Botika DevOps Intern Test - Lareza Farhan Wanaghi

Objectives

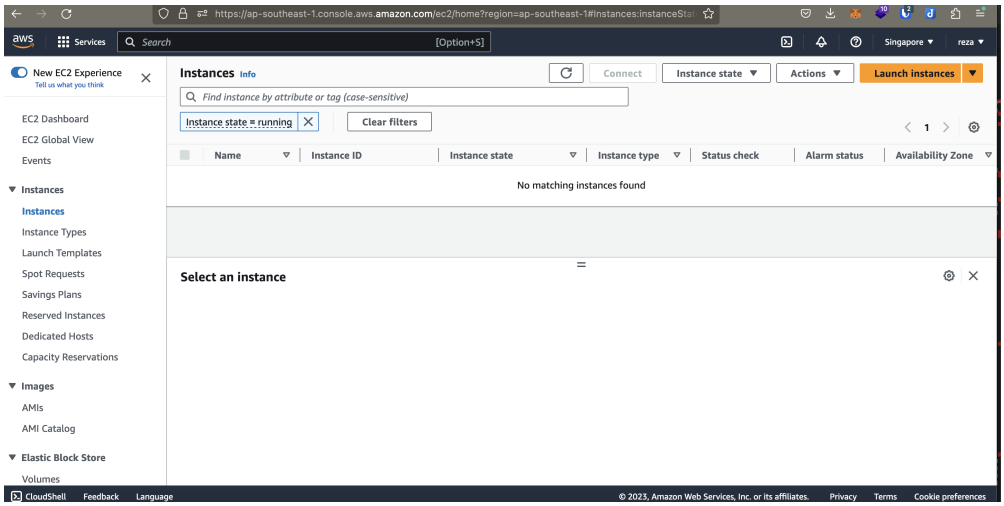
- Create 200 users with usernames in the format of user1-200. Each user is assigned a UID ranging from 1500 to 1699, and their passwords are generated following the pattern uid+username, for instance (1500+user1).
- Establish a request bin at https://webhook.site. Every user created will be directed to the URL of the request bin you configured (HTTP). HTTP request details:
  - Method: POST
  - Content-Type: application/json
- Develop an HTTP server application using your preferred programming language (such as Node.js, Go, or Python). The application will respond with the hostname and the current real-time server time, presented in JSON format. Package this program into a Docker image and launch at least two containers.
- Configure Nginx as a reverse proxy and load balancer for these servers, adhering to the following requirements:
  - Different displays should be presented when accessed via IP address and domain.
  - Ensure two containers are managed to appear as a single endpoint to users.
  - Control access to hidden files.
  - Automatically redirect users to HTTPS if they access via HTTP.
  - Implement a rate limiter.

Resolutions

1. Creating New Users and Sending Requests to a Webhook Request Bin

1.1 Set Up an AWS EC2 Virtual Machine

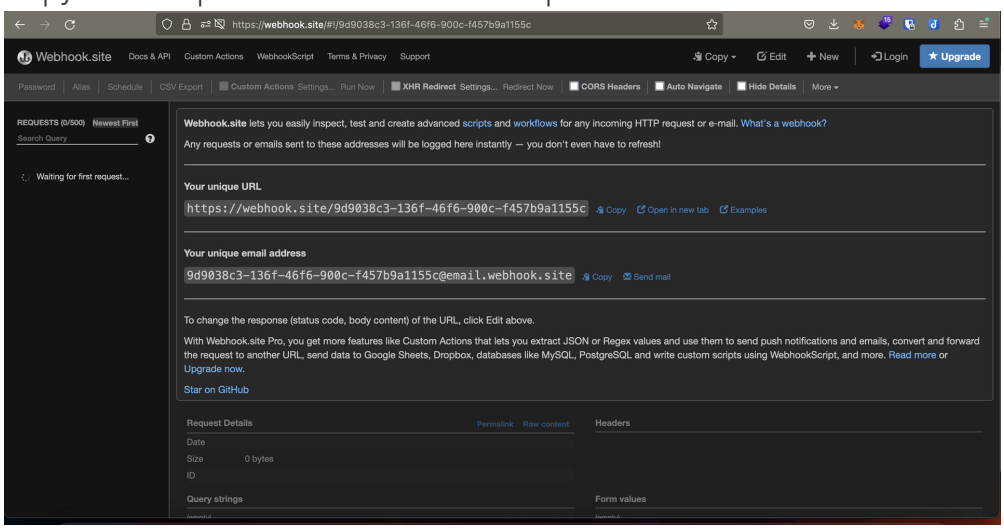
1. Launch a web browser and navigate to the AWS webpage.
2. Log in and proceed to the AWS EC2 Console.



3. Click the "Launch Instances" button and populate the form fields with the provided specifications:
    - Name: botika-devops-intern
    - OS Image: Ubuntu
    - Allow SSH traffic: Yes
    - Allow HTTPS traffic: Yes
    - Allow HTTP traffic: Yes
    - Key Pair Type: RSA (.pem file)
    - Key Pair Name: botika-devops-intern-keypair
4. Click the "Launch Instance" button to create the virtual machine.

1.2 Create a Webhook Request Bin

1. Visit https://webhook.site. The website will automatically generate a request bin for you.
2. Copy the URL provided in the "Your unique URL" section for later use.



1.3 SSH to the VM

1. Navigate to the `.pem` file you downloaded during the EC2 creation and adjust its permissions.

```
chmod 400 botika-devops-intern-keypair.pem
```

2. Initiate an SSH connection to the EC2 instance.

```
ssh -i "botika-devops-intern-keypair.pem" ubuntu@ec2-54-169-8-226.ap-southeast-1.compute.amazonaws.com
```

1.4 Create an sh Script to Create Users and Send Messages

1. Open a terminal window.
2. Navigate to your base directory and create a new folder for this task.

```
cd ~
mkdir soal1
cd soal1
```

3. Create an `.sh` file for user creation and message sending.

```
nano createusers.sh
```

Copy and paste the provided content. Replace `<YOUR WEBHOOK URL>` with the actual URL copied from the Webhook Request Bin step.

```
#!/bin/bash

# Loop to create 200 users
for i in {1..200}; do
    # Define username using loop index
    username="user$i"

    # Calculate UID using loop index
    uid=$((1500 + i - 1))

    # Generate password using UID and username
    password="$uid$username"

    # Add new user with specified UID, home directory, and hashed
password
    sudo useradd -m -u $uid -p $(openssl passwd -1 $password) $username

    # Prepare and send POST request to webhook URL
    curl -X POST -H 'Content-Type: application/json' -H "Authorization:
Basic $(echo -n $username:$password | base64)" -d "
{\"username\":\"$username\",\"password\":\"$password\"}"
https://webhook.site/c3c68791-7f9d-4de8-82df-2f01b897ba24

    # Print completion message for the current user iteration
```

```
    echo "completed $i"
done
```

4. Execute the `.sh` file to create users and send messages.

```
bash createusers.sh
```

1.5 Verify User Creations via the `/etc/passwd` File

1. Create an `.sh` file to validate user creations.

```
nano checkusers.sh
```

Copy and paste the provided content.

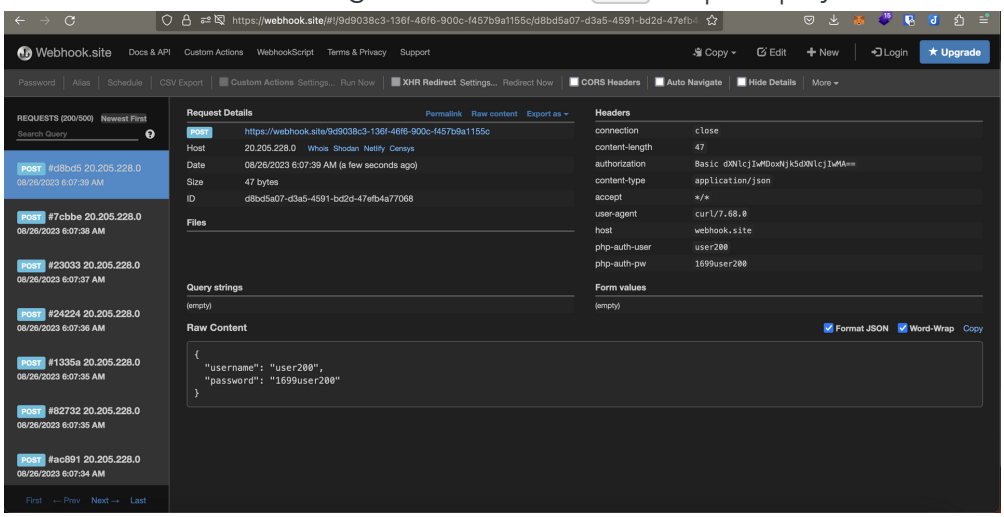
```
#!/bin/bash

# Loop through user numbers from 1 to 200
for i in {1..200}; do
    # Generate username based on loop index
    username="user$i"

    # Check if the user exists in the /etc/passwd file
    grep "^$username:" /etc/passwd
done
```

2. Execute the `.sh` file to check whether the users exist in the `/etc/passwd` file.

```
bash checkusers.sh
```

1.6 Examine the Webhook Request Bin

1. Visit the Webhook Request Bin URL you copied earlier.
2. You should see the messages sent from the `.sh` script displayed on the webpage.



2. Creating a Go Application and Nginx Web Server

2.1 Install Go

1. Install Go:

```
sudo apt update
sudo apt install golang
```

2.2 Install Docker

1. Update the package repository and install necessary packages:

```
sudo apt-get update
sudo apt-get install ca-certificates curl gnupg
sudo install -m 0755 -d /etc/apt/keyrings
```

2. Download and install the Docker GPG key:

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg
```

3. Add the Docker repository to the package sources:

```
echo "deb [arch=$(dpkg --print-architecture) signed-
by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu
$(. /etc/os-release && echo "$VERSION_CODENAME") stable" | sudo tee
/etc/apt/sources.list.d/docker.list > /dev/null
```

4. Update the package repository again:

```
sudo apt-get update
```

5. Install Docker Engine:

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-
plugin docker-compose-plugin
```

2.3 Create a Go Project

1. Navigate to your base directory and create a new directory for this task.

```
cd ~
mkdir soal2
cd soal2
```

2. Create a subdirectory for the Go project and navigate into it.

```
mkdir go
cd go
```

3. Create a file named `main.go` and open it for editing.

```
nano main.go
```

Copy and paste the provided Go code into `main.go`.

```go
package main

import (
    "encoding/json"
    "net/http"
    "os"
    "time"
)

// Response represents the JSON response structure.
type Response struct {
    Hostname string    `json:"hostname"`
    Time     time.Time `json:"time"`
    Message  string    `json:"message"`
}

// handler handles incoming HTTP requests.
func handler(w http.ResponseWriter, r *http.Request) {
    // Get the hostname of the server
    hostname, _ := os.Hostname()

    // Determine the welcome message based on the request's host
    var message string
    if r.Host == "domain.tbd" {
        message = "Welcome to the domain!"
    } else {
        message = "Welcome to the IP address!"
    }

    // Create a Response instance with hostname, current time, and
message
    response := Response{
        Hostname: hostname,
        Time:     time.Now(),
        Message:  message,
    }

    // Marshal the Response to JSON
```

```go
    jsonResponse, _ := json.Marshal(response)

    // Set response headers and write the JSON response
    w.Header().Set("Content-Type", "application/json")
    w.WriteHeader(http.StatusOK)
    w.Write(jsonResponse)
}

func main() {
    // Handle the root path ("/") with the handler function
    http.Handle("/", http.HandlerFunc(handler))

    // Listen and serve on port 8080
    http.ListenAndServe(":8080", nil)
}
```

4. Initialize the Go project and manage dependencies.

```
go mod init go-http-server
go mod tidy
```

5. Create a Dockerfile for the Go project.

```
nano Dockerfile
```

Copy and paste the provided Dockerfile content into the file.

```dockerfile
# Use the latest official Golang image as the base image
FROM golang:latest

# Set the working directory inside the container to /app
WORKDIR /app

# Copy the current directory contents into the container at /app
COPY . .

# Download the dependencies specified in the go.mod file
RUN go mod download

# Build the Go application and generate an executable named 'main'
RUN go build -o main .

# Set the command to run when the container starts
CMD ["./main"]
```

6. Build a Docker image for the Go project.

```
sudo docker build -t go-http-server .
```

2.4 Create the Nginx Setup

1. Create a directory for the Nginx configuration and SSL certificates.

```
cd ~/soal2
mkdir nginx
cd nginx
```

2. Generate an SSL certificate for the `domain.tbd` domain.

```
mkdir ssl
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
ssl/server.key -out ssl/server.crt
```

3. Create an Nginx configuration file named `nginx.conf`.

```
nano nginx.conf
```

Copy and paste the provided Nginx configuration content into the file.

```
# Define worker connections for events
events {
    worker_connections 1024;
}

# Main HTTP configuration
http {
    # Configure rate limiting using the binary remote address
    limit_req_zone $binary_remote_addr zone=one:10m rate=1r/s;

    # Define upstream servers for load balancing
    upstream backend_servers {
        server go-http-server1:8080;
        server go-http-server2:8080;
    }

    # Server block for HTTP to HTTPS redirection
    server {
        listen 80 default_server;
        server_name _;

        # Redirect HTTP requests to HTTPS
        return 301 https://$host$request_uri;
    }
```

```
    # Server block for HTTPS configuration
    server {
        listen 443 ssl;
        server_name _;

        # SSL certificate and key paths
        ssl_certificate /etc/nginx/ssl/server.crt;
        ssl_certificate_key /etc/nginx/ssl/server.key;

        # Location block for handling requests
        location / {
            # Apply rate limiting
            limit_req zone=one burst=5 nodelay;

            # Proxy requests to backend servers
            proxy_pass http://backend_servers;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
        }

        # Location block to deny access to hidden files
        location ~ /\. {
            deny all;
        }
    }
}
```

4. Create a Dockerfile for the Nginx setup.

```
nano Dockerfile
```

Copy and paste the provided Dockerfile content into the file.

```
# Use the official nginx image as the base image
FROM nginx:latest

# Copy the custom nginx configuration file to the appropriate location
COPY nginx.conf /etc/nginx/nginx.conf

# Copy the SSL certificates and keys to the nginx SSL directory
COPY ssl /etc/nginx/ssl
```

5. Build a Docker image for the Nginx setup.

```
sudo docker build -t my-nginx-image .
```

2.5 Run Go Apps and Nginx

1. Create a Docker network.

```
sudo docker network create app
```

2. Run the Go app containers using the created network.

```
sudo docker run -d --name go-http-server1 -p 8081:8080 --network app go-http-server
sudo docker run -d --name go-http-server2 -p 8082:8080 --network app go-http-server
```

3. Run the Nginx container using the same network.

```
sudo docker run -d -p 80:80 -p 443:443 --name my-nginx-container --network app my-nginx-image
```

2.6 Test the App

1. On your local machine, modify the `/etc/hosts` file to redirect the `domain.tbd` domain to localhost.

```
sudo nano /etc/hosts
```

Add the line (change the IP to your EC2 public IP): `54.169.8.226 domain.tbd`.

2. Test the app using its domain in HTTPS mode.

```
curl -k https://domain.tbd
```

3. Test the app using its IP address in HTTPS mode.

```
curl -k https://54.169.8.226
```

4. Test the app using its domain in HTTP mode.

```
curl -k http://domain.tbd
```

5. Test the app using its IP address in HTTP mode.

```
curl -k http://54.169.8.226
```

```
● @lareza-farhan-wanaghi → /workspaces/codespaces-blank $ curl -k http://54.169.8.226
  <html>
  <head><title>301 Moved Permanently</title></head>
  <body>
  <center><h1>301 Moved Permanently</h1></center>
  <hr><center>nginx/1.25.2</center>
  </body>
  </html>
○ @lareza-farhan-wanaghi → /workspaces/codespaces-blank $
```