Homework 1b: Rocking AWS!

Start Assignment

- Due Monday by 12pm
- Points 10
- Submitting a text entry box, a website url, or a file upload

WELCOME to AWS Academy!

Hopefully you have an email with an invite to AWS academy (check your spam), where you will be able to get access to $50 of credits in the Learner Lab. Some of you have extensive experience with this, and for some of you it is your first time seeing it. Feel free to check it out, and start with the little exercises below, but don't let us limit you if you want to go further! We will post the next assignment later this week...

Part I: Overview

In this part of the assignment, you will setup your AWS learner's account so that you can run your future assignments remotely on AWS (short intro video **here (https://youtu.be/koHCloGH8ns)** !). You will also setup an **EC2 (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html)** on AWS, and experiment with running your server from Homework 1a on AWS!

Step I - Setup Account

Follow this guide: **AWS Academy Learner Lab - Student Guide.pdf (https://northeastern.instructure.com/courses/239676/files?preview=39392413)** . Note that the new link for login is **https://awsacademy.instructure.com/ (https://awsacademy.instructure.com/)** .

Make sure you are aware of your budget, and and why you will always want to cleanup resources you don't need!

Step II - Create EC2 Instance

Change the region on the top right to **Oregon (us-west-2)**. Search for EC2 on the top left.

Click on *Launch an instance*, refer to **the official tutorial (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html)** . You are free to explore different options but we recommend:

- OS: *Amazon Linux 2023*
- Instance Type: *t2.micro* (Free tier for limited costs)
- Create a key pair RSA .pem key pair (or .ppk with PuTTY)
- Create a security group that allows SSH traffic from *My IP.*
- Advanced Details -> IAM Instance Profile -> LabInstanceProfile

- Hit the *Launch Instance* button on the left!

You should see your instance up and runnng once you click into *Instances* on the left of EC2 main menu. Wait for it to finish initialization.

Select the instance and click *connect.* Use provided example ssh command to establish a connection to your EC2 instance, note you will need your key in that directory, and you will have to have the permissions set correctly. You will need SSH on your system if you haven't already.

Step III - Cross Compile Your Go-Gin code!

- Basically, we want to cross compile the Go-Gin code from Homework 1a on your local machine into binary executable file that you can run on EC2 directly! Make sure that you change the original code from `localhost:8080` to `0.0.0.0:8080`
    - What you need is to specify the target Operating System (OS) and architecture when compiling and build your *Go* application.
- After you build and test your code, open a bash terminal, navigate to the main directory of your Go-Gin code file (where your main.go resides),
    - For Windows, you will need to open a git-bash terminal from VS Code. Check **here (https://code.visualstudio.com/docs/sourcecontrol/intro-to-git#_git-bash-on-windows)**
- Run the following command to cross-compile your code
    - Here, GOOS sets the target OS to linux,
    - GOARCH sets the target architecture to amd64. (For AWS Linux, it will either be amd64 or arm64, try the other if the first didn't work)
    - For list of available OS and GOARCH, check **here (https://go.dev/doc/install/source#environment)**
    - go build command builds the executable file with code resides on main.go (the file name you write your code)
    - -o specifies the output name of the executable binaries file
    - `<your-filename>` can be something like, *newmain* or whatever you like!
    - Useful reference for compilation tutorial **here (https://go.dev/doc/tutorial/compile-install)** , command options **here (https://pkg.go.dev/cmd/go)**

```
GOOS=linux GOARCH=amd64 go build -o <your-filename> main.go
```

Step IV: Upload your binary executable file to EC2 and Run it.

- You will need to set up a security group to access port 8080. Here's how to do it:
    1. In your AWS console, go to the EC2 service
    2. Find your running instance in the instance list
    3. Click on your instance to select it

4. Look at the details below - you should see a "Security groups" section
5. Click on the security group name (it'll be something like "sg-xxxxxxxxx")

This will show you the current rules. Look at the "Inbound rules" section.

You'll see rules for SSH (port 22) but nothing for port 8080!

To fix this, you need to add an inbound rule that allows:

- Type: Custom TCP
- Port: 8080
- Source: MyIP or ... 0.0.0.0/0 (this opens up to allow traffic from anywhere, is that ok? ask Claude!)

- Now, from your machine, try SSHing into EC2, and make a directory of your choice using mkdir command

```
#create your directory, if permission denied, add sudo
(sudo) mkdir <your_dir_name>

#run following command to allow permission to upload files
#if permission denied, add sudo
(sudo) chmod -R 777 <your_dir_name>
```

- Open a command prompt window from your local machine, and use **scp (https://www.linuxfoundation.org/blog/blog/classic-sysadmin-how-to-securely-transfer-files-between-servers-with-scp)** to copy the file onto EC2

```
sudo scp -i <path to your pem file for aws private key, include.pem extension> <path to your execu
table binary cross-compiled> ec2-user@<EC2_IP_ADDR>:<folder of your choices>
```

- Note the whitespace, if I separate the command by whitespace I should see something like

```
sudo
scp
-i
<path to your pem file for aws private key, include.pem extension>
<path to your executable binary cross-compiled>
ec2-user@<EC2_IP_ADDR>:<folder of your choices>
```

- Notes for above commands
  - *sudo* is not required if you open the command prompt from Window with administrator right
  - for the paths, the first two (perm file and compiled file) are paths on your local machine
  - ec2-user is the login name you used when ssh into your ec2 instance
  - EC2_IP_ADDR -> this is the public IP address of your EC2 instance
  - For folder path on EC2, if you previously mkdir folder on directory when you logged in, it will typically be /home/ec2-user/

- Now you can try run your go server, navigate to the directory where your upload the file

```
#run your go-server file
./<your-filename>
```

- do this to change permissions if you get permission denied `sudo chmod -R 777 <your-filename>`
- Now, using curl / Postman, send an HTTP request to your server, your address should be something like

  `curl -v http://<your public IP address>:8080/albums`

  Or if you are ssh'd into your instance, you can test it locally with curl, for example,

  `curl localhost:8080/albums`

Once you get this far, life looks pretty good! First mission accomplished! In 3 weeks you'll be able to do all this in your sleep :)

Notes

Some things that might come up based on first experiences with the Learner Lab:

- Download a new key pair when you launch your first instance. You can then use this for all subsequent instances you launch
- a .cer file seems to be the same as a .pem file, so you can use that in ssh commands

Here is some troubleshooting guide if something is not working!

- If you receive permission error when uploading / running the file
  - try use sudo chmod -R 777 <dir/filename> command to change the permission
- If you receive other error when try to run the executable file on EC2
  - try compile again the file from local into other GOARCH (arm64 or amd64)
- If you can see the server running on SSH interface, but cannot receive valid http responses
  - check your security group setting to allow inbound traffic on required port from your PC
  - check your url path information

STAY TUNED! There will be more glitches here that we will try to keep on top of! :)

Please add your teaching team member and myself (mcoady) to your private Khoury github repo, and be ready to demonstrate your code in your 1:1 meeting with your team in your weekly meeting. Just to be sure we do not miss anything, also upload a .zip file here for our marking.

Part II: Learning Outcomes

CONGRATS, you moved over to AWS, learned about ssh and security groups... BUT WOW this is painful to set up compared to what we saw on GPC!

You have a working virtual machine hosted remotely by AWS through Learner's Lab. Questions for you to explore:

- What is a Virtual Machine? How is it running a program on a **VM (https://aws.amazon.com/what-is/virtual-machine/)** different from running it locally?
- You allowed the ssh connection from your IP address to the EC2 instance. What do you need to do if your IP address changes?
- Your EC2 will be assigned a different IP everytime it starts running from stopped, terminated, or restarted state. See how **Elastic IP address (https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/elastic-ip-addresses-eip.html)** can help, at a cost.
- What level of service are you "paying" for with your selected instance type? See **instance type (https://aws.amazon.com/ec2/instance-types/?trk=8c0f4d22-7932-45ae-9a50-7ec3d0775c47&sc_channel=ps&ef_id=CjwKCAjwy7HEBhBJEiwA5hQNohpgp2SVMVW3HX3JmDI8QAvD_BwE:G:s&s_kwcid=AL!4422!3!536392632643!p!!g!!amazon%20ec2%20pricing!11346198414!111K4H9T8ZoyY8wrSaB_zqpRL&gclid=CjwKCAjwy7HEBhBJEiwA5hQNohpgp2SVMVW3HX3JmDIhRC7tX8QAvD_BwE)**. Why would you care about these specs?

What was different between getting your little backend going on GCP and AWS?

What testing did you do, and how did you do it?

Use Claude in Learner mode to walk through the code and the concepts involved, and make a small slide show (5 slides) overviewing the concepts you learned, or find most interesting. Be ready to share these in your mock interview.

Part III: Performance Testing and Understanding Response Times

Now that your Go web service is running in the cloud, let's investigate how it performs under load. Create a Python script that will help you understand the response time characteristics of your service.

Step 1: Create a Load Testing Script

Write or generate a Python script with the following requirements:

1. **Duration**: This one is set up to run for 30 seconds, but you can experiment with longer times!
2. **Requests**: Send GET requests to your EC2 instance's public IP
3. **Data Collection**: Record the response time for each request
4. **Visualization**: Plot the response times in a histogram or scatter plot

Here's a template to get you started:

```
import requests
import time
import matplotlib.pyplot as plt
import numpy as np
```

```
from datetime import datetime, timedelta

def load_test(url, duration_seconds=30):
    response_times = []
    start_time = time.time()
    end_time = start_time + duration_seconds

    print(f"Starting load test for {duration_seconds} seconds...")

    while time.time() < end_time:
        try:
            start_request = time.time()
            response = requests.get(url, timeout=10)
            end_request = time.time()

            response_time = (end_request - start_request) * 1000  # Convert to milliseconds
            response_times.append(response_time)

            if response.status_code == 200:
                print(f"Request {len(response_times)}: {response_time:.2f}ms")
            else:
                print(f"Request {len(response_times)}: Failed with status {response.status_code}")

        except requests.exceptions.RequestException as e:
            print(f"Request failed: {e}")

    return response_times

# Replace with your EC2 public IP
EC2_URL = "http://YOUR-EC2-PUBLIC-IP:8080/albums"

# Run the test
response_times = load_test(EC2_URL)

# Plot the results
plt.figure(figsize=(12, 8))

# Histogram
plt.subplot(2, 1, 1)
plt.hist(response_times, bins=50, alpha=0.7, color='blue')
plt.xlabel('Response Time (ms)')
plt.ylabel('Frequency')
plt.title('Distribution of Response Times')

# Scatter plot over time
plt.subplot(2, 1, 2)
plt.scatter(range(len(response_times)), response_times, alpha=0.6)
plt.xlabel('Request Number')
plt.ylabel('Response Time (ms)')
plt.title('Response Times Over Time')

plt.tight_layout()
plt.show()

# Print statistics
print(f"\nStatistics:")
print(f"Total requests: {len(response_times)}")
print(f"Average response time: {np.mean(response_times):.2f}ms")
print(f"Median response time: {np.median(response_times):.2f}ms")
print(f"95th percentile: {np.percentile(response_times, 95):.2f}ms")
print(f"99th percentile: {np.percentile(response_times, 99):.2f}ms")
print(f"Max response time: {max(response_times):.2f}ms")
```

Step 2: Install Required Dependencies

Before running the script, install the required Python packages:

```
pip install requests matplotlib numpy
```

Step 3: Understanding the Results

After running your script, you should observe a characteristic "long tail" distribution in your response times. This means:

- **Most requests** complete quickly (low latency)
- **Some requests** take significantly longer (high latency outliers)

Questions to Consider:

1. **Distribution Shape**: Does your histogram show a long tail? What percentage of requests fall into the "slow" category?
2. **Consistency**: Are the response times consistent throughout the 30-second window, or do you see patterns or spikes?
3. **Percentiles**: What's the difference between your median (50th percentile) and 95th percentile response times? A large gap indicates high variability.
4. **Infrastructure Impact**: How might running this single container on a basic EC2 instance contribute to the response time variability you're seeing?
5. **Scaling Implications**: Based on this data, what do you think would happen if you had 100 concurrent users instead of sequential requests?
6. **Network vs. Processing**: Are the longer response times due to network latency, server processing time, or both? How could you investigate this further?

This exercise demonstrates why understanding performance characteristics is crucial before deploying services to production. The "tail latency" you're observing is a real-world phenomenon that affects user experience and system reliability.

WHAT TO HAND IN!

Part I: Start your Khoury Github repository, and share the link with your TA (submit the link here, with this assignment). Capture and submit some screen shots of your exciting output! :). Be ready to show your code, and all of this *running* in your mock interview.

Part II: Upload screen shots of your results, add some notes for your observations of what you are seeing! Be ready to discuss this in your mock interview.

Mock Interview Scoring!

Criteria	Ratings					Pts
Code Quality	**2 pts** **Full Marks** Excellent! Very clear and concise, outstanding documentation, readability, and maintainability. Beautiful code!	**1.5 pts** **Good quality** Clarity could be improved, in particular documentation.	**1 pts** **Could use improvement** Code readability and maintainability could be improved, this will come with practice!	**0.5 pts** **Major improvement expected** Code quality is poor, not something that could easily be shared with a team in a code walk.	**0 pts** **No code submitted by due date.**	2 pts
Code Completion	**2 pts** **Full Marks** Excellent! Code is complete and runs without error, thought has been put into testing and error checking.	**1.5 pts** **Small bugs** Minor runtime errors, cleared up in interview time, but otherwise code runs correctly.	**1 pts** **Needs improvement** Code does not run and suffers from major errors, and requires further testing.	**0.5 pts** **Needs major improvement** Some code was submitted, but has significant errors and has not been subjected to adequate testing.	**0 pts** **No Code was submitted by due date.**	2 pts
Code understanding	**2 pts** **Full Marks** Excellent job in answering questions about the code and analysis posed by both the TA and other group members!	**1.5 pts** **Good answers** Answers to questions were clear, but there is room to improve and go deeper into technical explanations.	**1 pts** **Communication improvements expected** Ability to show a deeper understanding is expected.	**0.5 pts** **Major Communication Issue** Answering questions will require more practice, both in terms of listening carefully to the question that is asked, and thinking deeply about the tradeoffs in the answers.	**0 pts** **No Code was submitted by due date.**	2 pts

Criteria	Ratings					Pts
Listening and Engaging with team	**2 pts** **Full Marks** Excellent job of contributing to the group discussion! Listening closely to others, asking insightful questions, helping consider alternatives!	**1.5 pts** **Strong interaction** Some elements of group interaction can be improved, in particular making questions/comments specific and insightful.	**1 pts** **Some improvements expected** Ensure listening is uninterrupted and the discussion is progressing. Do not repeat questions/comments that have already been shared.	**0.5 pts** **Major Engagement Issue** Did not appear to be listening, and was unable to contribute beyond superficial comments.	**0 pts** **No questions asked** No interaction with team members	2 pts
Understanding of Concepts and Tradeoffs	**2 pts** **Full Marks** Excellent job of mapping concrete code examples to high level concepts! This was demonstrated throughout the interview, and made abundantly clear in the posting on the reading on Piazza!	**1.5 pts** **Good sense of tradeoffs** Is able to see the connection to higher level concepts, and is working on communicating the relationships between code examples and concepts. Posting on Piazza was strong.	**1 pts** **Room for improvement** With practice, the ability to reason about concepts and tradeoffs and communicate clear points will be improved. Developing this new skill, posting on Piazza showed promise.	**0.5 pts** **Needs Major improvement** Posting on Piazza needs more depth. Currently not connecting examples to concepts and tradeoffs, but aware of the nature of this relationship.	**0 pts** **No Marks** No posting on the reading on Piazza.	2 pts
					Total Points: 10	