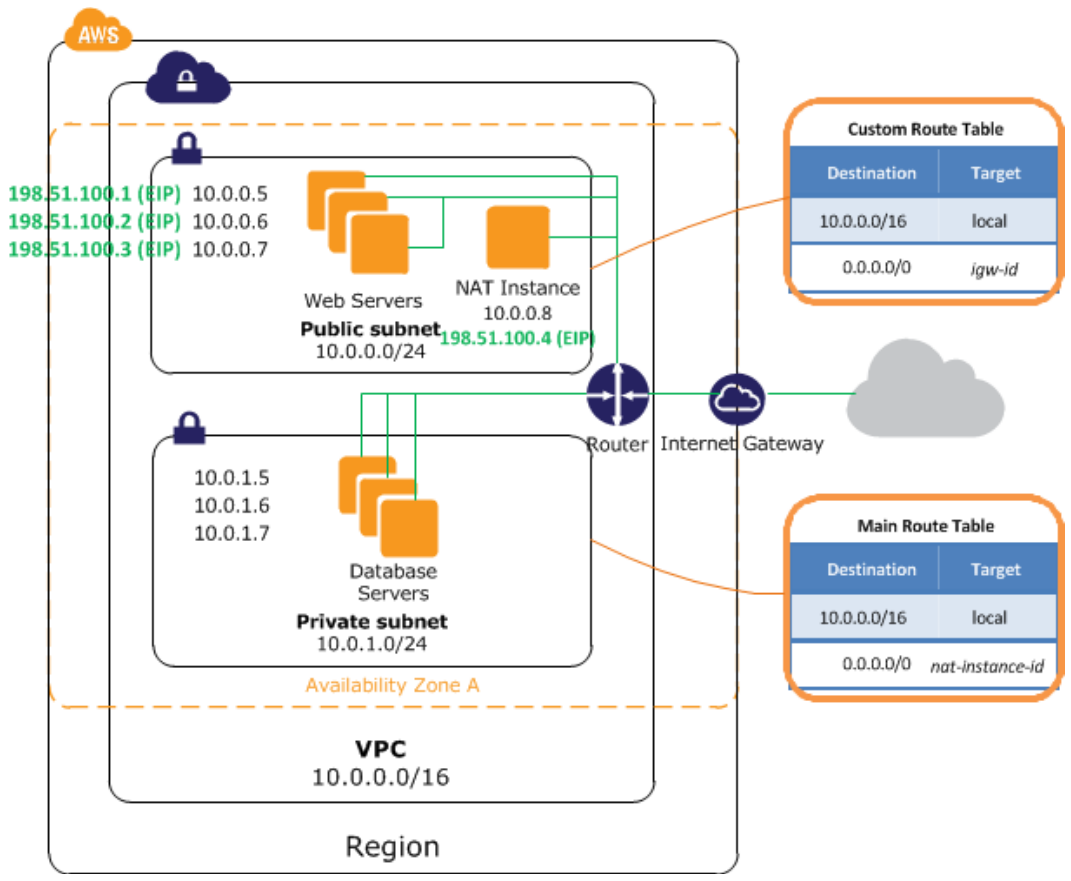
CS 594—Spring 2021
Enterprise Security and Information Assurance
Assignment One—Virtual Private Cloud

In this assignment, you will set up the IT infrastructure that you will be using for deploying your applications later in the semester. You will set up this infrastructure in the Amazon Virtual Private Cloud (VPC), which provides an infrastructure-as-a-service (IaaS) for configuring a custom virtual network and virtual machines. The particular scenario you will work with, using a virtual cloud network with a public subnet for the application server and a private subnet for the database server, is described by the following diagram:



Custom Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*igw-id*

Main Route Table

Destination	Target
10.0.0.0/16	local
0.0.0.0/0	*nat-instance-id*

You can find more information about Amazon VPC here:

http://aws.amazon.com/documentation/vpc/

The following document contains further information about the specific VPC setup for this assignment:

http://docs.aws.amazon.com/AmazonVPC/latest/UserGuide/VPC_Scenario2.html

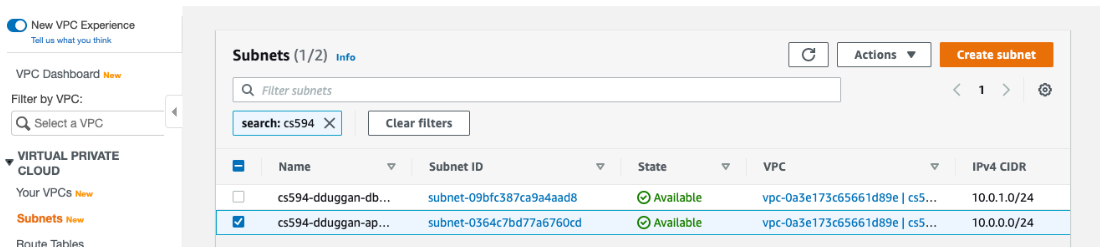
In sum, the steps are as follows:

1. Create a virtual private cloud with two subnets, a private subnet and a public subnet. See below.
2. Create security groups to control external access to your instances: your NAT instance (for Internet access), public subnet app server, and private subnet database server. This sets up firewall policies to protect you against attackers attempting to compromise your deployment. Also create a network ACL to add an additional layer of protection for your private subnet.
3. Launch an EC2 instance into your public subnet, to host your application server. Launch the instance from a bare Amazon Linux 64-bit AMI, without any additional software installed.
4. Launch another EC2 instance into your private subnet, to host your database server.
5. Allocate a volume in EBS for storing a database. Attach this volume to the database server instance, and mount it as an external file system on your database instance.
6. Install a Docker-managed database management system (Postgresql) on the database server instance, using the external file system to store its data.
7. Install a Docker-managed application server (Payara) on the app server instance.
8. Configure a connection pool for your database in the application server, authenticating with the credentials of the simple database user, and a JNDI resource that applications will use to access the database.
9. Deploy a simple "Hello, World" application that you will be provided with.
10. Save your customized app server and database server instances as AMIs, so you can use them for subsequent assignments. Do not leave your instances running when you have finished recording videos. Do not leave the VPC NAT running (stop it, do not terminate it!).

Setting Up The Private Cloud

Create a virtual private cloud with two subnets, a private subnet and a public subnet. The VPC will use a /16 CIDR address block, while each subnet will use a /24 CIDR address block. Use the following naming schemes for your VPC and subnets:

- VPC: `cs594-`*`yourstevensuserid`*
- Public subnet: `cs594-`*`yourstevensuserid`*`-app-server`
- Private subnet: `cs594-`*`yourstevensuserid`*`-db-server`

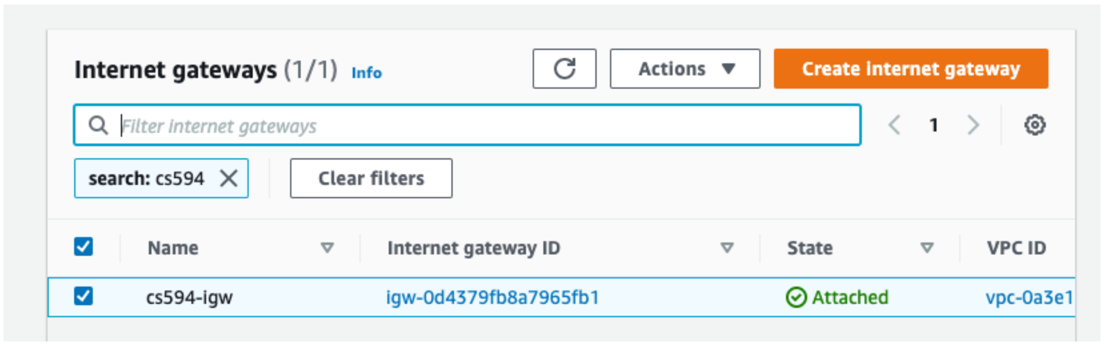


Subnets (1/2) Info

	Name	Subnet ID	State	VPC	IPv4 CIDR
☐	cs594-dduggan-db…	subnet-09bfc387ca9a4aad8	Available	vpc-0a3e173c65661d89e | cs5…	10.0.1.0/24
☑	cs594-dduggan-ap…	subnet-0364c7bd77a6760cd	Available	vpc-0a3e173c65661d89e | cs5…	10.0.0.0/24

You will need different the routing tables for these subnets, a main routing table, that just allows routing within the virtual cloud, and a custom routing table.

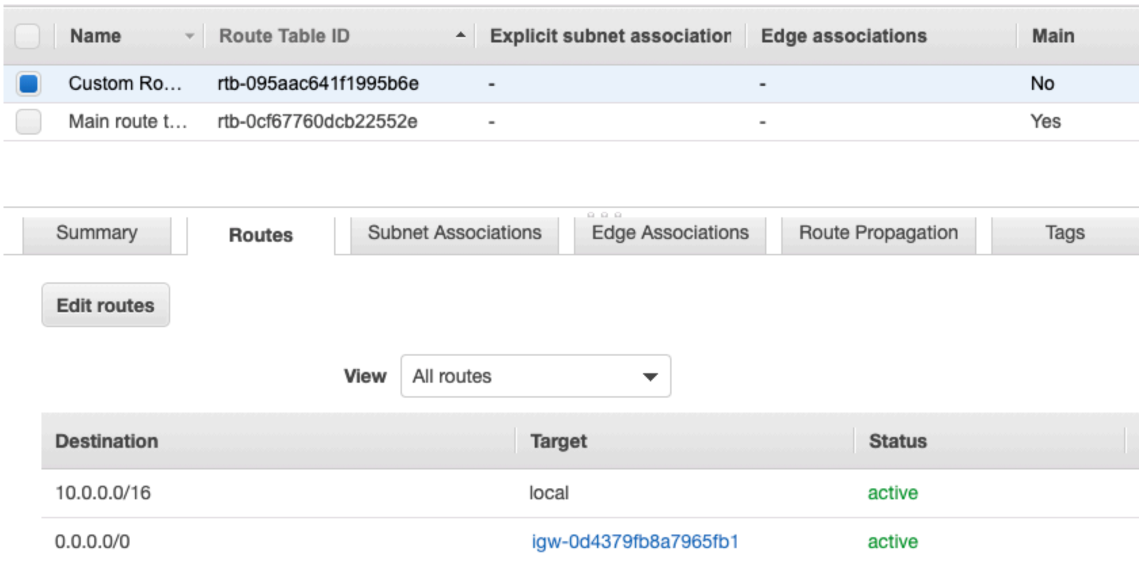
You will need to create a custom routing table, explicitly associated with the public subnet, allowing routing of traffic from the public subnet to the Internet through an Internet gateway (0.0.0.0/0). First, create an Internet gateway associated with this VPC:



Internet gateways (1/1) Info

	Name	Internet gateway ID	State	VPC ID
☑	cs594-igw	igw-0d4379fb8a7965fb1	Attached	vpc-0a3e1

Now create a custom routing table that allows all hosts in the VPC to communicate with each other, and allows these hosts to connect to the Internet via this Internet gateway:

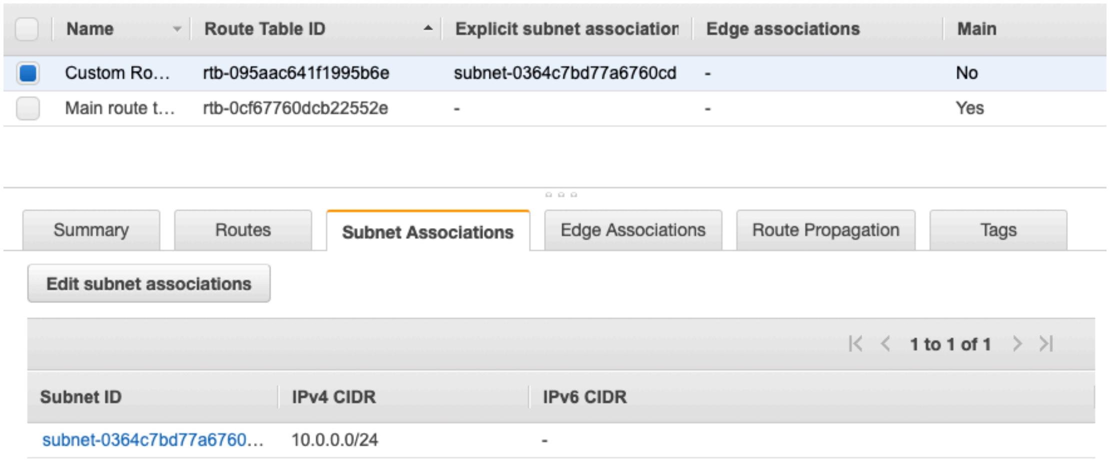


	Name	Route Table ID	Explicit subnet association	Edge associations	Main
■	Custom Ro…	rtb-095aac641f1995b6e	-	-	No
☐	Main route t…	rtb-0cf67760dcb22552e	-	-	Yes

View: All routes

Destination	Target	Status
10.0.0.0/16	local	active
0.0.0.0/0	igw-0d4379fb8a7965fb1	active

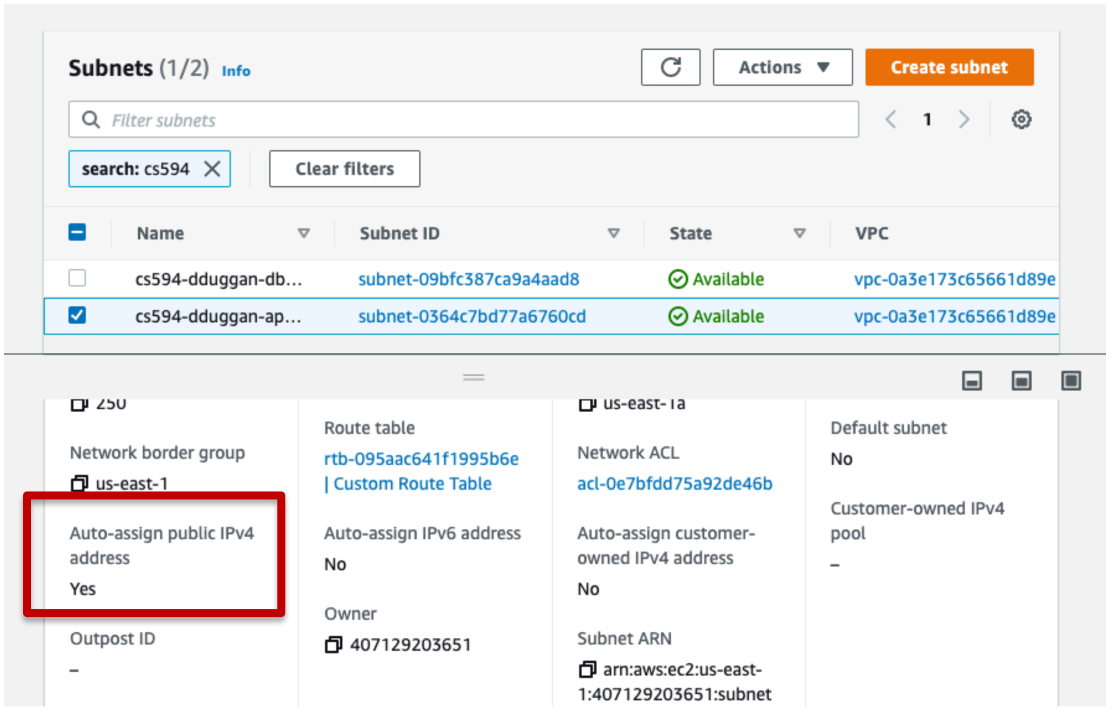
Then associate this routing table with the public subnet:



	Name	Route Table ID	Explicit subnet association	Edge associations	Main
✓	Custom Ro…	rtb-095aac641f1995b6e	subnet-0364c7bd77a6760cd	-	No
	Main route t…	rtb-0cf67760dcb22552e	-	-	Yes

Subnet ID	IPv4 CIDR	IPv6 CIDR
subnet-0364c7bd77a6760…	10.0.0.0/24	-

You should also explicitly allow auto-assignment of public IPv4 addresses to instance in your public subnet, otherwise your app server will have no public IP address and you will not be able to connect to via ssh:



	Name	Subnet ID	State	VPC
	cs594-dduggan-db…	subnet-09bfc387ca9a4aad8	Available	vpc-0a3e173c65661d89e
✓	cs594-dduggan-ap…	subnet-0364c7bd77a6760cd	Available	vpc-0a3e173c65661d89e

250

Network border group
us-east-1

Auto-assign public IPv4 address
Yes

Outpost ID
–

Route table
rtb-095aac641f1995b6e | Custom Route Table

Auto-assign IPv6 address
No

Owner
407129203651

us-east-1a

Network ACL
acl-0e7bfdd75a92de46b

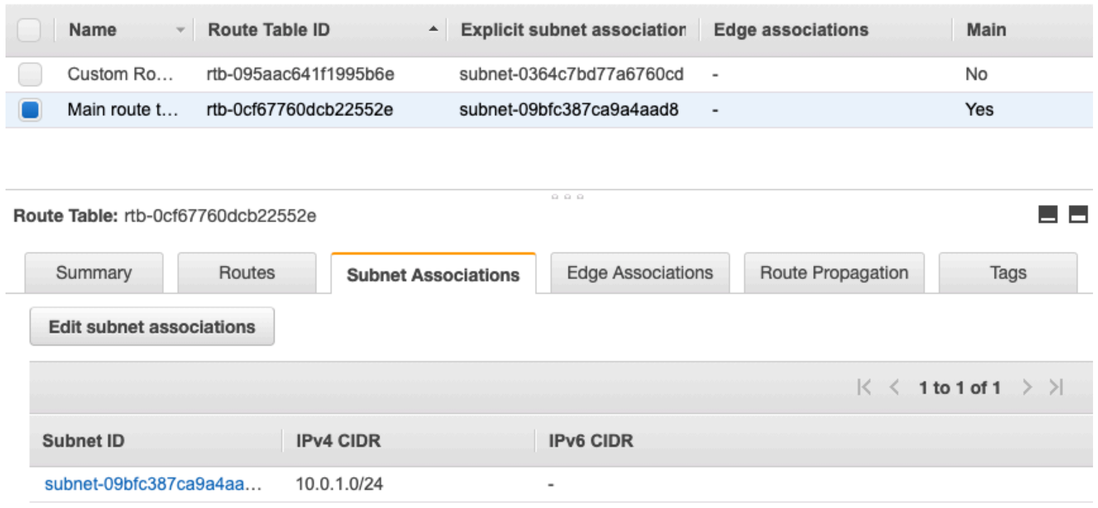
Auto-assign customer-owned IPv4 address
No

Subnet ARN
arn:aws:ec2:us-east-1:407129203651:subnet

Default subnet
No

Customer-owned IPv4 pool
–

The main routing table should be associated with the private subnet:



	Name	Route Table ID	Explicit subnet association	Edge associations	Main
☐	Custom Ro…	rtb-095aac641f1995b6e	subnet-0364c7bd77a6760cd	-	No
■	Main route t…	rtb-0cf67760dcb22552e	subnet-09bfc387ca9a4aad8	-	Yes

Route Table: rtb-0cf67760dcb22552e

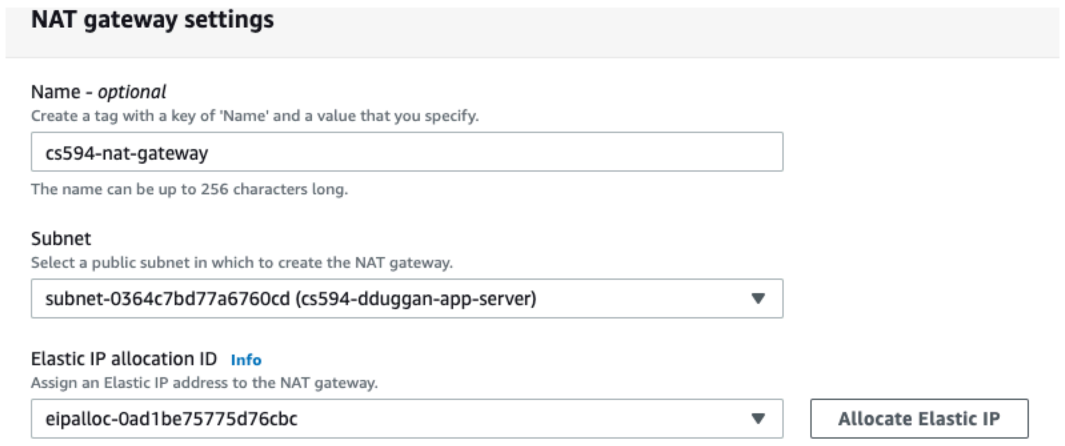
Summary | Routes | **Subnet Associations** | Edge Associations | Route Propagation | Tags

Edit subnet associations

1 to 1 of 1

Subnet ID	IPv4 CIDR	IPv6 CIDR
subnet-09bfc387ca9a4aa…	10.0.1.0/24	-

This allows an instance in that subnet to communicate with any other instance in the VPC, while communications outside the VPC should be directed to the Internet through a NAT instance. You will need to create a NAT gateway, for which you need to allocate an elastic IP address:



NAT gateway settings

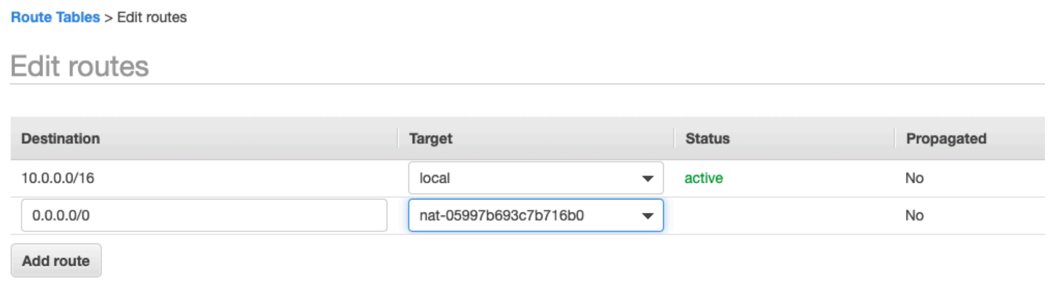
Name - *optional*
Create a tag with a key of 'Name' and a value that you specify.
cs594-nat-gateway
The name can be up to 256 characters long.

Subnet
Select a public subnet in which to create the NAT gateway.
subnet-0364c7bd77a6760cd (cs594-dduggan-app-server)

Elastic IP allocation ID Info
Assign an Elastic IP address to the NAT gateway.
eipalloc-0ad1be75775d76cbc

Allocate Elastic IP

Then add a rule in the main routing table to allow access to this NAT gateway, this will allow e.g. a database server on the private subnet to pull down software updates:



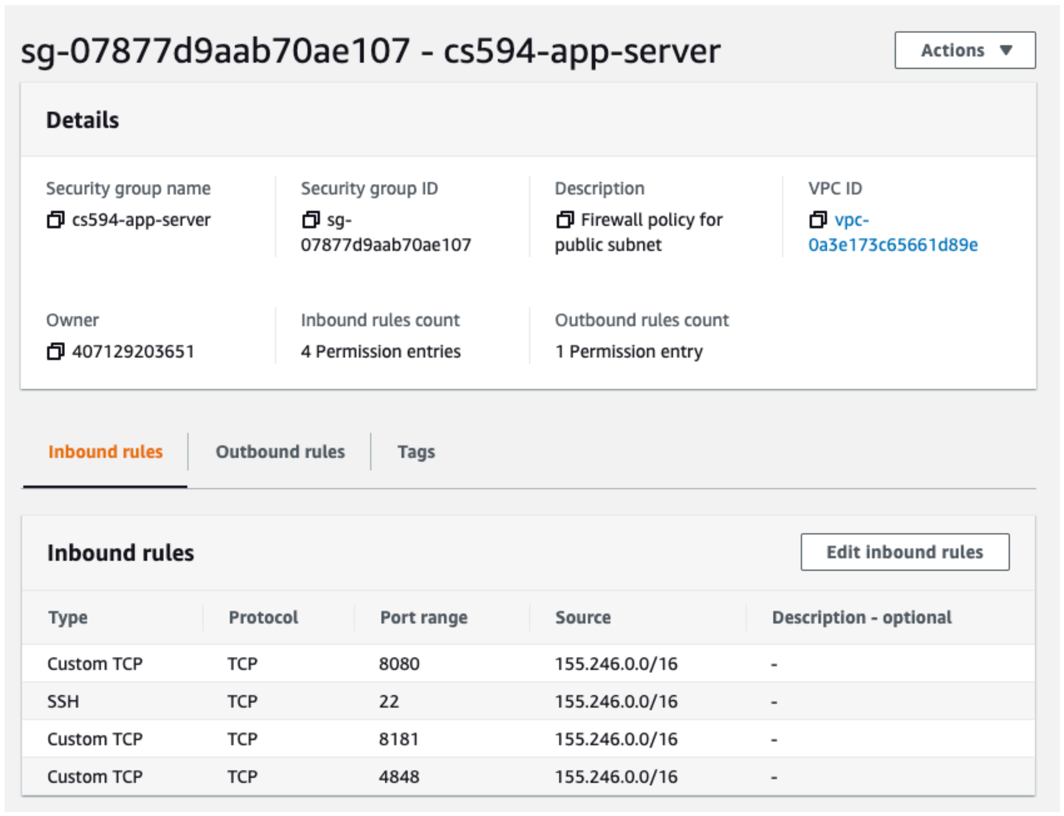
Route Tables > Edit routes

Edit routes

Destination	Target	Status	Propagated
10.0.0.0/16	local	active	No
0.0.0.0/0	nat-05997b693c7b716b0		No

Add route

You will want to create security groups to control external access to your instances. A security group sets up a firewall policy to protect you against attackers attempting to compromise your deployment. *These security groups are associated with a particular VPC, so make sure you specify the correct VPC.* For the public subnet, you will want to allow yourself ssh access (port 22) to the virtual machine on which the app server runs, as well as https access (port 4848) to the administration console, but you will want to limit access to your own IP address. In general you would presumably allow remote access from anywhere to apps in the app server (say at port 8080 and 8181, for http and https, respectively), but for the purposes of this class it is okay to again limit access to the Stevens subnet (155.246.0.0/16). You will need to allow outbound requests from the public subnet to ssh (port 22) and postgresql (port 5432) in the private subnet, although you would want to restrict access to port 22 on the private subnet in a production deployment.



sg-07877d9aab70ae107 - cs594-app-server

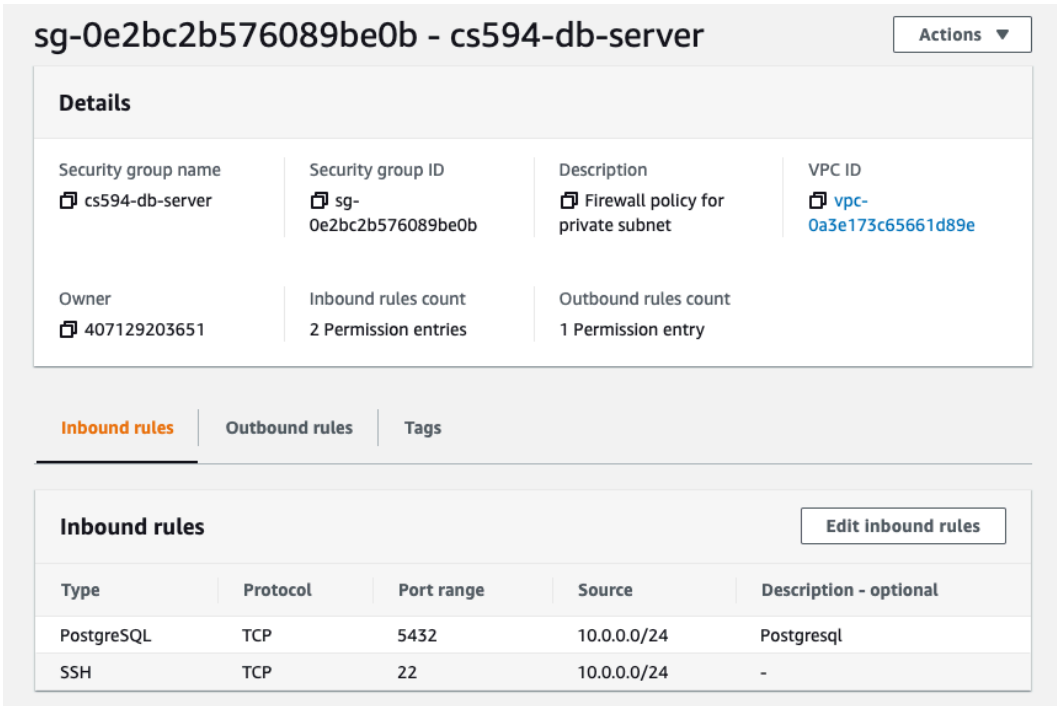
Details

Security group name	Security group ID	Description	VPC ID
cs594-app-server	sg-07877d9aab70ae107	Firewall policy for public subnet	vpc-0a3e173c65661d89e
Owner	Inbound rules count	Outbound rules count	
407129203651	4 Permission entries	1 Permission entry	

Inbound rules

Type	Protocol	Port range	Source	Description - optional
Custom TCP	TCP	8080	155.246.0.0/16	-
SSH	TCP	22	155.246.0.0/16	-
Custom TCP	TCP	8181	155.246.0.0/16	-
Custom TCP	TCP	4848	155.246.0.0/16	-

For the database server, you will want to allow inbound access to ssh (port 22) and postgresql (port 5432), but only from the app server subnet.



sg-0e2bc2b576089be0b - cs594-db-server

Details

Security group name	Security group ID	Description	VPC ID
cs594-db-server	sg-0e2bc2b576089be0b	Firewall policy for private subnet	vpc-0a3e173c65661d89e

Owner	Inbound rules count	Outbound rules count
407129203651	2 Permission entries	1 Permission entry

Inbound rules

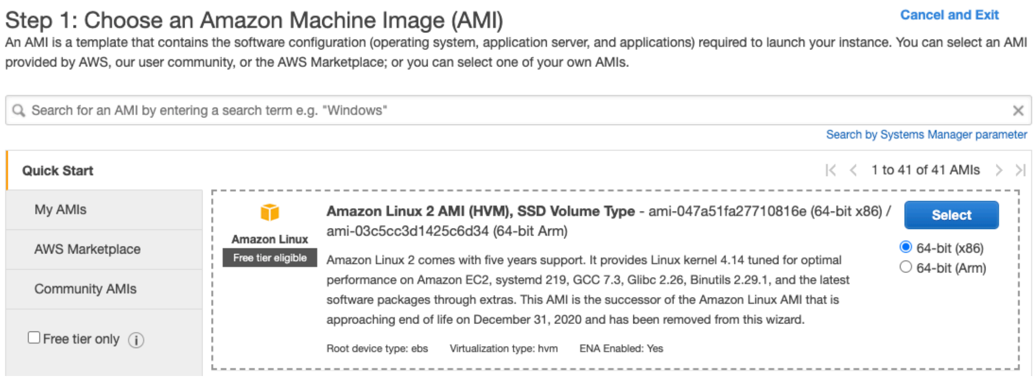
Type	Protocol	Port range	Source	Description - optional
PostgreSQL	TCP	5432	10.0.0.0/24	Postgresql
SSH	TCP	22	10.0.0.0/24	-

As for outbound rules, these may be useful for supporting status reports and code update pull requests, though in general you would want to restrict the target of these outbound requests.

Setting Up The Database Server

Step 1: Launch The Database Instance

Launch an EBS-backed EC2 instance into the private subnet. Launch the instance from a bare **Amazon Linux 2** 64-bit AMI, without any additional software installed. Elastic Block Store (EBS) is essentially a virtual disk that Amazon provides for backing storage for EC2 instances.



Step 1: Choose an Amazon Machine Image (AMI)

An AMI is a template that contains the software configuration (operating system, application server, and applications) required to launch your instance. You can select an AMI provided by AWS, our user community, or the AWS Marketplace; or you can select one of your own AMIs.

Search for an AMI by entering a search term e.g. "Windows"

1 to 41 of 41 AMIs

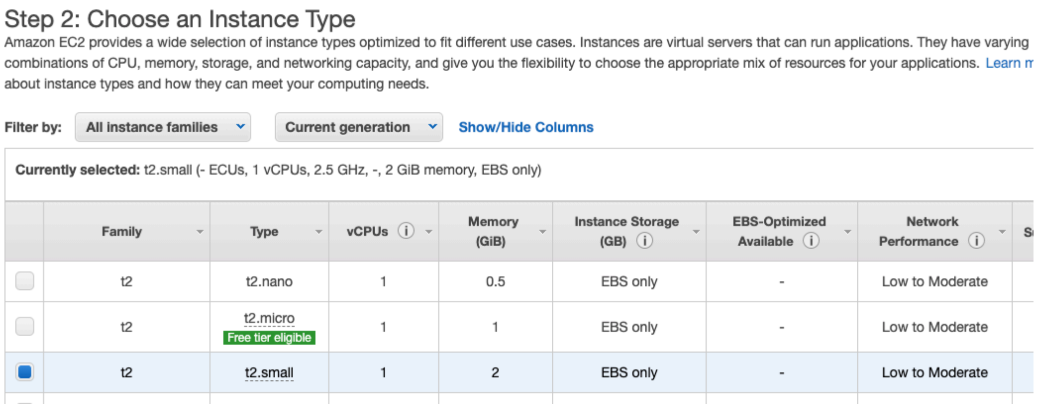
Amazon Linux 2 AMI (HVM), SSD Volume Type - ami-047a51fa27710816e (64-bit x86) / ami-03c5cc3d1425c6d34 (64-bit Arm)

Amazon Linux 2 comes with five years support. It provides Linux kernel 4.14 tuned for optimal performance on Amazon EC2, systemd 219, GCC 7.3, Glibc 2.26, Binutils 2.29.1, and the latest software packages through extras. This AMI is the successor of the Amazon Linux AMI that is approaching end of life on December 31, 2020 and has been removed from this wizard.

Root device type: ebs Virtualization type: hvm ENA Enabled: Yes

[✓ 64-bit (x86)] 64-bit (Arm)

Pick a small-sized instance. Provided you do not leave an instance running, your charges should be minimal:



Step 2: Choose an Instance Type

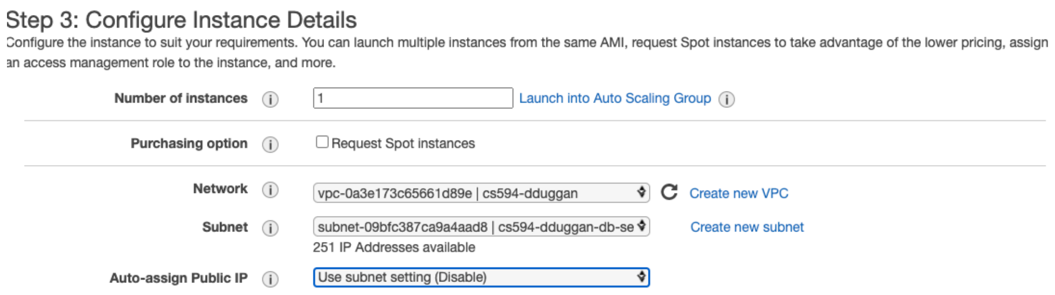
Amazon EC2 provides a wide selection of instance types optimized to fit different use cases. Instances are virtual servers that can run applications. They have varying combinations of CPU, memory, storage, and networking capacity, and give you the flexibility to choose the appropriate mix of resources for your applications. Learn m about instance types and how they can meet your computing needs.

Filter by: All instance families | Current generation | Show/Hide Columns

Currently selected: t2.small (- ECUs, 1 vCPUs, 2.5 GHz, -, 2 GiB memory, EBS only)

	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance	S
☐	t2	t2.nano	1	0.5	EBS only	-	Low to Moderate	
☐	t2	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate	
☑	t2	t2.small	1	2	EBS only	-	Low to Moderate	

Make sure that you choose the private subnet on the VPC that you are setting up:

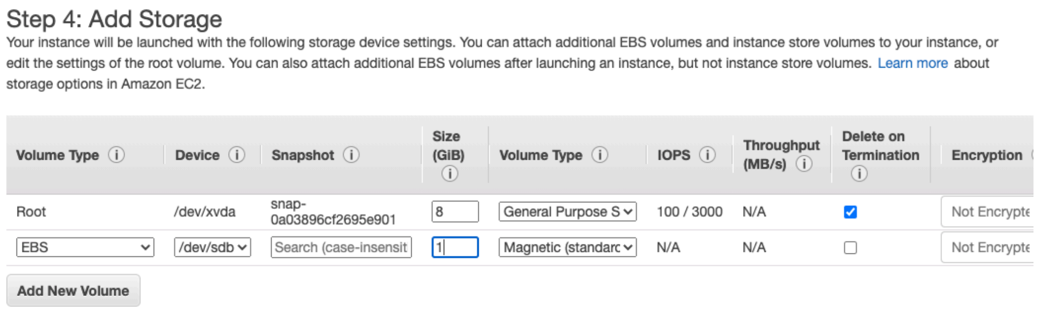


Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1 — Launch into Auto Scaling Group
Purchasing option	☐ Request Spot instances
Network	vpc-0a3e173c65661d89e | cs594-dduggan — Create new VPC
Subnet	subnet-09bfc387ca9a4aad8 | cs594-dduggan-db-se — Create new subnet 251 IP Addresses available
Auto-assign Public IP	Use subnet setting (Disable)

Allocate an additional EBS volume to hold the persistent state of the database:



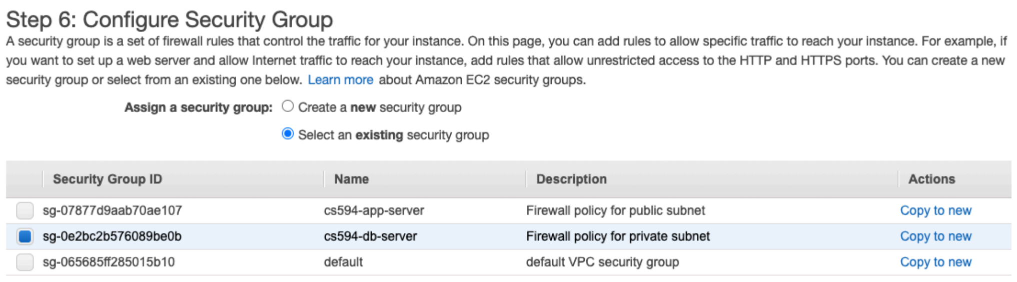
Step 4: Add Storage

Your instance will be launched with the following storage device settings. You can attach additional EBS volumes and instance store volumes to your instance, or edit the settings of the root volume. You can also attach additional EBS volumes after launching an instance, but not instance store volumes. Learn more about storage options in Amazon EC2.

Volume Type	Device	Snapshot	Size (GiB)	Volume Type	IOPS	Throughput (MB/s)	Delete on Termination	Encryption
Root	/dev/xvda	snap-0a03896cf2695e901	8	General Purpose S	100 / 3000	N/A	☑	Not Encrypte
EBS	/dev/sdb	Search (case-insensit	1	Magnetic (standarc	N/A	N/A	☐	Not Encrypte

Add New Volume

Use a firewall policy (security group) for your instance that you defined above for the private subnet.



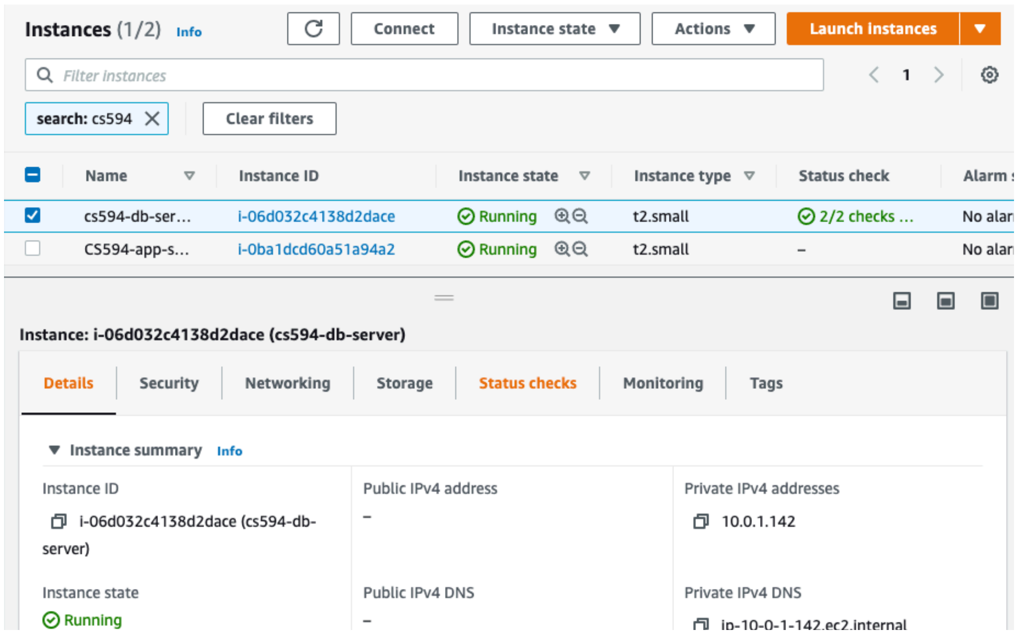
Step 6: Configure Security Group

A security group is a set of firewall rules that control the traffic for your instance. On this page, you can add rules to allow specific traffic to reach your instance. For example, if you want to set up a web server and allow Internet traffic to reach your instance, add rules that allow unrestricted access to the HTTP and HTTPS ports. You can create a new security group or select from an existing one below. Learn more about Amazon EC2 security groups.

Assign a security group: Create a **new** security group [✓ Select an **existing** security group]

	Security Group ID	Name	Description	Actions
	sg-07877d9aab70ae107	cs594-app-server	Firewall policy for public subnet	Copy to new
✓	sg-0e2bc2b576089be0b	cs594-db-server	Firewall policy for private subnet	Copy to new
	sg-065685ff285015b10	default	default VPC security group	Copy to new

You will be asked to define a private/public key pair, to be used to authenticate to the running instance. Be sure to protect this key pair, never share it with anyone. At this point, you can see the instance running in your EC2 dashboard. Note the public DNS address for your running instance, you will use that, with the key pair, to ssh to the instance to set it up:



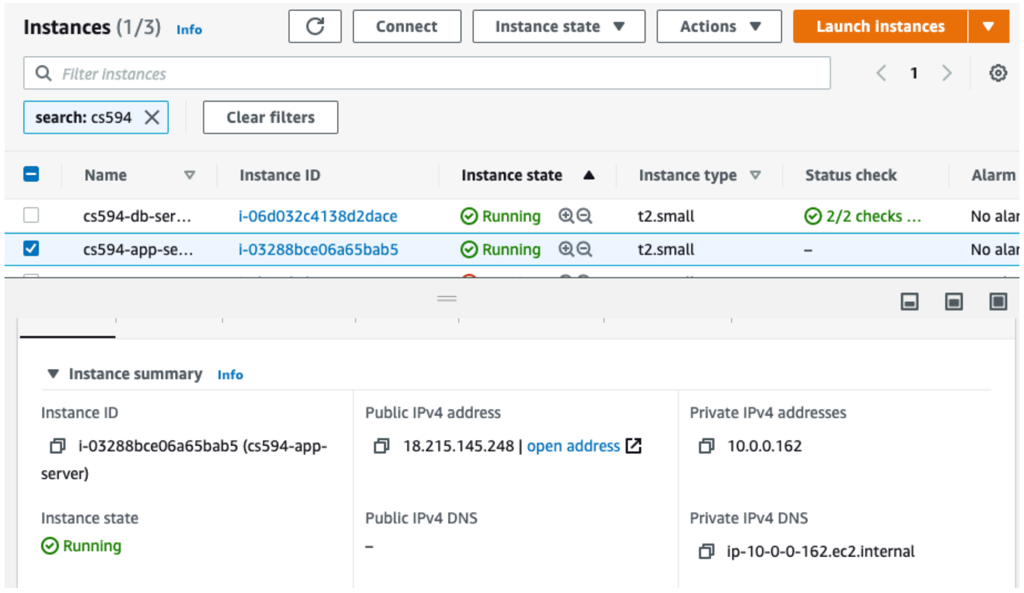
	Name	Instance ID	Instance state	Instance type	Status check	Alarm
✓	cs594-db-ser…	i-06d032c4138d2dace	Running	t2.small	2/2 checks …	No alar
	CS594-app-s…	i-0ba1dcd60a51a94a2	Running	t2.small	–	No alar

Instance: i-06d032c4138d2dace (cs594-db-server)

Instance ID	Public IPv4 address	Private IPv4 addresses
i-06d032c4138d2dace (cs594-db-server)	–	10.0.1.142
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	–	ip-10-0-1-142.ec2.internal

You can also view the volumes that have been allocated, one for the software you will install, the other for the persistent database content.

In a similar fashion, you can launch an instance for the application server, but this time into the public subnet (which will assign it a public IP address) and with the security policy for the application server:



Name	Instance ID	Instance state	Instance type	Status check	Alarm
cs594-db-ser...	i-06d032c4138d2dace	Running	t2.small	2/2 checks ...	No alar
cs594-app-se...	i-03288bce06a65bab5	Running	t2.small	–	No alar

Instance summary Info

Instance ID	Public IPv4 address	Private IPv4 addresses
i-03288bce06a65bab5 (cs594-app-server)	18.215.145.248 | open address	10.0.0.162
Instance state	**Public IPv4 DNS**	**Private IPv4 DNS**
Running	–	ip-10-0-0-162.ec2.internal

Step 2: Mount the disk for the database

Use ssh to access your instance[1], and install any software updates:

```
ssh -i cs594-keypair.pem -l ec2-user <ip-address-for-database-instance>

$ sudo yum update
```

Edit the mount table `/etc/fstab` to automatically mount the external volume as a Linux file system whenever the machine boots. In what follows, `[ec2-user]` denotes the default user prompt, while `[root]` denotes the superuser prompt, while ****** denotes the UUID you obtain from `file -s`:

```
[ec2-user] sudo su -
[root] fdisk -l
[root] mkfs -t ext3 /dev/xvdb
[root] file -s /dev/xvdb
[root] echo "UUID=***** /data ext3 noatime 0 0" >> /etc/fstab
[root] mkdir /data
[root] mount /data
[root] fdisk -l
[root] exit
```

Step 3: Install Docker on your instance

[1] You will have to do this from your public instance, by ssh-ing to that instance public DNS address, because there is no public DNS or IP address for the database instance on the private subnet.

Install Docker, which is available as a package, and start the Docker service. You will be using the command-line Docker client to manage Docker containers and images:

```
[ec2-user] sudo yum install -y docker

[ec2-user] sudo service docker start

[ec2-user] sudo usermod -a -G docker ec2-user
```

The last command makes it unnecessary to use sudo to execute the Docker client. You will need to log out and log back in again, and you may need to reboot the instance (do not terminate it).

Step 4: Create the database container

Rather than install Postgresql natively using yum, you will instead install a docker image that includes a Postgresql installation, and run this as a container. Since the app server will run on a separate machine, it will be useful to add "`-p 5432:5432`" as an option when running the database container, to expose the port[2]:

```
[ec2-user] docker pull postgres

[ec2-user] docker run -d --name cs594db -p 5432:5432 -v
/data:/var/lib/postgresql/data -e POSTGRES_PASSWORD=XXXXXX -e
PGDATA=/var/lib/postgresql/data/pgdata postgres

[ec2-user] docker ps
```

The `docker run` command will create and start the database container in the background (due to the `–d` flag). You have mounted the external disk at `/data`, and this is now mounted at `/var/lib/postgresql/data` in the container. The `PGDATA` environment variable sets the container directory where the database will be stored. The `POSTGRES_PASSWORD` environment variable defines the password for the database superuser (default superuser is `postgres`).

Note: If you do not see the running container, you can see all docker containers including those that have stopped, and view the logs for a particular docker container by specifying its container id:

```
docker ps -a

docker logs <container-id>
```

[2] The `--hostname` option here specifies the "virtual host name" for the running docker instance. This is useful if you are setting up postgresql and payara on your local machine, using Docker Desktop, for development purposes (which you should).

You can stop a container using `docker stop`, remove a container using `docker rm`, and remove all stopped docker containers using `docker container prune`.
With the database container still running, run a bash shell container from the same image in the same virtual network, and use a Postgresql command line tool in the shell to create a user for the database you will be creating. This command will connect to the running database. In this case, the `-it` flag runs the command interactively, while the `--rm` flag removes the shell container when you are done[3]:

```
[ec2-user] docker exec -it <container-id> /bin/bash
# createuser cs594user -P --createdb -U postgres
Enter password for new role: YYYYYY
Enter it again: YYYYYY
# exit
```

Note that the `createuser` command connects to the database server running in the docker container, as superuser `postgres`. Run a `psql` shell to create the database for this user:

```
[ec2-user] docker exec -it <container-id> psql -U postgres
postgres=# create database cs594 with owner cs594user;
postgres=# \q
```

Setting Up The Application Server

Step 5: Launch The Server Instance

Launch an instance as before, this time into the public subnet of your VPC. Connect to this instance using ssh and install any software updates, as before.

Step 6: Create the server container

Create a custom app server image that includes the latest JDBC driver. First download the driver (go to `http://jdbc.postgresql.org` to find the link for the latest version):

```
[ec2-user] mkdir cs594-payara
[ec2-user] cd cs594-payara
[ec2-user] wget <link to the driver>
```

Create a file called `Dockerfile` that copies the JDBC driver into the server libraries[4]:

```
FROM payara/server-full
```

[3] If you are running this on your laptop, replace the IP address with the virtual host name `cs594db`.
[4] There are two lines, the second line is broken to fit in this document.

```
COPY <JDBC driver file name>
               /opt/payara/appserver/glassfish/domains/domain1/lib/
```

Save this file, and create a custom app server image with the JDBC driver:

```
[ec2-user] docker build -t cs594/server .
[ec2-user] docker images
[ec2-user] cd ..
```

Create and start the server container. Note that you specify the same image name as above when you executed `docker build`; this image will have been cached locally. You will need to expose several ports to allow access from your Web browser, through your EC2 firewall:

```
[ec2-user] docker run -d --name payara -p 4848:4848 -p 8080:8080 -p
8181:8181 cs594/server
```

Use a Web browser to verify that the application server is running by going to the admin console at port 4848. The default user name and password are "`admin`", you should change the password in the admin console.

You need to create a JDBC connection pool so applications deployed in the application server can connect to the database. In the admin console, navigate to `Resources | JDBC | Connection Pools`. Create a connection pool with name `cs594Pool`, with type `javax.sql.ConnectionPoolDataSource`, and with vendor `PostgreSQL`. Set these data source properties[5]:

- `DatabaseName=cs594`
- `Password=`*`YYYYYY`* (see above)
- `ServerName=`*`db-ip-address`*
- `User=cs594user`
- `URL=jdbc:postgresql://`*`database-server-ip`*`:5432/cs594`

When you have saved these changes, select the new connection pool and click `Ping`. This will make sure that the application server is able to connect to the database server.

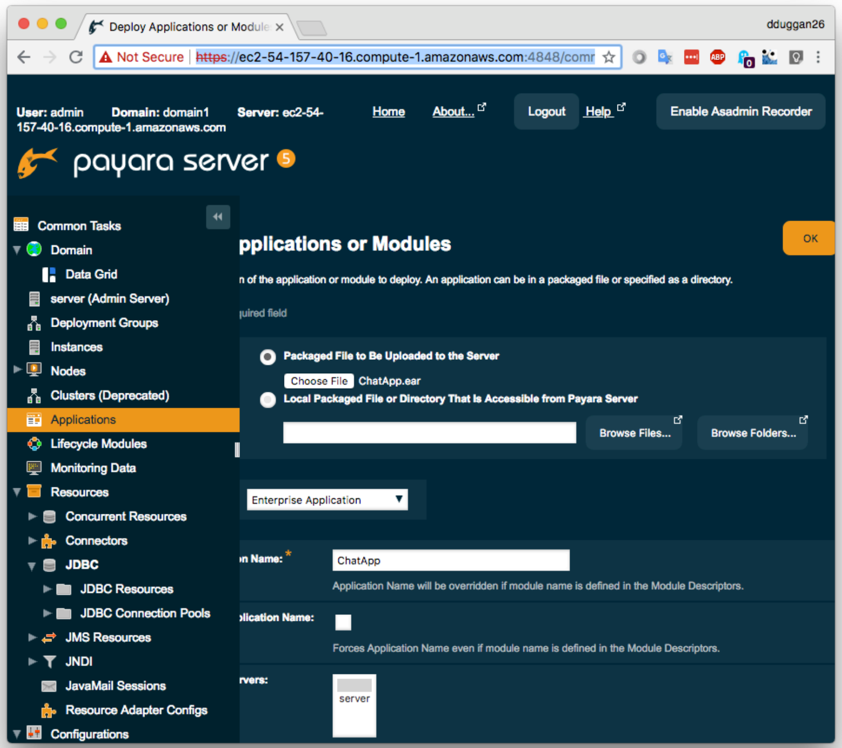
You still need to set up a JDBC resource for the connection pool, which your deployed applications can inject. In the admin console, navigate to `Resources | JDBC | JDBC Resources`. Create a new JDBC resource with these properties:

- JNDI Name: `jdbc/cs594`
- Connection pool: `cs594Pool`

Step 7: Deploy an application

[5] For a local setup on your laptop, use `cs594db` as *`database-server-ip`*.

In the admin console, select the Applications tab:



User: admin Domain: domain1 Server: ec2-54-157-40-16.compute-1.amazonaws.com

pplications or Modules

n of the application or module to deploy. An application can be in a packaged file or specified as a directory.

juired field

● Packaged File to Be Uploaded to the Server
ChatApp.ear
○ Local Packaged File or Directory That Is Accessible from Payara Server

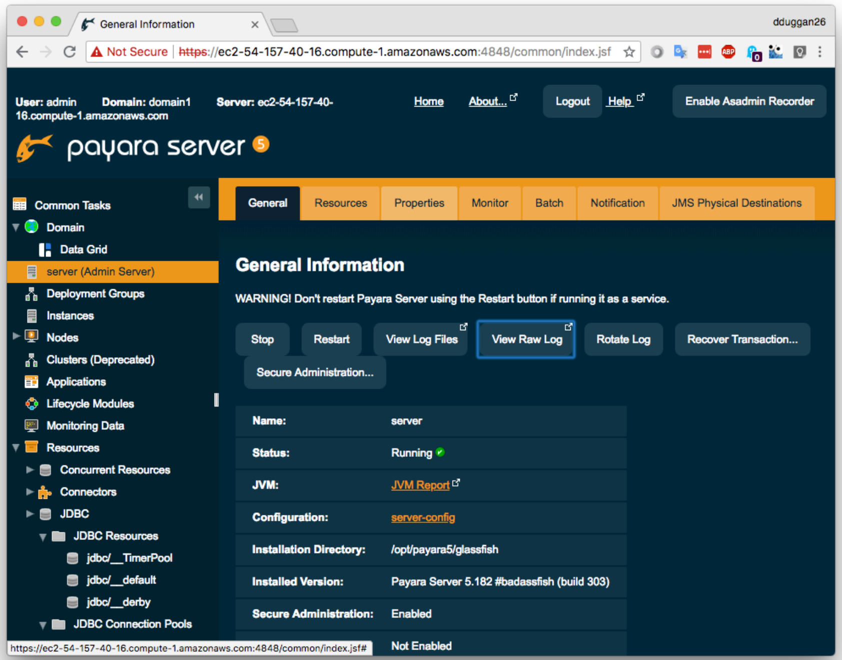
Enterprise Application

n Name: * ChatApp
Application Name will be overridden if module name is defined in the Module Descriptors.

plication Name: ☐
Forces Application Name even if module name is defined in the Module Descriptors.

rvers: server

If you have any problems, view the raw logs to investigate what is going on:



User: admin Domain: domain1 Server: ec2-54-157-40-16.compute-1.amazonaws.com

General Information

WARNING! Don't restart Payara Server using the Restart button if running it as a service.

Name:	server
Status:	Running
JVM:	JVM Report
Configuration:	server-config
Installation Directory:	/opt/payara5/glassfish
Installed Version:	Payara Server 5.182 #badassfish (build 303)
Secure Administration:	Enabled
	Not Enabled

Step 8: Enable autostart of the database and application server on boot

As root on the database server instance, create the file `/etc/systemd/system/cs594db.service`:

```
[Unit]
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start cs594db
ExecStop=/usr/bin/docker stop cs594db

[Install]
WantedBy=multi-user.target
```

Now you can start the database as a service:

```
$ systemctl start cs594db
```

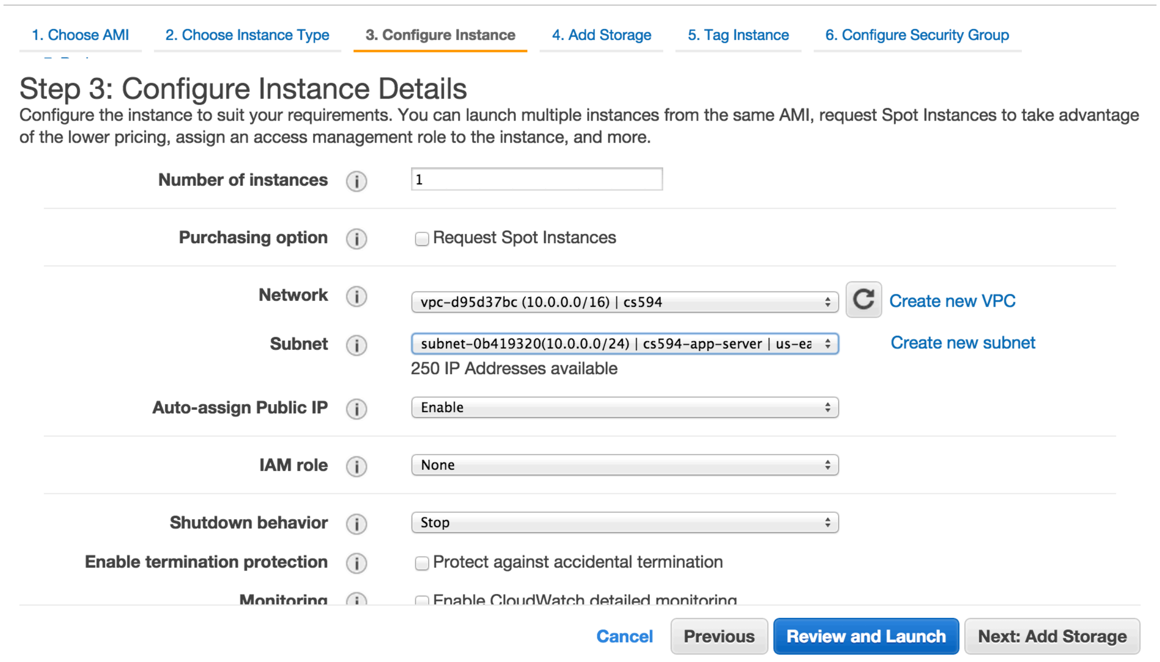
Enable the service to be executed during boot:

```
$ systemctl enable cs594db
```

Similarly set up the Payara server to run as a boot service (call it cs594, for example).

You should not leave the EC2 instance running, since you may incur bills of hundreds of dollars from Amazon if you do this. Instead, leave your instances and VPC stopped, and use IAM to provide graders with access to your VPC and EC2 consoles, so they can start the instance, and grant them access to your EC2 instances via ssh public keys. See the separate document detailing how to grant graders access to the instance.

When you launch your instances, you will launch them into the VPC that you have created. Be careful to choose the right subnet for launching, following the naming scheme prescribed above for your subnets:



Step 3: Configure Instance Details

Configure the instance to suit your requirements. You can launch multiple instances from the same AMI, request Spot Instances to take advantage of the lower pricing, assign an access management role to the instance, and more.

Number of instances	1
Purchasing option	Request Spot Instances
Network	vpc-d95d37bc (10.0.0.0/16) | cs594
Subnet	subnet-0b419320(10.0.0.0/24) | cs594-app-server | us-ea 250 IP Addresses available
Auto-assign Public IP	Enable
IAM role	None
Shutdown behavior	Stop
Enable termination protection	Protect against accidental termination
Monitoring	Enable CloudWatch detailed monitoring

Be sure to assign the correct security group that you set up earlier, that limits access to the instance. You should also enable auto-assignment of a public IP address for the app server instance, so you can access it remotely from your own machine. In an enterprise setting, you would associate an elastic IP address with this instance, but that is obviously overkill for this course (and Amazon would revoke the IP address after a period of non-use).

In general, instances launched into VPC cannot be EBS-backed, except at the very smallest tier which may be too small to host an application server or a database server. However you can go ahead and launch even if the original instance when you created it was EBS-backed, and it will give you the option of magnetic disk or SSD. Magnetic storage, the cheapest option, is obviously sufficient for our purposes. For the database server instance, you will have added an EBS volume for database storage to the instance when setting it up, and if you launch into the private subnet from an AMI, this volume will be attached to the instance as magnetic storage.

In addition, do the following to save your personal information in the app server instance:

```
cd
echo “YOUR-NAME” > info.txt
echo “YOUR-CWID” >> info.txt
echo “YOUR-EMAIL” >> info.txt
```

For your submission, provide a video that provides each of the following:

a. View of your VPC console showing the subnets for the VPC.
b. View of the custom routing table for the public subnet.
c. View of the security groups protecting access to instances in the public and private subnets, as well as for the NAT instance.
d. View of your AWS Console showing the running instances, including the external DNS for the app server.
e. View of your Payara Admin Console, showing `Resources|JDBC|Connection Pools`.
f. View of your local browser window showing the “`Hello, World`” application being deployed and running in your app server instance (same DNS as above). Access this instance via its Web page to demonstrate it running.

You are also strongly encouraged to back up your instances as Amazon machine images (AMIs). This way, if an instance ever becomes corrupted in some way, you can instantiate a new version from the AMI that you have created. However you do not need to share the AMI with the graders, they will just be accessing the instances that you have stopped. Be sure to include, in the report for this and later assignment submissions, complete instructions on how to start the instances from your VPC and EC2 consoles. Payara and postgresql should start automatically when the instances start, if you have set them up right.

Record a short mpeg video of a demonstration of your deployment working. In this video, demonstrate your logging in to the application server administration console, installation of the `ChatApp` application, and finally your running this application.

Make sure that your name appears at the beginning of the video. For example, display the contents of a file that provides your name. *Do not provide private information such as your email or cwid in the video*. Be careful of any “free” apps that you download to do the recording, there are known cases of such apps containing Trojan horses, including key loggers.

Your submission should be uploaded via the Canvas classroom, as a zip file. This zip file should have the same name as your Canvas userid. It should unzip to a folder with this same name, which should contain the files and subfolders with your submission.

It is important that you provide a document that documents your submission, included as a PDF document in your submission root folder. Name this document README.pdf. This should document how you have set up your virtual private cloud, as described above.