

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

```
PS C:\Users\sindh> minikube
minikube provisions and manages local Kubernetes clusters optimized for development workflows.

Basic Commands:
start           Starts a local Kubernetes cluster
status          Gets the status of a local Kubernetes cluster
stop            Stops a running local Kubernetes cluster
delete          Deletes a local Kubernetes cluster
dashboard       Access the Kubernetes dashboard running within the minikube cluster
pause           pause Kubernetes
unpause         unpause Kubernetes

Images Commands:
docker-env      Provides instructions to point your terminal's docker-cli to the Docker Engine inside minikube.
(Useful for building docker images directly inside minikube)
podman-env      Configure environment to use minikube's Podman service
cache           Manage cache for images
image           Manage images

Configuration and Management Commands:
addons          Enable or disable a minikube addon
config          Modify persistent configuration values
profile         Get or list the current profiles (clusters)
update-context  Update kubeconfig in case of an IP or port change

Networking and Connectivity Commands:
service         Returns a URL to connect to a service
tunnel          Connect to LoadBalancer services

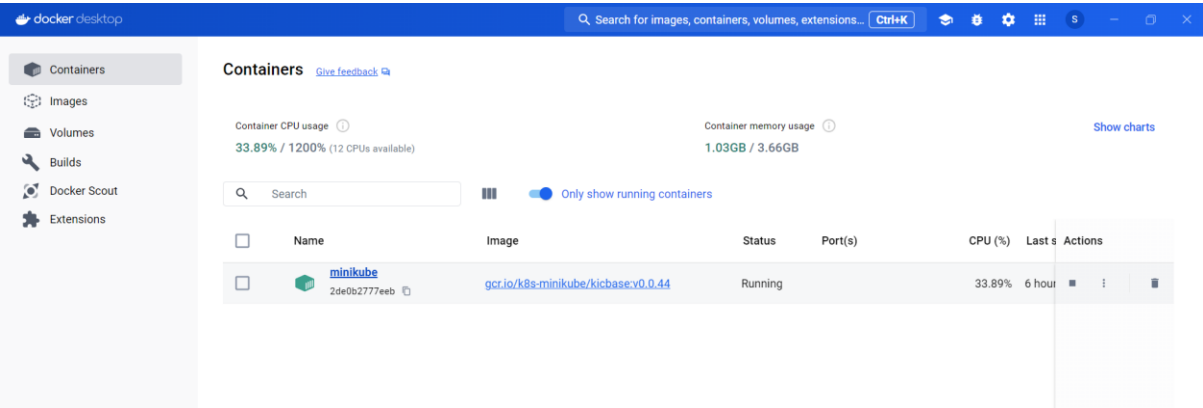
Advanced Commands:
mount           Mounts the specified directory into minikube
ssh             Log into the minikube environment (for debugging)
kubectl         Run a kubectl binary matching the cluster version
node            Add, remove, or list additional nodes
cp             Copy the specified file into minikube

Troubleshooting Commands:
ssh-key        Retrieve the ssh identity key path of the specified node
ssh-host       Retrieve the ssh host key of the specified node
ip             Retrieves the IP address of the specified node
logs           Returns logs to debug a local Kubernetes cluster

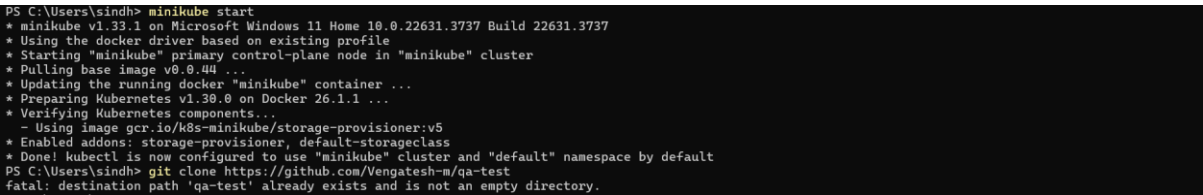
Other Commands:
completion     Generate command completion for a shell
license        Outputs the licenses of dependencies to a directory

Use "minikube <command> --help" for more information about a given command.
```

Docker also running



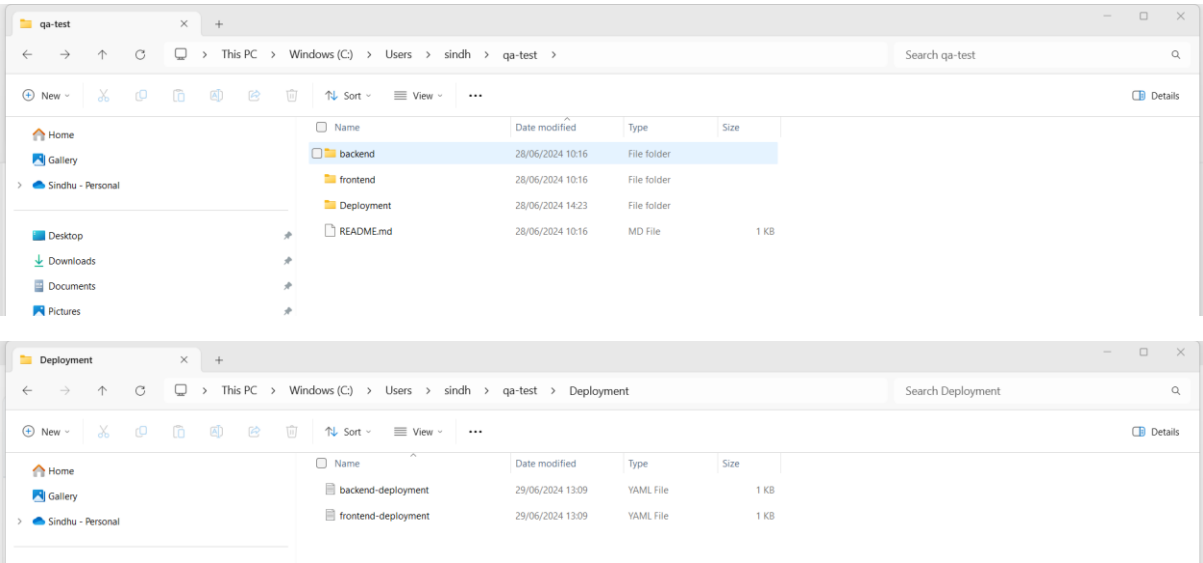
Step 2: Start Minikube



Step 3: Clone the Repository



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-64458c89d4-5m4l8 1/1     Running   1 (5m4s ago)   69m
backend-deployment-64458c89d4-hv6lh 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
sindhul989/backend  latest       a1bcc0e804ba  4 hours ago   916MB
your-dockerhub-username/backend latest       a1bcc0e804ba  4 hours ago   916MB
sindhul989/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 sindhul989/frontend:latest .
[+] Building 3.0s (10/10) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 447B
=> [internal] load metadata for docker.io/library/node:14
=> [auth] library/node:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> [internal] load build context
=> => transferring context: 1.48kB
=> CACHED [2/4] WORKDIR /usr/src/app
=> CACHED [3/4] COPY . .
=> CACHED [4/4] RUN npm install
=> exporting image
=> => exporting layers
=> => writing image sha256:7111ff5015c446aa214a2cc727af1b661bbfc986858055a4693c774e2ea8e5eb
=> => naming to docker.io/sindhul989/frontend:latest
```

View build details: [docker-desktop://dashboard/build/desktop-linux/desktop-linux/rwm8w3dvsf64my389f7g4fx31](#)

What's next:

[View a summary of image vulnerabilities and recommendations](#) → [docker scout quickview](#)

Step 9: Backend Docker images

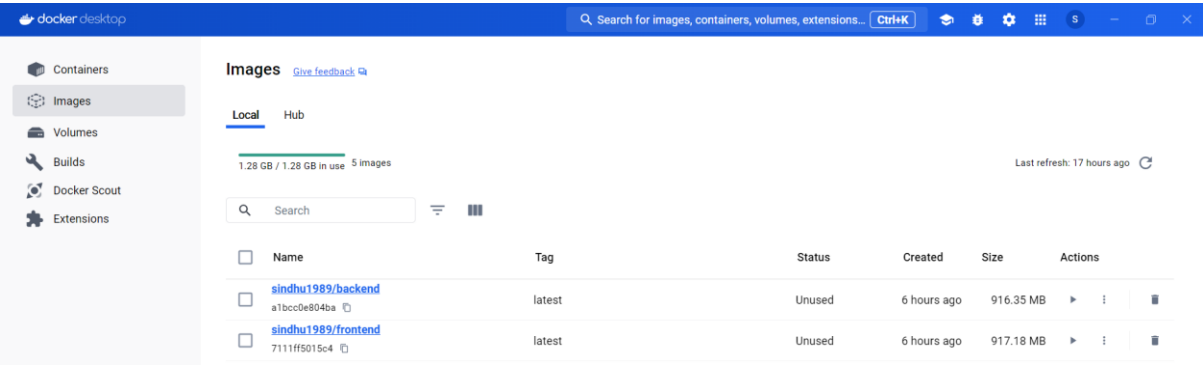
```
PS C:\Users\sindh\qa-test\frontend> cd C:\Users\sindh\qa-test\backend
PS C:\Users\sindh\qa-test\backend> docker build -t sindhul989/backend:latest .
[+] Building 1.2s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 449B
=> [internal] load metadata for docker.io/library/node:14
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/node:14@sha256:a158d3b9b4e3fa813fa6c8c590b8f0a860e015ad4e59bbce5744d2f6fd8461aa
=> [internal] load build context
=> => transferring context: 1.05kB
=> CACHED [2/4] WORKDIR /usr/src/app
=> CACHED [3/4] COPY . .
=> CACHED [4/4] RUN npm install
=> exporting image
=> => exporting layers
=> => writing image sha256:a1bcc0e804ba40a620f762403eec424066b902bcb3dea7d4a162e085809aff37
=> => naming to docker.io/sindhul989/backend:latest
```

View build details: [docker-desktop://dashboard/build/desktop-linux/desktop-linux/xjg4l8qdxgruptd14yh5052kj](#)

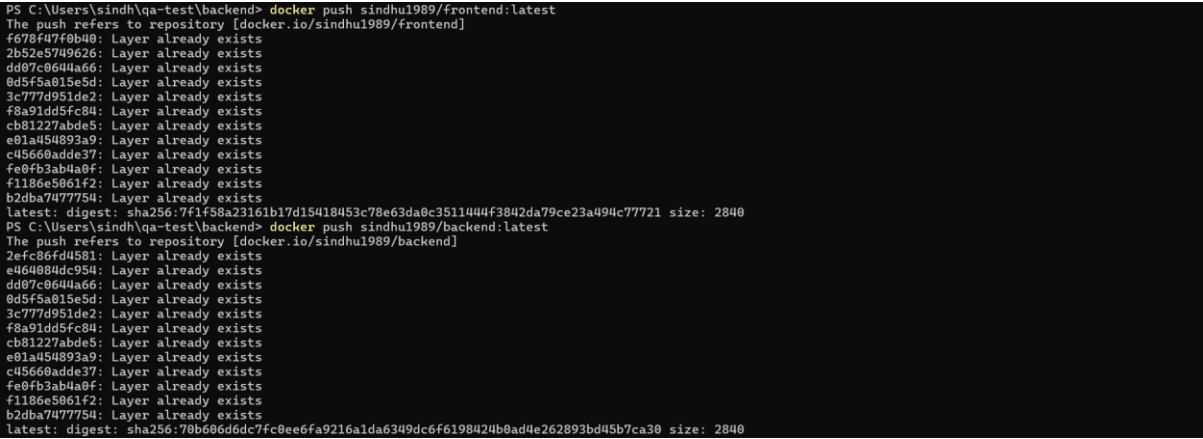
What's next:

[View a summary of image vulnerabilities and recommendations](#) → [docker scout quickview](#)

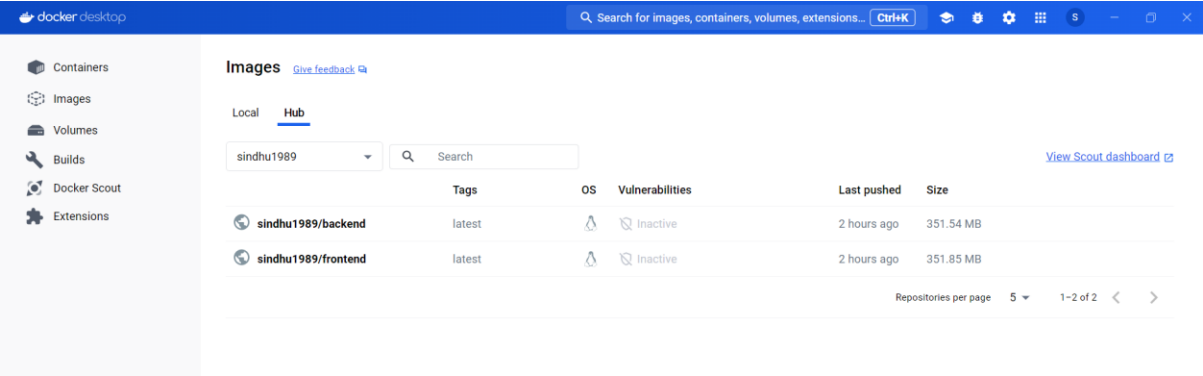
Images created in Docker



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>.

```
PS C:\Users\sindh\qa-test\backend> minikube service frontend-service
```

NAMESPACE	NAME	TARGET PORT	URL
default	frontend-service	80	http://192.168.49.2:30097

```
* Starting tunnel for service frontend-service.
```

NAMESPACE	NAME	TARGET PORT	URL
default	frontend-service		http://127.0.0.1:49214

```
* Opening service default/frontend-service in default browser...  
! Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

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`