

# CS 548—Spring 2023

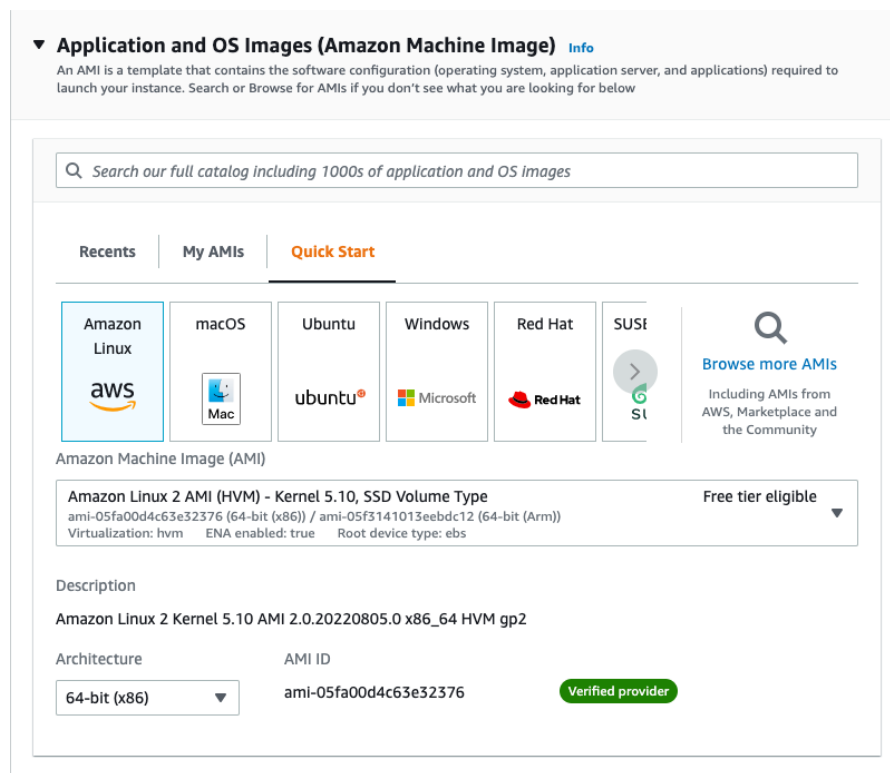
## Enterprise Software Architecture and Design

### Assignment One—Cloud Computing

In this assignment, you will set up a simple web application in the cloud. You will set this up in the Amazon Elastic Compute Cloud<sup>1</sup> (EC2), which provides an infrastructure-as-a-service (IaaS) for configuring a custom software stack. You will use Docker to ensure you will have the same environment for local development and for deployment<sup>2</sup>, and to isolate application components that you deploy.

#### Step 1: Launch an EC2 instance

Launch an EBS-backed EC2 instance. 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.



<sup>1</sup> <https://console.aws.amazon.com/ec2>.

<sup>2</sup> If your development machine is Windows or Intel Mac, you should install Docker Desktop. This will install a Linux VM, and the docker command line tool will manage containers running on this VM. This will allow you to develop on your local machine, and then deploy your working application on EC2. Unfortunately, the Payara image we will be using is not currently available for Apple ARM, so you will have to develop on EC2. Alternatively, you can for now develop natively on your Mac and demonstrate deployment on EC2.

Since you will be installing both Postgresql and Payara on your instance, a micro-sized instance will not have enough memory. Pick a small-sized instance. Provided you do not leave an instance running, your charges should be minimal:

▼ Instance type [info](#)

Instance type

t2.small

Family: t2    1 vCPU    2 GiB Memory

On-Demand Linux pricing: 0.023 USD per Hour

On-Demand Windows pricing: 0.032 USD per Hour

[Compare instance types](#)

If this is your first time creating an EC2 instance, you will want to create a (RSA) key pair and save this on your laptop. You will need the key pair in order to access the EC2 instance. Be sure to protect this key pair and never share it with anyone.

▼

**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*

cs548-keypair

▼

↻

Create new key pair

For network settings, the EC2 wizard will automatically create a virtual private cloud (VPC) and public subnet with that cloud for your instance. You must set up a firewall policy to protect your instance. **You should only access from your laptop.** By default, the EC2 wizard will create a rule allowing you to ssh to your instance to administer it:

**▼ Network settings**

VPC - required Info  
vpc-83d402e4 (default) ↻  
172.31.0.0/16

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  
cs548-policy

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  
Firewall policy for C5548 EC2 Instance

---

### Inbound security groups rules

Type	Protocol	Port range	Action
ssh	TCP	22	<button>Remove</button>

Source type  
My IP

Source  
[Add CIDR, prefix list or security group](#)

Description - optional Info  
ssh

You should add other rules to the firewall policy, to allow access to ports 8080 and 8181 (accessing the application) and 5432 (for accessing the database server). The latter is in case you want to connect to the database using your IDE, which you would not be doing in a production environment. **Make sure that your instance can only be accessed from your IP address.**

▼ Security group rule 2 (TCP, 8080, 173.54.213.107/32, app server) <span>Remove</span>		
Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>
Custom TCP ▼	TCP	8080
Source type <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
My IP ▼	<input type="text" value="Add CIDR, prefix list or security"/>	app server
▼ Security group rule 3 (TCP, 8181, 173.54.213.107/32, app server ssl) <span>Remove</span>		
Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>
Custom TCP ▼	TCP	8181
Source type <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
My IP ▼	<input type="text" value="Add CIDR, prefix list or security"/>	app server ssl
▼ Security group rule 4 (TCP, 4848, 173.54.213.107/32, app server admin) <span>Remove</span>		
Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>
Custom TCP ▼	TCP	4848
Source type <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
My IP ▼	<input type="text" value="Add CIDR, prefix list or security"/>	app server admin
▼ Security group rule 5 (TCP, 5432, 173.54.213.107/32, database server) <span>Remove</span>		
Type <a href="#">Info</a>	Protocol <a href="#">Info</a>	Port range <a href="#">Info</a>
Custom TCP ▼	TCP	5432
Source type <a href="#">Info</a>	Source <a href="#">Info</a>	Description - optional <a href="#">Info</a>
My IP ▼	<input type="text" value="Add CIDR, prefix list or security"/>	database server

For storage, EC2 will allocate an 8G virtual disk for the instance, to hold the OS and application. However, if you terminate the instance, this disk is deleted. Allocate an additional EBS volume to hold the persistent state of the database:

▼ Configure storage
Info
Advanced

1x
8
GiB
gp2
▼
Root volume

1x
1
GiB
gp3
▼
EBS volume
Remove

Free tier eligible customers can get up to 30 GB of EBS General Purpose (SSD) or Magnetic storage
X

Add new volume

0 x File systems
Edit

Once you have launched your instance, you can see it running in your EC2 dashboard. Note the public DNS address and IP address for your running instance, you will use one of these, with the key pair, to ssh to the instance to set it up:

Instance: i-0802694ea03246d38 (CS548)

Details
Security
Networking
Storage
Status checks
Monitoring
Tags

▼ Instance summary Info

Instance ID  
i-0802694ea03246d38 (CS548)

Public IPv4 address  
34.229.146.77 | open address

Private IPv4 addresses  
172.31.14.172

IPv6 address  
-

Instance state  
Running

Public IPv4 DNS  
ec2-34-229-146-77.compute-1.amazonaws.com | open address

Hostname type  
IP name: ip-172-31-14-172.ec2.internal

Private IP DNS name (IPv4 only)  
ip-172-31-14-172.ec2.internal

Elastic IP addresses  
-

Answer private resource DNS name  
IPv4 (A)

Instance type  
t2.small

AWS Compute Optimizer finding  
Opt-in to AWS Compute Optimizer for recommendations.  
Learn more

Auto-assigned IP address  
34.229.146.77 [Public IP]

VPC ID  
vpc-83d402e4

Auto Scaling Group name  
-

IAM Role  
-

Subnet ID  
subnet-f99ab38f

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

Instance: i-0802694ea03246d38 (CS548)

Details
Security
Networking
Storage
Status checks
Monitoring
Tags

▼ Root device details

Root device name  
/dev/xvda

Root device type  
EBS

EBS optimization  
disabled


▼ Block devices

Filter block devices

Volume ID	Device name	Volume size (GiB)	Attachment status	Attachment time
vol-06b49064f4545ae45	/dev/xvda	8	Attached	Sat Sep 10 2022 1
vol-023fad410c474d00f	/dev/sdb	1	Attached	Sat Sep 10 2022 1

## Step 2: Mount the disk for the database

Use ssh to access your instance:

A screenshot of a terminal window. The title bar shows a red, yellow, and green window control icon, followed by the text 'ec2-user@ip-172-31-14-172:~ — ssh -i cs548-keypair.pem -l ec2-user ec...'. The terminal content shows the command 'ssh -i cs548-keypair.pem -l ec2-user ec2-34-229-146-77.compute-1.amazonaws.com' being executed. It displays the host's authenticity warning, the key fingerprint 'ED25519:twmA5QQemCCZmpKwc+ZAajPEL5dDAajB2dZmDAWqMuE', and asks for confirmation to continue. The user responds 'yes'. A warning message states that the host has been added to the known hosts list. Below this is the Amazon Linux 2 AMI logo, which consists of a stylized 'A' made of horizontal lines. The terminal then shows the URL 'https://aws.amazon.com/amazon-linux-2/' and a message that 3 packages are needed for security updates out of 8 available. It prompts the user to run 'sudo yum update'. The user enters the command, and the prompt changes to '[ec2-user@ip-172-31-14-172 ~]\$'.

```
ec2-user@ip-172-31-14-172:~ — ssh -i cs548-keypair.pem -l ec2-user ec...
[Home-MBP:Docs<7>] ssh -i cs548-keypair.pem -l ec2-user ec2-34-229-146-77.compute-1.amazonaws.com
The authenticity of host 'ec2-34-229-146-77.compute-1.amazonaws.com (34.229.146.77)' can't be established.
ED25519 key fingerprint is SHA256:twmA5QQemCCZmpKwc+ZAajPEL5dDAajB2dZmDAWqMuE.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added 'ec2-34-229-146-77.compute-1.amazonaws.com' (ED25519) to the list of known hosts.

  _ _ | _ _ | _ _ )
  _ | ( _ /      Amazon Linux 2 AMI
  _ _ | \ _ _ | _ _ |

https://aws.amazon.com/amazon-linux-2/
3 package(s) needed for security, out of 8 available
Run "sudo yum update" to apply all updates.
[ec2-user@ip-172-31-14-172 ~]$ sudo yum update
```

**Install any software updates.** It is very important that you keep the software on your machine up-to-date, particularly with security updates. Then 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

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). If you reboot, you will have to restart the docker service.

## 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. First, create a virtual network and then the container that runs on that network. *For development only, it may be useful to add “-p 5432:5432” as an option when running the database container, to expose the port so IntelliJ can connect to the database server:*

```
[ec2-user] docker network create --driver bridge cs548-network
```

```
[ec2-user] docker pull postgres
```

```
[ec2-user] docker run -d --name cs548db --network cs548-network -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). The container will run in the virtual network `cs548-network`, with virtual host name `cs548db`. 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>
```

You can stop a container using `docker stop`, remove a container using `docker rm`, and remove all stopped docker containers using `docker container prune`.

## Step 5: Initialize the database user

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<sup>3</sup>:

```
[ec2-user] docker run -it --rm --network cs548-network postgres
/bin/bash
# createuser cs548user -P --createdb -h cs548db -U postgres
Enter password for new role: YYYYYY
Enter it again: YYYYYY
Password: XXXXXX (see above)
# exit
```

Note that the `createuser` command connects to the virtual host `cs548db`, as superuser `postgres`. Run a `psql` shell to create the database for this user<sup>4</sup>:

```
[ec2-user] docker run -it --rm --network cs548-network postgres psql -h
cs548db -U postgres
postgres=# create database cs548 with owner cs548user;
postgres=# \q
```

---

<sup>3</sup> Alternatively, you might use `docker exec` to attach to a running container and execute a command, such as starting a shell.

<sup>4</sup> If you later find you need to drop the database because you've changed the schema, you can do so (as user `postgres`) with this `psql` command: `drop database cs548 with (force);`



## Step 6: Create the application container

We will be using the Payara Micro docker image for the server runtime<sup>5</sup>. You should have used scp to copy the Web archive (war) file provided to the home folder on the EC2 instance. Now create a directory called cs548-payara, move the war file there, and navigate into this directory:

```
[ec2-user] mkdir cs548-payara
[ec2-user] mv chat-webapp.war cs548-payara
[ec2-user] cd cs548-payara
```

In this directory, create a file called Dockerfile that copies the WAR file for the application to the container file system<sup>6</sup>:

```
FROM payara/micro:6.2023.6-jdk17
COPY --chown=payara:payara chat-webapp.war ${DEPLOY_DIR}
CMD [ "--contextroot", "chat",
      "--deploy", "/opt/payara/deployments/chat-webapp.war" ]
```

Save this file, and create a custom server image:

```
[ec2-user] docker build -t cs548/chat .
[ec2-user] docker images
[ec2-user] cd ..
```

Create and start the server container, on the same virtual network as the database. 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 chat --network cs548-network -p
8080:8080 -p 8181:8181 -e DATABASE_USERNAME=cs548user -e
DATABASE_PASSWORD=YYYYYY -e DATABASE=cs548 -e DATABASE_HOST=cs548db
cs548/chat
```

You specify the properties for the database connection as environment variables when you run the application container<sup>7</sup>. The log for the application (accessed using the docker logs command on that container) should be of the form:

```
[2023-09-07T20:27:08.519+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1
_ThreadName=main] [timeMillis: 1694118428519] [levelValue: 800] [[
```

---

<sup>5</sup> <https://hub.docker.com/r/payara/micro>.

<sup>6</sup> There are three lines in the dockerfile, the third line is broken to fit in this document.

<sup>7</sup> This is **not** a best practice. For example, anyone on the machine you are running on that looks at the currently running processes will see the database user password. It is at least better than putting the password in the source code. A better approach would be to use something like Docker Secrets to store the database credentials in an encrypted volume that is mounted by a container for this microservice.

```

{
  "Instance Configuration": {
    "Host": "...",
    "Http Port(s)": "8080",
    "Https Port(s)": "",
    "Instance Name": "Fancy-Barramundi",
    "Instance Group": "MicroShoal",
    "Hazelcast Member UUID": "cbf92101-ad52-4b07-b9de-e21a0715ef0e",
    "Deployed": [
      {
        "Name": "chat-webapp",
        "Type": "war",
        "Context Root": "/chat"
      }
    ]
  }
}
]]

[2023-09-07T20:27:08.526+0000] [] [INFO] [] [PayaraMicro] [tid: _ThreadID=1
_ThreadName=main] [timeMillis: 1694118428526] [levelValue: 800] [[

Payara Micro URLs:
http://...:8080/chat

]]

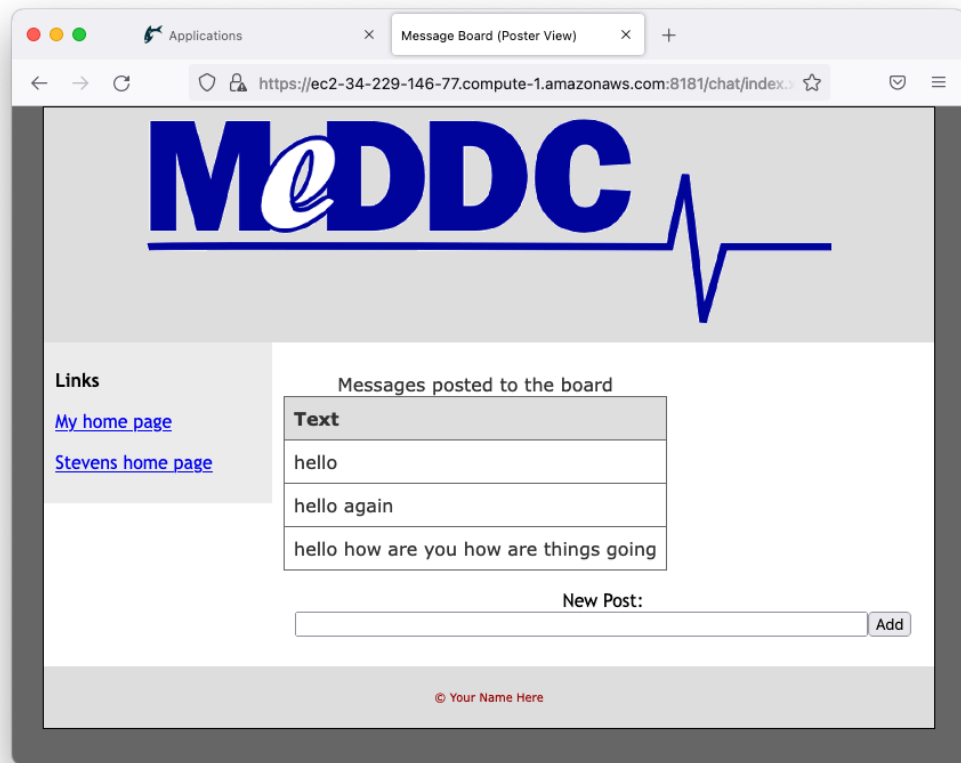
```

If there is no Payara Micro URL, then that means that deployment has failed (see below). If you have successfully deployed the app, you can access it via the URL<sup>8</sup>:

`https://ec2-instance-public-dns-name:8080/chat`

---

<sup>8</sup> The Community Edition of Payara Micro does not support HTTPS, so there is no listener at port 8181.



If you have any problems, you can view the raw logs using the `docker logs` command. For example, save the logs into a file and then view the last 100 lines of the file:

```
[ec2-user] docker logs container-id >& LOG
[ec2-user] tail -100 LOG
```

It is well worth your while learning at least some rudimentary bash commands, and how to view a file using the vim editor<sup>9</sup>.

---

<sup>9</sup> Bash and vim are available in MacOS. On Windows, install Cygwin.

## Step 7: Enable autostart of the database and application on boot

As root, create the file `/etc/systemd/system/cs548db.service`:

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

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start cs548db
ExecStop=/usr/bin/docker stop cs548db

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

Now you can start the database as a service:

```
$ sudo systemctl start cs548db
```

Enable the service to be executed during boot:

```
$ sudo systemctl enable cs548db
```

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

## Submission

For your submission, provide videos and a report in PDF format that describes what you did, including each of the following:

- a. Screenshots of your AWS Console showing the instance running and the volumes you have allocated on EBS.
- b. Output of Linux command “`fdisk -l`” in the instance.
- c. Output of running `docker inspect` on the database and application containers.
- d. Information on how to access your EC2 console using IAM permissions (see the additional specification for this).
- e. Video of a demonstration of your deployment working. In this video, demonstrate your running this application (adding some messages to test the database connection), with the full URL clearly visible in the browser, then show the log including deploying the application and the application contacting the database (using `docker logs` on the application container). Include your name in at least one message.
- f. Provide a completed rubric with your submission (see the provided rubric).

**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 instance stopped, and use IAM to provide graders with access to your EC2 console, so they can start the instance. Both the database and application must automatically start when the instance boots. See the separate document detailing how to grant graders access to the instance.

You are also strongly encouraged to back up your instance as an Amazon machine image (AMI). This way, if the 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 instance that you have stopped. Be sure to include, in the report for this and later assignment submissions, complete instructions on how to start the instance from your EC2 console. The application and postgresql should start automatically when the instance starts, if you have set it up right.

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 video must be MP4 format!**

Your submission should be uploaded via the Canvas classroom, as a zip file. This zip file should have the same name as your Stevens username. 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 the video(s) that you provide demonstrating that you have correctly set up your cloud environment. You should also provide a completed rubric.**