Return to the Session window by selecting Session from the Category pane.
 - Enter a session name (such as ec21, ec22 and ec23 in one of the pictures above)
 - Click save.
- Click Open. You will see the following security alert. However, our server is trusted, so click Accept.

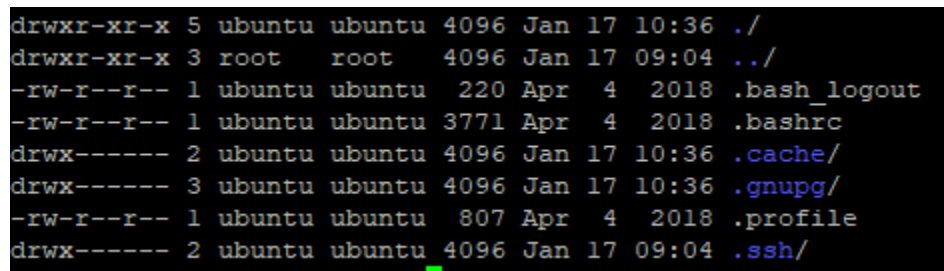


- In **login as:** enter *ubuntu* (this is your default username on the EC2 instance).
- Hit enter and you should be in your EC2 instance.

```
ubuntu@ip-172-31-89-33: ~  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:        https://ubuntu.com/advantage  
  
System information as of Mon Jan 17 11:13:30 UTC 2022  
  
System load:  0.0           Processes:            93  
Usage of /:   15.2% of 7.69GB Users logged in:         0  
Memory usage: 18%          IP address for eth0: 172.31.89.33  
Swap usage:   0%  
  
0 updates can be applied immediately.  
  
New release '20.04.3 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Mon Jan 17 10:36:13 2022 from 39.45.163.133  
To run a command as administrator (user "root"), use "sudo <command>".  
See "man sudo_root" for details.  
  
ubuntu@ip-172-31-89-33:~$
```

You can now execute Linux commands on the remote machine. Here is a [link](#) of some basic Linux commands. You'll find plenty of help on this on the Web.

- For example, enter **ls -aIF** to view all the files and directories in the current directory, including hidden files.



```
drwxr-xr-x 5 ubuntu ubuntu 4096 Jan 17 10:36 ./
drwxr-xr-x 3 root root 4096 Jan 17 09:04 ../
-rw-r--r-- 1 ubuntu ubuntu 220 Apr 4 2018 .bash_logout
-rw-r--r-- 1 ubuntu ubuntu 3771 Apr 4 2018 .bashrc
drwx----- 2 ubuntu ubuntu 4096 Jan 17 10:36 .cache/
drwx----- 3 ubuntu ubuntu 4096 Jan 17 10:36 .gnupg/
-rw-r--r-- 1 ubuntu ubuntu 807 Apr 4 2018 .profile
drwx----- 2 ubuntu ubuntu 4096 Jan 17 09:04 .ssh/
```

There is plenty of information here. If you further type **cd ..** you will go one level up into the home directory. Then type **ls -aIF** again and you'll see the directory you were previously in, i.e., **ubuntu**. Doing a **cd ubuntu** will take you back inside the **ubuntu** directory. The columns of information in the graphic above show file permissions, number of links, owner name, owner group, file size in bytes, time of last modification, and file name.

2.2.2 Install Python on the EC2 instance

This is a good time to install Python on our EC2 instance.

Simply entering the following on the command line should do the job:

```
$ sudo apt install python3.7
```

(when prompted, enter 'Y' and proceed)

Once python is installed, you can check it by running:

```
$ python3
```

```
//this will take you inside Python terminal
```

```
>>> 3+5
```

```
///hitting enter will show you the result
```

```
>>> exit()
```

```
///hitting enter will exit from Python and take you back to Linux command line.
```

sudo is command that allows you to perform administrative tasks on a Linux system.

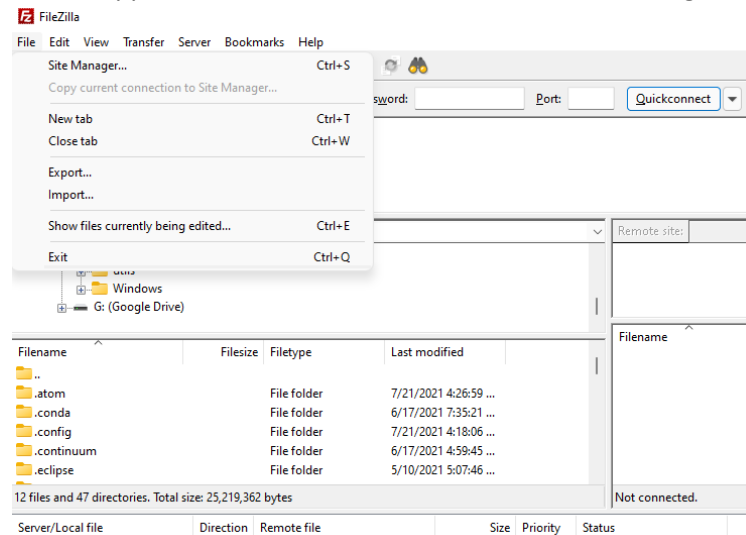
apt (Advanced Package Tool) is used to install various packages on the system. You can use it to install anything you need on the EC2 instance. Python here has been used as an example.

2.2.3 Move files to the server using FileZilla

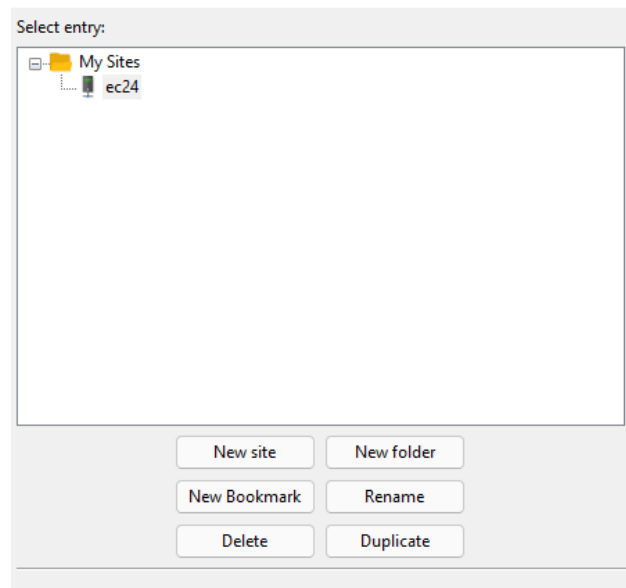
With Python installed, we are now able to execute python code on our EC2 instance. We can write this code directly on the server using a Linux based editor like nano. However, it is much more

convenient to first write the code on our local machine, then move it to EC2 and run it there. We need a way to transfer files from our local machine to EC2. We use WinSCP to accomplish this.

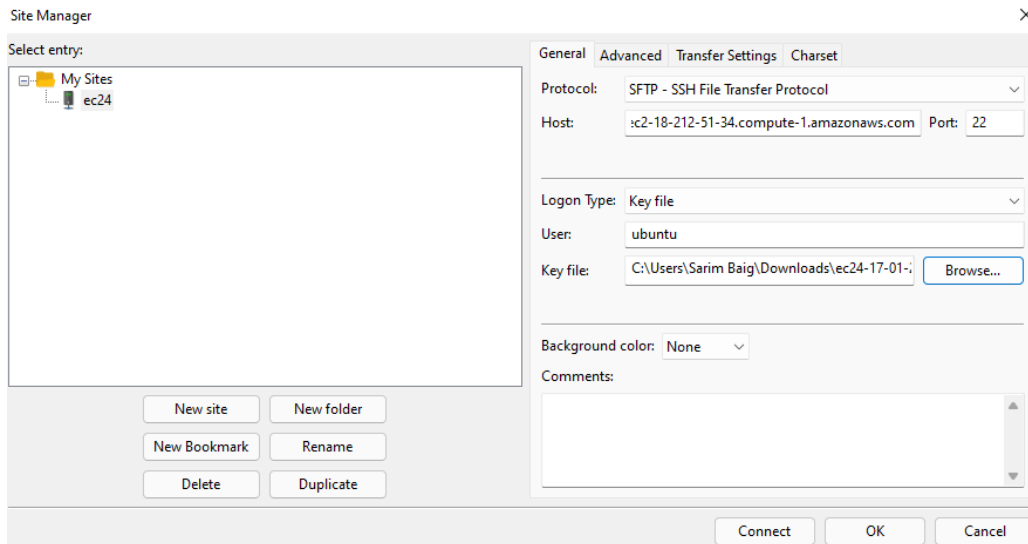
- First, download and install [FileZilla](#).
- Run the FileZilla application. From the File menu select Site Manager.



- In Site Manager, click 'New site' and give a suitable name to the new entry. In the picture below I've given it the name ec24

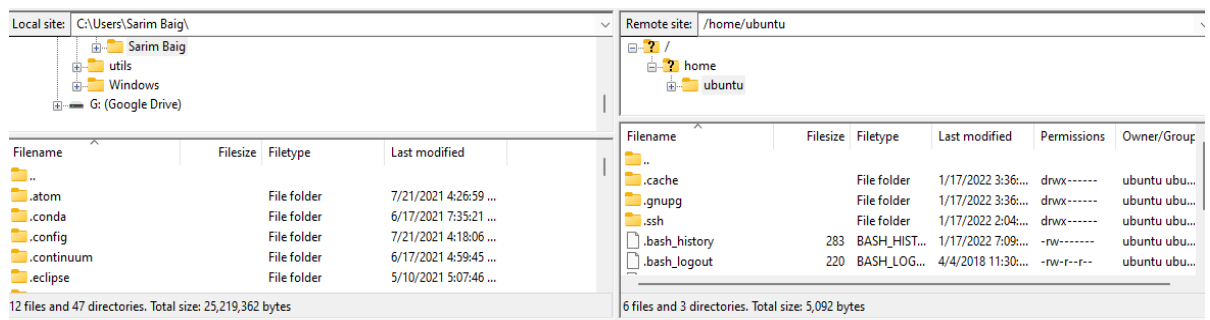


- Under the General tab, add information as shown below:



- In Protocol, choose SFTP – SSH File Transfer Protocol
- In Host, enter the Public IPv4 DNS of your EC2 instance.
- In Port write 22 (the SSH port)
- In the Logon Type, select Key file
- In User, enter ubuntu.
- For key file, Browse to the .ppk file created using PuTTYgen.
- Click Connect.

You will now see the local site (local computer) and remote site (EC2 instance) directory structure side by side. You can drag and drop files between them.



Move the file **simpleAdd.py** from your local machine to the EC2 instance. (This file has been provided with this guide.)

- Now let's return to PuTTY and list the contents in the current folder. You should see simpleAdd.py.


```
ubuntu@ip-172-31-89-33: ~  
* Support: https://ubuntu.com/advantage  
  
System information as of Mon Jan 17 15:26:52 UTC 2022  
  
System load: 0.0 Processes: 96  
Usage of /: 15.4% of 7.69GB Users logged in: 0  
Memory usage: 20% IP address for eth0: 172.31.89.33  
Swap usage: 0%  
  
* Ubuntu Pro delivers the most comprehensive open source security and  
compliance features.  
  
https://ubuntu.com/aws/pro  
  
0 updates can be applied immediately.  
  
New release '20.04.3 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Mon Jan 17 11:13:31 2022 from 39.45.163.133  
ubuntu@ip-172-31-89-33:~$ ls  
simpleAdd.py  
ubuntu@ip-172-31-89-33:~$
```

- To execute the program, enter:
python3 simpleAdd.py

In the next section, we will move our familiar **tcpserver.py** file (from Software Systems) to our EC2 instance and run in there.

2.3 Running a TCP Server on the EC2 instance

We first set up the simple TCP server and client. Later, we run our server as a service on the EC2 instance so that even after our SSH session with the EC2 instance terminates the TCP server keeps running.

2.3.1 The Server and the Client

You might recall the following code from your Software Systems class. This is a TCP server. You are familiar with the basic concepts of TCP sockets used here.

```
import socket  
print("We're in tcp server...");  
  
#select a server port  
server_port = 12000  
#create a welcoming socket  
welcome_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

```
#bind the server to the localhost at port server_port
welcome_socket.bind(('0.0.0.0',server_port))

welcome_socket.listen(1)

#ready message
print('Server running on port ', server_port)

#Now the main server loop
while True:
    connection_socket, caddr = welcome_socket.accept()
    #notice recv and send instead of recvto and sendto
    cmsg = connection_socket.recv(1024)
    cmsg = cmsg.decode()
    if(cmsg.isalnum() == False):
        cmsg = "Not alphanumeric.";
    else:
        cmsg = "Alphanumeric";
    connection_socket.send(cmsg.encode())
```

Notice the IP address 0.0.0.0 used in the bind() function. When we wrote the same code on our local computer, we used the address 127.0.0.1 (i.e. 'localhost'). This is the IP address assigned to the "loopback" or local-only interface. This is a 'fake' network adapter that can only communicate within the same host. In comparison, when a server is told to listen on 0.0.0.0 that means 'listen on every available network interface', which is exactly what we want our remote server to do.

There is nothing special about the port number 12000. Any number outside of the reserved space may be used.

- Copy this code to a python file, say *tcpserver.py*
- Move *tcpserver.py* to your EC2 instance using FileZilla
- Execute *tcpserver.py* on the EC2 instance.

```
ubuntu@ip-172-31-89-33: ~  
  
System load: 0.0          Processes:          96  
Usage of /: 15.4% of 7.69GB Users logged in:    1  
Memory usage: 20%        IP address for eth0: 172.31.89.33  
Swap usage: 0%  
  
* Ubuntu Pro delivers the most comprehensive open source security and  
  compliance features.  
  
  https://ubuntu.com/aws/pro  
  
0 updates can be applied immediately.  
  
New release '20.04.3 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.  
  
Last login: Mon Jan 17 15:26:53 2022 from 39.45.171.242  
ubuntu@ip-172-31-89-33:~$ ls  
simpleAdd.py  tcpserver.py  
ubuntu@ip-172-31-89-33:~$ python3 tcpserver.py  
We're in tcp server...  
Server running on port 12000
```

The server is now running and ready to listen to a TCP client. We will run the client on our local machine. Following is the familiar python code for the TCP client.

```
import socket  
print("We're in tcp client...");  
  
#the server name and port client wishes to access  
server_name = '18.212.51.34'  
  
# '52.205.252.164'  
server_port = 12000  
#create a TCP client socket  
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)  
  
#Set up a TCP connection with the server  
#connection_socket will be assigned to this client on the server side  
client_socket.connect((server_name, server_port))  
  
msg = input("Enter a string to test if it is alphanumeric: ");  
  
#send the message to the TCP server  
client_socket.send(msg.encode())  
  
#return values from the server  
msg = client_socket.recv(1024)
```

```
print(msg.decode())
client_socket.close()
```

Once again, the thing to note is the IP address used in the connect function (stored in the `server_name` variable). This is the current Public IPv4 address of my EC2 instance. You should replace this address with the current Public IPv4 address of your instance.

Run this program on your computer. It should be able to communicate with the server.

2.3.2 Running the TCP Sever as a service

Currently, our TCP server will terminate as soon as our SSH session with the EC2 instance terminates. However, we will like to keep our server running in order to keep our application going. To keep our server running after the SSH session terminates (and to keep working on the terminal with the server running in the background) we need to launch our `tcpserver.py` code as a service (or a daemon, in Linux terminology).

This can be done in multiple ways. I describe just one, using `systemd`, a software component within Linux used to configure services.

Broadly, we will need to take the following steps:

Step 1: Configuring `tcpserver.py` as a service by creating a service configuration file

All services are located in `/etc/systemd/system`

Create a file, called `tcpserver.service` in this location. You can do this using the nano editor as follows:

```
~$ sudo nano /etc/systemd/system/tcpserver.service
```

The role of `sudo` is important here. We need super user rights to create the service file.

Add the following content to `tcpserver.service`

```
[Unit]
Description=TCP server service
After=multi-user.target

[Service]
Type=simple
ExecStart=/usr/bin/python3 /home/ubuntu/tcpserver.py

[Install]
```

WantedBy=multi-user.target

This content is quite self-explanatory. The multi-user.target specifications indicate when after the system startup this service will be executed. ExecStart flag takes in the command that you want to run. The first argument is the python path and the second argument is the path to the script that needs to be executed.

Step 2: Launch the service

To launch the service, enter the following on the command line:

```
~$ sudo systemctl daemon-reload
```

```
~$ sudo systemctl enable tcpserver.service
```

```
~$ sudo systemctl start tcpserver.service
```

In these three lines, we have: reloaded the services information, enabled our service and started our service respectively. The TCP server is now running in the background as a service.

In order to confirm that it's there, enter the following on the command line:

```
~$ ps -ef
```

ps displays the currently running processes. You should see the tcpserver.py in the list.

Step 3: Communicate between client and server (now a service)

Simply run the tcpclient on your local machine. It should communicate with the server just as before.

Step 4 (How to) kill the server

At some stage you might want to kill the current server process before your launch an updated version. In order to do that, once again enter:

```
~$ ps -ef
```

Notice that the enter for tcpserver.py has a process id. This is the number in the second column in the tcpserver.py row. On my system at the moment this number is 25294. In order to kill the process, I will enter the following:

```
~$ sudo kill -9 25294
```

If you have multiply python processes running as services, you might want to save your time by doing

~\$ sudo pkill python

This will terminate all python processes.

2.4 Optional Exercises (beyond the lab)

- (a) Compute the average RTT between your local computer (tcpclient.py) and the EC2 instance (tcpserver.py) while transmitting one integer of data. RTT is the time it takes for the integer to travel from the client to the server and back. Average this time over 500 communications and print the running average.
- (b) tcpclient.py can accept data from our IoT devices and forward it to tcpserver.py, which can perform computations on it, and a so on. Add code to your tcpclient.py to accept accelerometer data from one of the FPGA boards. Send this data in string format to the tcpserver.py on EC2. The server should append each data unit with the prefix "Received:" and return it to the client. The client should print the output line by line.
