AccuKnox QA Engineer Practical Assessment – Sindhu Mattegunta

Problem Statement 1:

Project Repository: https://github.com/Vengatesh-m/qa-test

Question

Kubernetes Deployment:

- Deploy the services given in the above-mentioned repository to a local Kubernetes cluster (e.g., Minikube or Kind).

Verification:

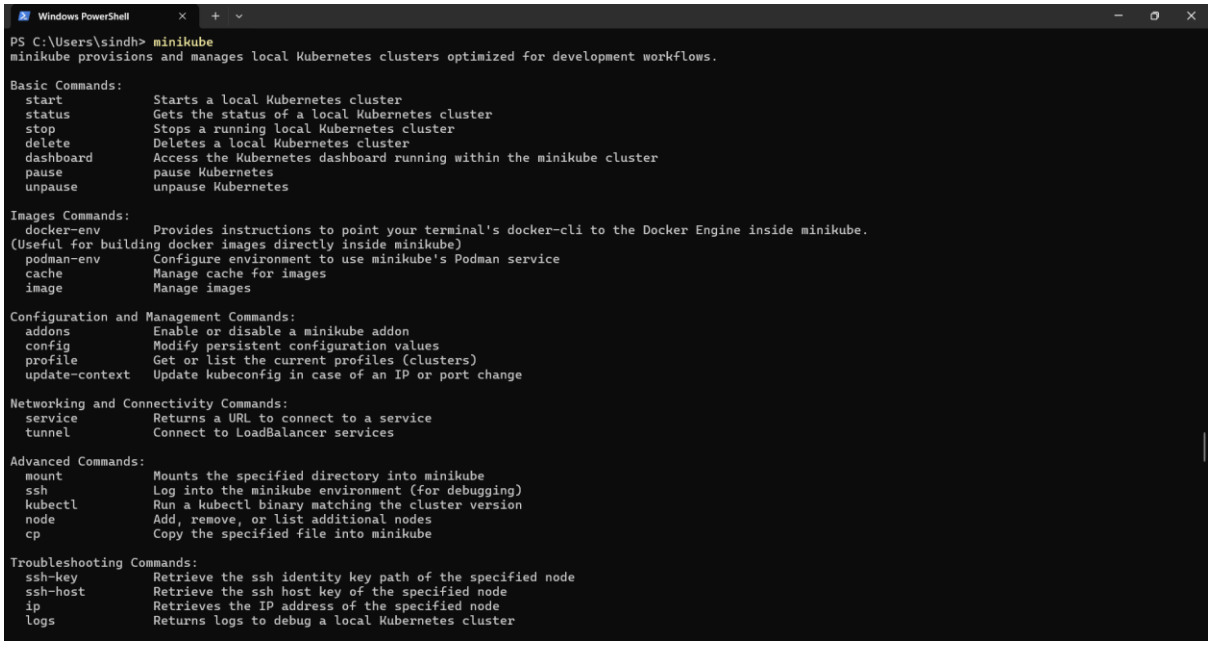
- Ensure the frontend service can successfully communicate with the backend service.
- Verify that accessing the frontend URL displays the greeting message fetched from the backend.

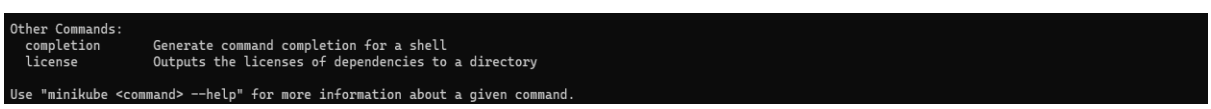
Setup, Execution Instructions and Output

I have deployed your frontend and backend services and built and pushed the Docker images to Docker Hub. Now, frontend service is accessible at http://127.0.0.1:49214 due to the Minikube tunnel.

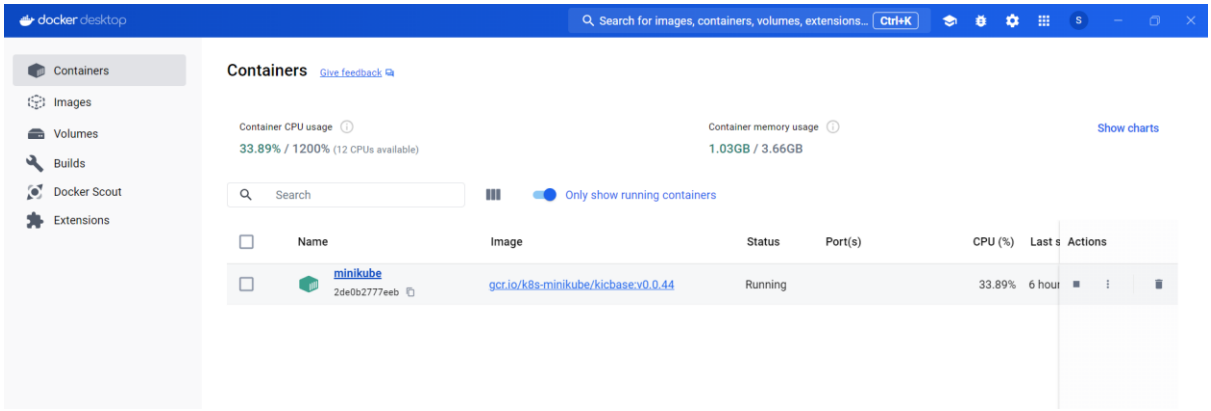
Setup and Execution Instructions

Step 1: Minikube is running





Docker also running



Step 2: Start Minikube

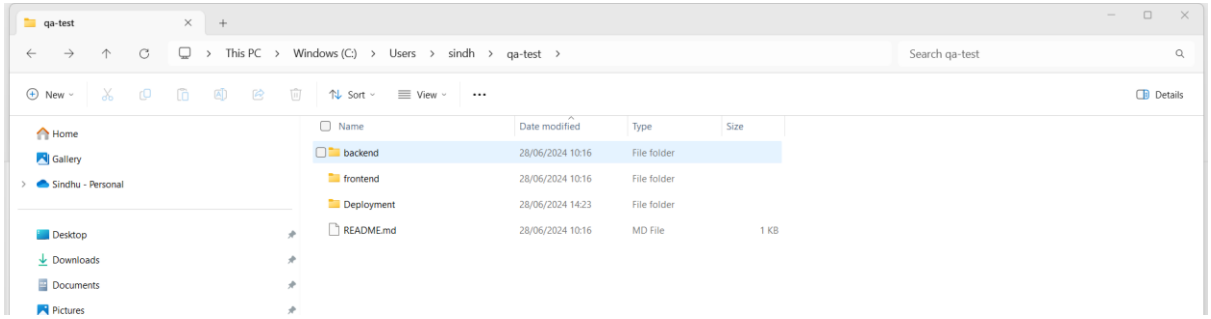
```
PS C:\Users\sindh> minikube start
* minikube v1.33.1 on Microsoft Windows 11 Home 10.0.22631.3737 Build 22631.3737
* Using the docker driver based on existing profile
* Starting "minikube" primary control-plane node in "minikube" cluster
* Pulling base image v0.0.44 ...
* Updating the running docker "minikube" container ...
* Preparing Kubernetes v1.30.0 on Docker 26.1.1 ...
* Verifying Kubernetes components...
  - Using image gcr.io/k8s-minikube/storage-provisioner:v5
* Enabled addons: storage-provisioner, default-storageclass
* Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
PS C:\Users\sindh> git clone https://github.com/Vengatesh-m/qa-test
fatal: destination path 'qa-test' already exists and is not an empty directory.
```

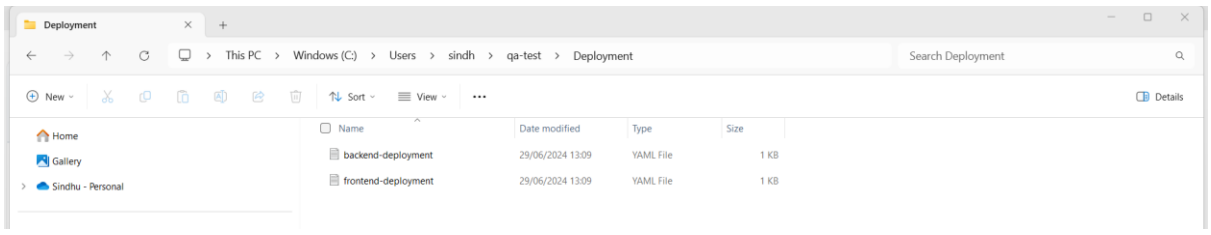
Step 3: Clone the Repository

```
PS C:\Users\sindh> git clone https://github.com/Vengatesh-m/qa-test
```

```
PS C:\Users\sindh> cd qa-test
PS C:\Users\sindh\qa-test> ls


    Directory: C:\Users\sindh\qa-test


Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
d-----        28/06/2024     10:16                backend
d-----        28/06/2024     14:23                Deployment
d-----        28/06/2024     10:16                frontend
-a----        28/06/2024     10:16            852 README.md
```

Step 4: Services cloned in Directory





Step 5: Deployed the Backend and Frontend to Kubernetes:

```
PS C:\Users\sindh\qa-test> cd C:\Users\sindh\qa-test\Deployment
PS C:\Users\sindh\qa-test\Deployment> kubectl apply -f backend-deployment.yaml
deployment.apps/backend-deployment configured
service/backend-service unchanged
PS C:\Users\sindh\qa-test\Deployment> kubectl apply -f frontend-deployment.yaml
deployment.apps/frontend-deployment configured
service/frontend-service unchanged
```

Step 6: Verified Deployments and Services:

```
PS C:\Users\sindh\qa-test\Deployment> kubectl get pods
NAME                                  READY   STATUS    RESTARTS      AGE
backend-deployment-64458c89d4-5w4l8   1/1     Running   1 (5m4s ago)   69m
backend-deployment-64458c89d4-hvblh   1/1     Running   1 (5m4s ago)   70m
frontend-deployment-846c98b6b5-8l5g4  1/1     Running   1 (5m4s ago)   70m
PS C:\Users\sindh\qa-test\Deployment> kubectl get services
NAME               TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)        AGE
backend-service    ClusterIP      10.110.236.197   <none>        3000/TCP       23h
frontend-service   LoadBalancer   10.104.64.174    <pending>     80:30097/TCP   23h
kubernetes         ClusterIP      10.96.0.1        <none>        443/TCP        45h
```

Step 7: Build Docker Images:

```
PS C:\Users\sindh\qa-test> docker images
REPOSITORY                          TAG       IMAGE ID       CREATED        SIZE
sindhu1989/backend                  latest    a1bcc0e804ba   4 hours ago    916MB
your-dockerhub-username/backend     latest    a1bcc0e804ba   4 hours ago    916MB
sindhu1989/frontend                 latest    7111ff5015c4   4 hours ago    917MB
your-dockerhub-username/frontend    latest    7111ff5015c4   4 hours ago    917MB
gcr.io/k8s-minikube/kicbase         v0.0.44   5a6e59a9bdc0   7 weeks ago    1.26GB
docker/welcome-to-docker            latest    c1f619b6477e   7 months ago   18.6MB
```

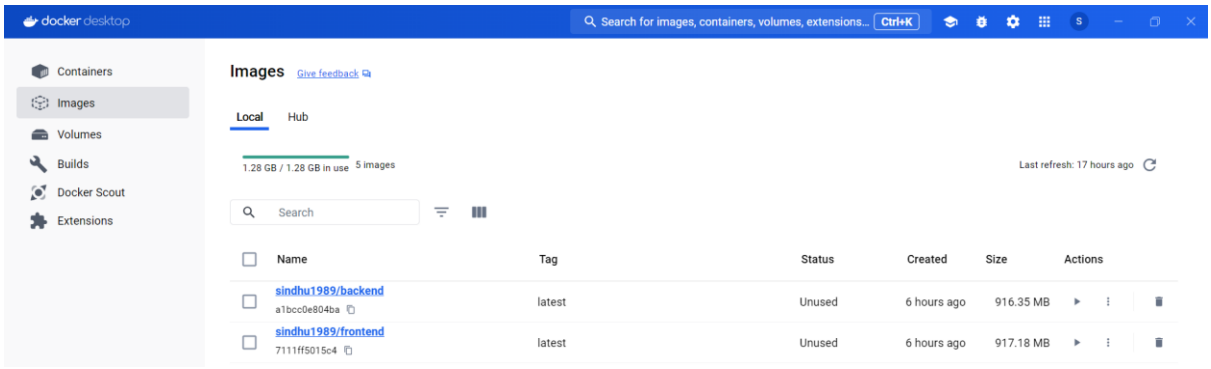
Step 8: Frontend Docker images

```
PS C:\Users\sindh\qa-test\frontend> docker build -t sindhu1989/frontend:latest .
[+] Building 3.0s (10/10) FINISHED                                                                                  docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                0.1s
 => => transferring dockerfile: 447B                                                                                                0.1s
 => [internal] load metadata for docker.io/library/node:14                                                                          2.6s
 => [auth] library/node:pull token for registry-1.docker.io                                                                         0.0s
 => [internal] load .dockerignore                                                                                                   0.0s
 => => transferring context: 2B                                                                                                     0.0s
 => [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa                    0.0s
 => [internal] load build context                                                                                                   0.1s
 => => transferring context: 1.48kB                                                                                                 0.1s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                                               0.0s
 => CACHED [3/4] COPY . .                                                                                                           0.0s
 => CACHED [4/4] RUN npm install                                                                                                    0.0s
 => exporting to image                                                                                                              0.0s
 => => exporting layers                                                                                                             0.0s
 => => writing image sha256:7111ff5015c446aa214a2cc727af1b661bbfc986858055a4693c774e2ea8e5eb                                        0.0s
 => => naming to docker.io/sindhu1989/frontend:latest                                                                               0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/rwm8w3dvsf64wy389f7g4fx31

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

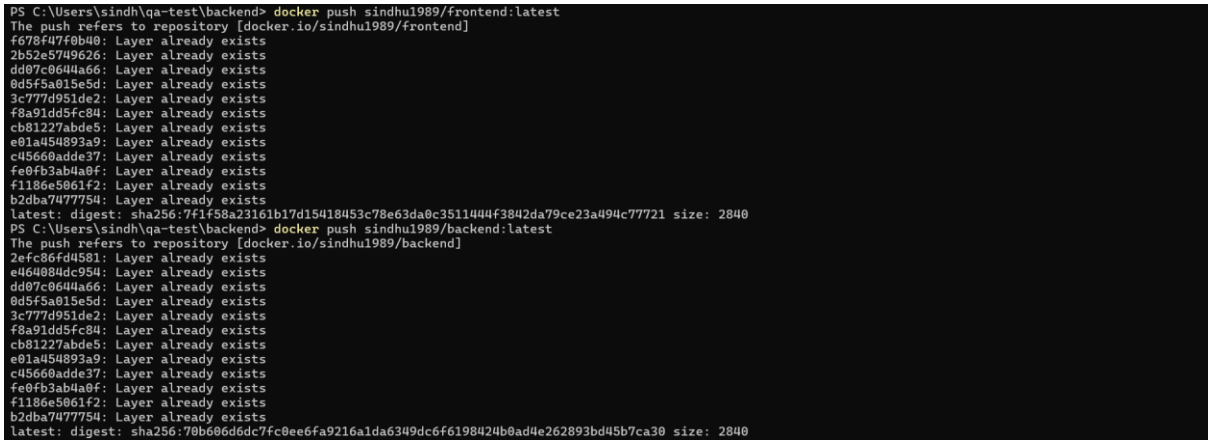
Step 9: Backend Docker images

```
PS C:\Users\sindh\qa-test\frontend>  cd C:\Users\sindh\qa-test\backend
PS C:\Users\sindh\qa-test\backend> docker build -t sindhu1989/backend:latest .
[+] Building 1.2s (9/9) FINISHED                                                                                    docker:desktop-linux
 => [internal] load build definition from Dockerfile                                                                                0.0s
 => => transferring dockerfile: 449B                                                                                                0.0s
 => [internal] load metadata for docker.io/library/node:14                                                                          1.0s
 => [internal] load .dockerignore                                                                                                   0.0s
 => => transferring context: 2B                                                                                                     0.0s
 => [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa                    0.0s
 => [internal] load build context                                                                                                   0.0s
 => => transferring context: 1.05kB                                                                                                 0.0s
 => CACHED [2/4] WORKDIR /usr/src/app                                                                                               0.0s
 => CACHED [3/4] COPY . .                                                                                                           0.0s
 => CACHED [4/4] RUN npm install                                                                                                    0.0s
 => exporting to image                                                                                                              0.0s
 => => exporting layers                                                                                                             0.0s
 => => writing image sha256:a1bcc0e804ba40a620f762403eec424066b902bcb3dea7d4a162e005809aff37                                        0.0s
 => => naming to docker.io/sindhu1989/backend:latest                                                                                0.0s

View build details: docker-desktop://dashboard/build/desktop-linux/desktop-linux/xjg418qdxgruptd14yh5052kj

What's next:
    View a summary of image vulnerabilities and recommendations → docker scout quickview
```

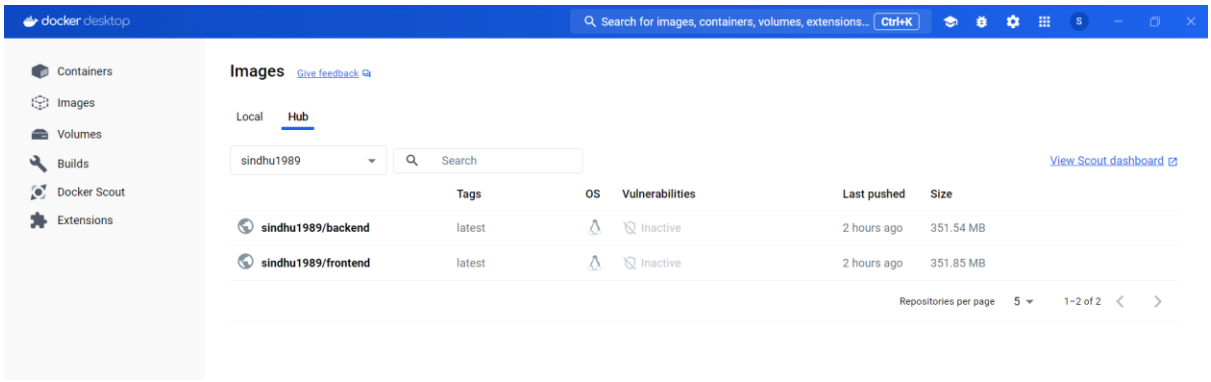
Images created in Docker



Step 10: Pushed Docker Images to Docker Hub



Can see images pushed to docker hub



Step 11: Access the Frontend Service

Final Output

The frontend service will be accessible at http://127.0.0.1:49214.

You can see the output with the URL http://127.0.0.1:49214

Note: The URL http://192.168.49.2:30097 isn't working because, in Minikube, LoadBalancer services don't automatically get an external IP like they do on cloud platforms. LoadBalancer services often have issues in Minikube and port forwarding can be used to access the service on local machine.

Summary

Here's a summary of the main commands:

- minikube start
- git clone https://github.com/Vengatesh-m/qa-test
- cd qa-test
- cd Deployment
- kubectl apply -f backend-deployment.yaml
- kubectl apply -f frontend-deployment.yaml
- kubectl get pods
- kubectl get services
- cd ../frontend
- docker build -t sindhu1989/frontend:latest .
- cd ../backend
- docker build -t sindhu1989/backend:latest .
- docker push sindhu1989/frontend:latest
- docker push sindhu1989/backend:latest
- minikube service frontend-service

Question

Automated Testing:

- Write a simple test script (using a tool of your choice) to verify the integration between the frontend and backend services.
- The test should check that the frontend correctly displays the message returned by the backend.

Test Script on Bash

```
# Save the following content into a file named test_integration.sh

#!/bin/bash

# URL of the frontend service
FRONTEND_URL="http://127.0.0.1:49214"

# Fetch the frontend page
RESPONSE=$(curl -s $FRONTEND_URL)

# Expected message
EXPECTED_MESSAGE="Hello from the backend!"

# Check if the response contains the expected message
if echo "$RESPONSE" | grep -q "$EXPECTED_MESSAGE"; then
  echo "Test Passed: Frontend displays the correct message from the backend."
  exit 0
else
  echo "Test Failed: Frontend does not display the correct message from the backend."
  exit 1
fi
```

Steps to Execute:

1. **Create the Script:**

   Create a file named test_integration.sh
   nano test_integration.sh

2. **Make the Script Executable:**

   chmod +x test_integration.sh

3. **Run the Test Script:**

   ./test_integration.sh

Problem Statement 2:

1. System Health Monitoring Script: I have attempted to achieve with Bash
Develop a script that monitors the health of a Linux system. It should check CPU usage, memory usage, disk space, and running processes. If any of these metrics exceed predefined thresholds (e.g., CPU usage > 80%), the script should send an alert to the console or a log file.

System Health Monitoring Script

This script will monitor CPU usage, memory usage, disk space, and running processes. If any of these metrics exceed predefined thresholds, it will send an alert to the console.

Test Script in Bash

```
#!/bin/bash

# Thresholds
CPU_THRESHOLD=80
MEMORY_THRESHOLD=80
DISK_THRESHOLD=90

# Log file
LOG_FILE="/var/log/system_health.log"

# Function to check CPU usage
check_cpu_usage() {
  CPU_USAGE=$(top -bn1 | grep "Cpu(s)" | \
           sed "s/.*, *\([0-9.]*\)%* id.*/\1/" | \
           awk '{print 100 - $1}')
  echo "CPU Usage: $CPU_USAGE%"
  if (( ${CPU_USAGE%.*} > CPU_THRESHOLD )); then
    echo "ALERT: CPU usage is above $CPU_THRESHOLD%" | tee -a $LOG_FILE
  fi
}

# Function to check memory usage
check_memory_usage() {
  MEMORY_USAGE=$(free | grep Mem | awk '{print $3/$2 * 100.0}')
  echo "Memory Usage: $MEMORY_USAGE%"
  if (( ${MEMORY_USAGE%.*} > MEMORY_THRESHOLD )); then
    echo "ALERT: Memory usage is above $MEMORY_THRESHOLD%" | tee -a
$LOG_FILE
  fi
}
```

```
# Function to check disk space usage
check_disk_space() {
  DISK_USAGE=$(df -h / | grep / | awk '{ print $5 }' | sed 's/%//g')
  echo "Disk Usage: $DISK_USAGE%"
  if (( DISK_USAGE > DISK_THRESHOLD )); then
    echo "ALERT: Disk usage is above $DISK_THRESHOLD%" | tee -a $LOG_FILE
  fi
}

# Function to check running processes
check_running_processes() {
  RUNNING_PROCESSES=$(ps aux --no-heading | wc -l)
  echo "Running Processes: $RUNNING_PROCESSES"
}
```

Instructions to Use the Scripts

1. **System Health Monitoring Script**
   - Save the script to a file, for example, system_health.sh.
   - Make the script executable:

     chmod +x system_health.sh
   - Run the script:

     ./system_health.sh

2. Automated Backup Solution:
Write a script to automate the backup of a specified directory to a remote server or a cloud storage solution. The script should provide a report on the success or failure of the backup operation.

Test Script in Bash

```
#!/bin/bash

# Variables
SOURCE_DIR="/path/to/source/directory"
REMOTE_USER="remote-user"
REMOTE_HOST="remote-host"
REMOTE_DIR="/path/to/remote/directory"
LOG_FILE="/var/log/backup.log"

# Function to perform the backup
perform_backup() {
```

```
  rsync -avz --delete $SOURCE_DIR
$REMOTE_USER@$REMOTE_HOST:$REMOTE_DIR
  if [[ $? -eq 0 ]]; then
    echo "$(date) - Backup successful" | tee -a $LOG_FILE
  else
    echo "$(date) - Backup failed" | tee -a $LOG_FILE
  fi
}
```

Instructions to Use the Scripts

1. **Automated Backup Solution**
    - Save the script to a file, for example, backup.sh.

- `/path/to/source/directory` should be the local directory you want to back up.
- `remote-user` should be the username for the remote server.
- `remote-host` should be the address of the remote server.
- `/path/to/remote/directory` should be the destination directory on the remote server.
- `/var/log/backup.log` should be the path where you want to store the log file.

    - Make the script executable:

      chmod +x backup.sh

    - Run the script:

      ./backup.sh