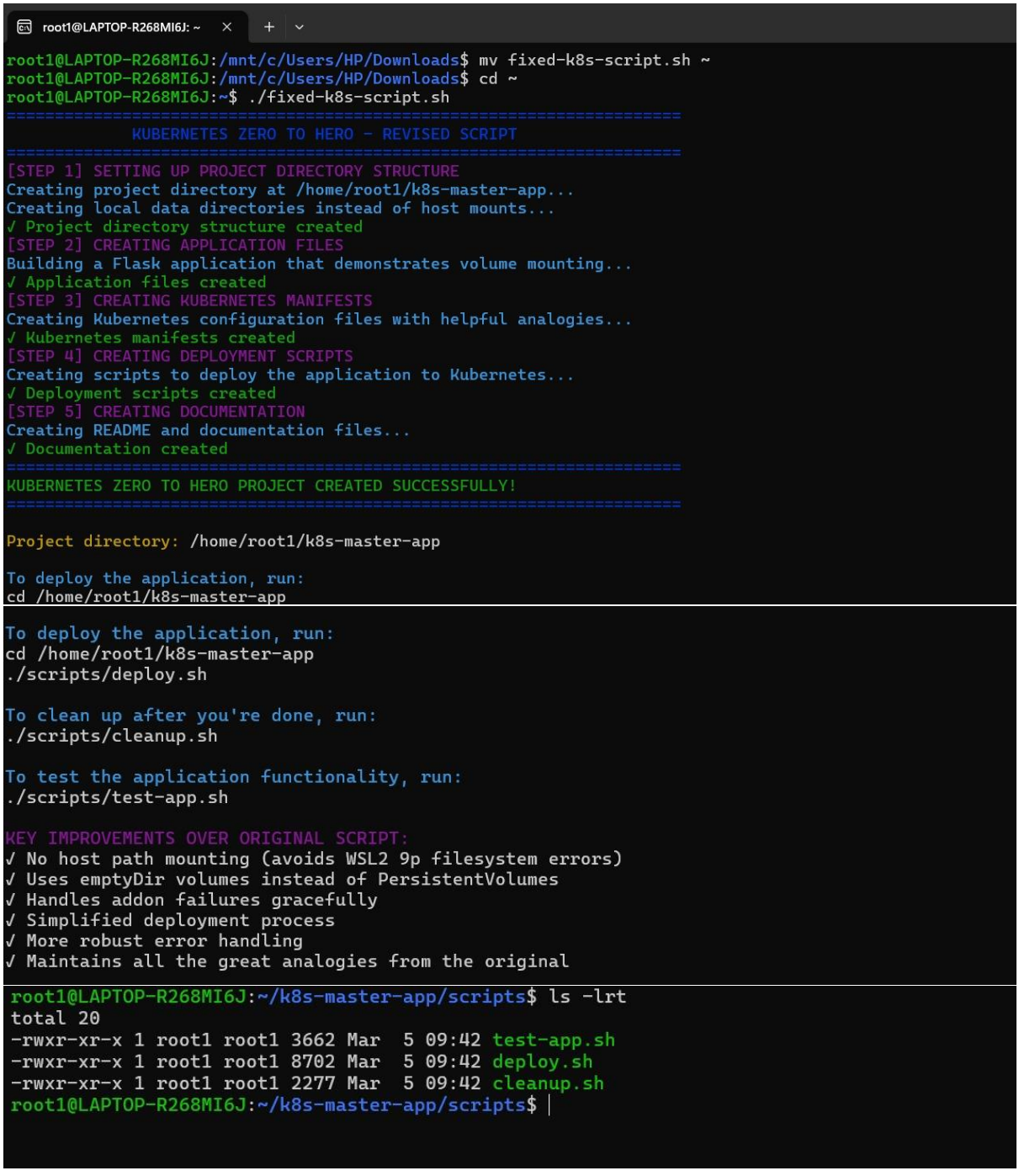
Kubernetes full implementation:

Firstly we would need to reset our minikube and run both the files from root.



And then we need to make some changes in our app.py files related to logs:

✅ Steps to Fix Your Deployment

Follow these steps to resolve your issues:

1.Update `app.py` to Fix Logging Issue

Run the following command to edit the file:

```
nano ~/k8s-master-app/app/app.py
```

This is the existing part we need to change:

```
# Set up logging to print to console and file
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(sys.stdout),
        logging.FileHandler(os.environ.get('LOG_PATH', '/app/app.log'))
    ]
)
```

Then **replace** the existing logging setup with this:

```
# Set up logging to print to console and file
log_file = os.path.join(os.environ.get('LOG_PATH', '/logs'), 'app.log')
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),
        logging.FileHandler(log_file)
    ]
)
```

- **Ensure proper indentation**.
- **Save the file** (`CTRL + X → Y → ENTER`).

---

2. Change Directory to the App Folder

```
cd ~/k8s-master-app/app
```

---

3. Rebuild the Docker Image

```
docker build -t k8s-master-app:latest .
```

This ensures the latest `app.py` changes are included in the container.

---

4. Delete Existing Pods

```
kubectl delete pod -l app=k8s-master -n k8s-demo
```

This will **force Kubernetes to pull the updated container**.

---

5. Move Back to the Project Root

```
cd ..
```

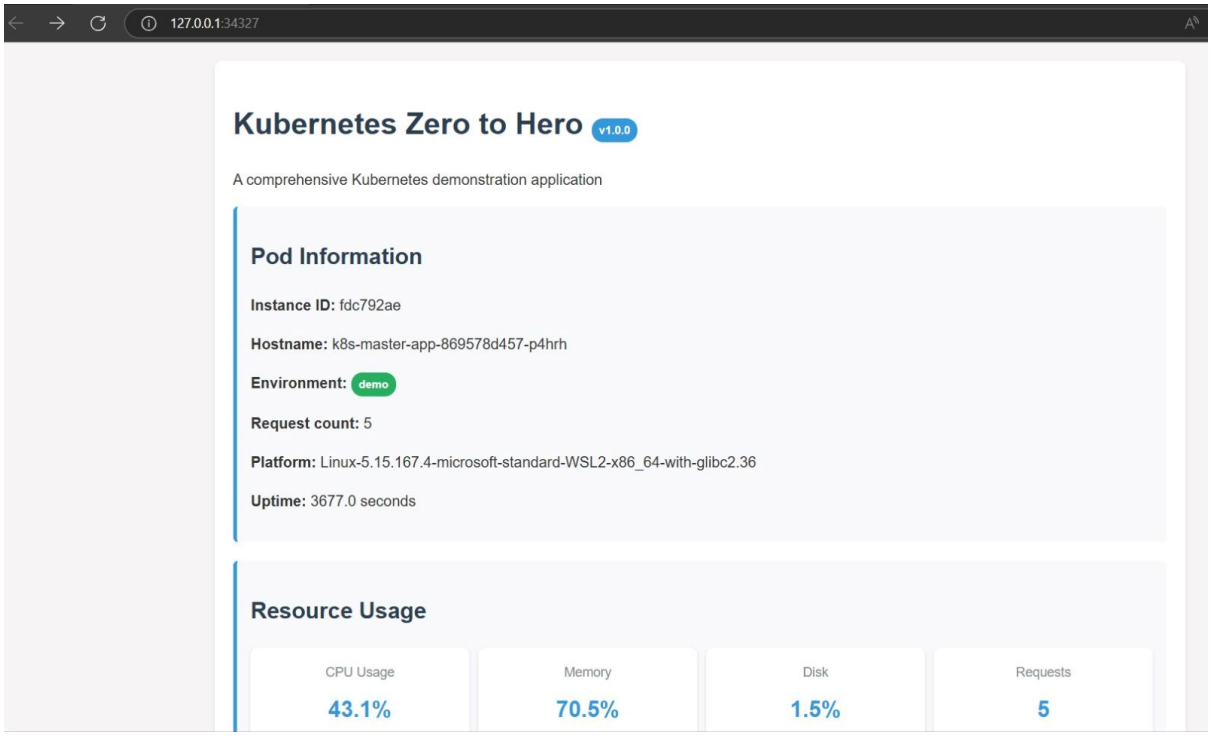
---

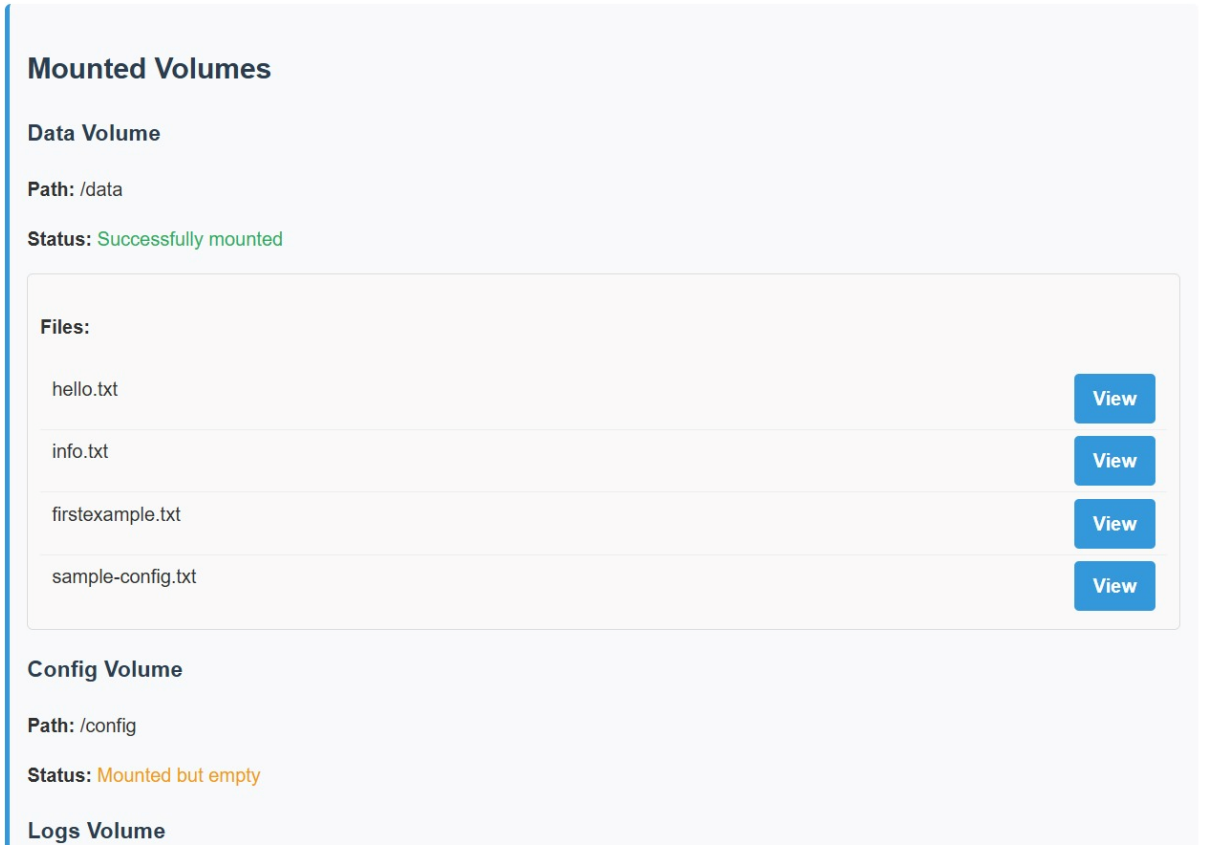
6. Redeploy the Application

```
./scripts/deploy.sh
```

This will redeploy the updated image with the fixed logging setup.

And after this our app would run:





Logs Volume

Path: /logs

Status: Successfully mounted

Files:

init.log	View
app.log	View
container.log	View
file_operations.log	View

Actions

Create a File | API Info | Health Check | Metrics

Flow of the Code:

When you execute deploy.sh, the sequence of execution generally follows these steps:

1. deploy.sh Execution

deploy.sh is typically a shell script that automates the deployment process.

It likely contains commands to build the Docker image, push it to a container registry (if needed), and apply Kubernetes YAML files to deploy the application.

2. Processing Kubernetes YAML Files (.yaml Files)

The .yaml files define the Kubernetes resources (Deployment, Service, ConfigMap, etc.).

The main important files involved:

Deployment YAML: Specifies how many replicas of the pod to run and which Docker image to use.

Service YAML: Exposes the Flask app running inside the pod.

3. Docker File Execution (Dockerfile)

When the Deployment is applied, Kubernetes pulls the Docker image from a registry (or builds it locally if specified).

The Dockerfile is executed to build the container:

Copies main.py and dependencies.

Installs necessary dependencies (like Flask).

Defines the command to start the Flask app (usually python main.py or similar).

4. main.py Execution (Flask Application)

Once the container starts, it runs main.py, which contains your Flask application.

The Flask app listens on a port (e.g., 5000).

If a Service exposes the pod, the Flask app becomes accessible.

Overall Execution Flow

1. Run deploy.sh → Triggers Kubernetes deployment.

2. Apply Kubernetes YAML Files → Creates pods and services.

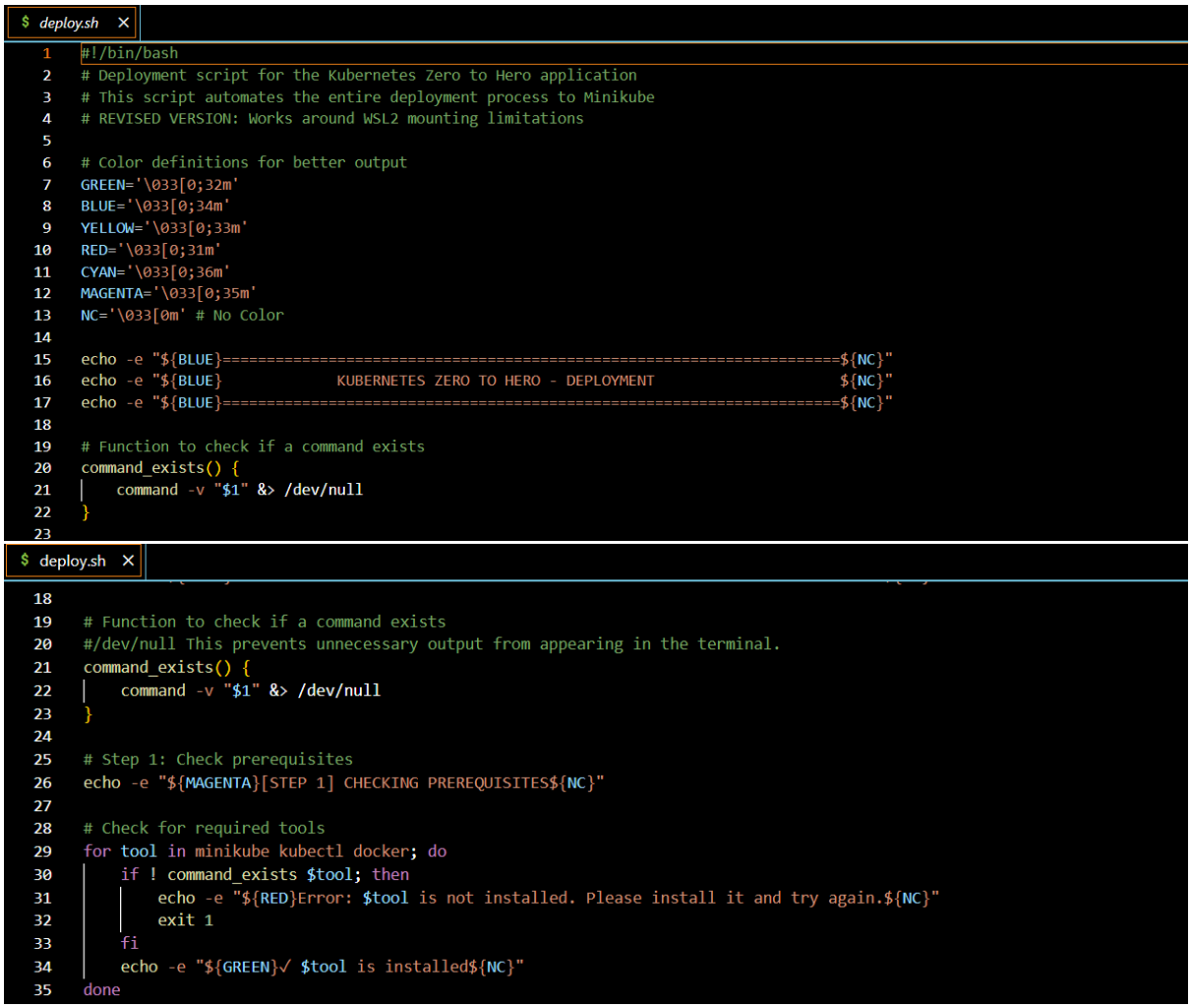
3. Pull/Build Docker Image → Executes Dockerfile to prepare the environment.

4. Run main.py inside the container → Starts the Flask app.

5. Expose Service → Makes the app accessible via a Kubernetes Service (LoadBalancer, NodePort, or ClusterIP).

Lets discuss the deploy.sh:

Firstly we would discuss about various colors for various messages for this we use ansi code's for color and store them in variable to use later and then we are creating a existing_command to check if a tool or dependency is present or not, so that to check within the code:



```
$ deploy.sh
#!/bin/bash
# Deployment script for the Kubernetes Zero to Hero application
# This script automates the entire deployment process to Minikube
# REVISED VERSION: Works around WSL2 mounting limitations

# Color definitions for better output
GREEN='\033[0;32m'
BLUE='\033[0;34m'
YELLOW='\033[0;33m'
RED='\033[0;31m'
CYAN='\033[0;36m'
MAGENTA='\033[0;35m'
NC='\033[0m' # No Color

echo -e "${BLUE}================================================================${NC}"
echo -e "${BLUE}            KUBERNETES ZERO TO HERO - DEPLOYMENT                ${NC}"
echo -e "${BLUE}================================================================${NC}"

# Function to check if a command exists
command_exists() {
    command -v "$1" &> /dev/null
}

$ deploy.sh
# Function to check if a command exists
#/dev/null This prevents unnecessary output from appearing in the terminal.
command_exists() {
    command -v "$1" &> /dev/null
}

# Step 1: Check prerequisites
echo -e "${MAGENTA}[STEP 1] CHECKING PREREQUISITES${NC}"

# Check for required tools
for tool in minikube kubectl docker; do
    if ! command_exists $tool; then
        echo -e "${RED}Error: $tool is not installed. Please install it and try again.${NC}"
        exit 1
    fi
    echo -e "${GREEN}✓ $tool is installed${NC}"
done
```

Output of first step:

```
hafsa_027@Dell:~/k8s-master-app$ ./scripts/deploy.sh
================================================================================
              KUBERNETES ZERO TO HERO - DEPLOYMENT
================================================================================
[STEP 1] CHECKING PREREQUISITES
✓ minikube is installed
✓ kubectl is installed
✓ docker is installed
```

Step2:

```
# Step 2: Ensure Minikube is running
echo -e "${MAGENTA}[STEP 2] ENSURING MINIKUBE IS RUNNING${NC}"

if ! minikube status | grep -q "host: Running"; then
    echo -e "${YELLOW}Minikube is not running. Starting Minikube...${NC}"

    # Start Minikube with appropriate configuration
    minikube start --cpus=2 --memory=4096 --disk-size=20g

    if [ $? -ne 0 ]; then #$? → Stores the exit status of the most recent command
    #0 → The exit code 0 typically means success.
        echo -e "${RED}Failed to start Minikube. Trying with minimal configuration...${NC}"

        # Fallback to minimal configuration
        minikube start --driver=docker #Uses Docker as the virtualization driver , Runs Kubernetes inside a Docker container instead of a virtual machine.

        if [ $? -ne 0 ]; then
            echo -e "${RED}Failed to start Minikube. Exiting.${NC}"
            exit 1
        fi
    fi
else
    echo -e "${GREEN}✓ Minikube is already running${NC}"
fi
```

Output of Step 2:

```
✓ docker is installed
[STEP 2] ENSURING MINIKUBE IS RUNNING
✓ Minikube is already running
```

Step3:

```
# Step 3: Enable required Minikube addons
echo -e "${MAGENTA}[STEP 3] ENABLING MINIKUBE ADDONS${NC}"

# We'll handle addons more carefully, checking if they're already enabled
# and handling errors better

# Function to safely enable an addon
enable_addon() {
    local addon=$1
    local already_enabled=$(minikube addons list | grep $addon | grep -c "enabled")

    if [ $already_enabled -eq 1 ]; then
        echo -e "${GREEN}✓ $addon addon is already enabled${NC}"
        return 0
    fi

    echo -e "${CYAN}Enabling $addon addon...${NC}"
    minikube addons enable $addon

    if [ $? -ne 0 ]; then
        echo -e "${YELLOW}Warning: Failed to enable $addon addon. Continuing without it.${NC}"
        return 1
    else
        echo -e "${GREEN}✓ $addon addon enabled${NC}"
        return 0
    fi
}

# Try to enable addons but don't fail if they don't work
enable_addon "dashboard" || true

echo -e "${YELLOW}Note: Skipping Ingress and Metrics Server addons as they may not work in all environments${NC}"
echo -e "${YELLOW}The application will still work without these addons${NC}"
```

Output of Step 3:

```
[STEP 3] ENABLING MINIKUBE ADDONS
✓ dashboard addon is already enabled
Note: Skipping Ingress and Metrics Server addons as they may not work in all environments
The application will still work without these addons
```

Step4:

```
# Step 4: Configure Docker to use Minikube's Docker daemon
echo -e "${MAGENTA}[STEP 4] CONFIGURING DOCKER TO USE MINIKUBE${NC}"
echo -e "${CYAN}This allows us to build images directly into Minikube's registry${NC}"

#minikube docker-env prints environment variables needed to redirect Docker commands to Minikube's internal Docker daemon.
eval $(minikube docker-env)
if [ $? -ne 0 ]; then
    echo -e "${RED}Failed to configure Docker to use Minikube. Exiting.${NC}"
    exit 1
fi
echo -e "${GREEN}✓ Docker configured to use Minikube's registry${NC}"
```

Output of Step 4:

```
[STEP 4] CONFIGURING DOCKER TO USE MINIKUBE
This allows us to build images directly into Minikube's registry
```

Step5:

```
# Step 5: Build the Docker image
echo -e "${MAGENTA}[STEP 5] BUILDING DOCKER IMAGE${NC}"

echo -e "${CYAN}Building k8s-master-app:latest image...${NC}"
cd ~/k8s-master-app/app

# Added retry mechanism for Docker build with better network settings
MAX_ATTEMPTS=3
BUILD_SUCCESS=false

for ATTEMPT in $(seq 1 $MAX_ATTEMPTS); do
    echo -e "${YELLOW}Build attempt $ATTEMPT of $MAX_ATTEMPTS...${NC}"

    # Use host network for better connectivity in WSL
    docker build --network=host -t k8s-master-app:latest .
    # Builds an image named k8s-master-app:latest from the current directory (.).
    # --network=host to avoid network-related issues.

    if [ $? -eq 0 ]; then
        BUILD_SUCCESS=true
        break
    else
        echo -e "${YELLOW}Build attempt $ATTEMPT failed. Waiting before retry...${NC}"
        sleep 5
    fi
done

if [ "$BUILD_SUCCESS" != "true" ]; then
    echo -e "${RED}Failed to build Docker image after $MAX_ATTEMPTS attempts. Exiting.${NC}"
    exit 1
fi

echo -e "${GREEN}✓ Docker image built successfully${NC}"
docker images | grep k8s-master-app
```

Output of Step 5:

```
[STEP 5] BUILDING DOCKER IMAGE
Building k8s-master-app:latest image...
Build attempt 1 of 3...
[+] Building 3.6s (15/15) FINISHED                                                          docker:default
 => [internal] load build definition from Dockerfile                                                  0.0s
 => => transferring dockerfile: 1.62kB                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.9                                         1.8s
 => [auth] library/python:pull token for registry-1.docker.io                                         0.0s
 => [internal] load .dockerignore                                                                     0.1s
 => => transferring context: 2B                                                                       0.0s
 => [1/9] FROM docker.io/library/python:3.9@sha256:5ea663a1c6ba266fdcac5949d1d2ea364ce30a2da92a3df95bb3c01437633ad9   0.0s
 => [internal] load build context                                                                     0.0s
 => => transferring context: 25.22kB                                                                  0.0s
 => CACHED [2/9] WORKDIR /app                                                                         0.0s
 => CACHED [3/9] RUN mkdir -p /data /config /logs &&     chmod 777 /data /config /logs                0.0s
 => CACHED [4/9] COPY requirements.txt .                                                              0.0s
 => CACHED [5/9] RUN pip install --no-cache-dir --upgrade pip &&     pip install --no-cache-dir -r requirements.txt   0.0s
 => [6/9] COPY app.py .                                                                               0.1s
 => [7/9] RUN chmod +x app.py                                                                         0.3s
 => [8/9] RUN echo '#!/bin/sh' > /healthcheck.sh &&     echo 'curl -s http://localhost:5000/api/health || exit 1' >> /healthcheck.sh &&   0.3s
 => [9/9] RUN useradd -m appuser &&     chown -R appuser:appuser /app /data /config /logs             0.5s
 => exporting to image                                                                                0.3s
 => => exporting layers                                                                               0.2s
 => => writing image sha256:3fd49816bef32c436402dc39118316a21c2f6fd05acb5d433ba49580c8e5b40e           0.0s
 => => naming to docker.io/library/k8s-master-app:latest                                              0.0s
✓ Docker image built successfully
k8s-master-app                      latest    3fd49816bef3   Less than a second ago   1.02GB
```

Step6:

```
# Step 6: Apply Kubernetes manifests
echo -e "${MAGENTA}[STEP 6] DEPLOYING TO KUBERNETES${NC}"
cd ~/k8s-master-app

echo -e "${CYAN}Creating namespace...${NC}"
kubectl apply -f k8s/base/namespace.yaml

echo -e "${CYAN}Creating ConfigMap and sample files...${NC}"
kubectl apply -f k8s/volumes/volumes.yaml

echo -e "${CYAN}Creating ConfigMap for application settings...${NC}"
kubectl apply -f k8s/config/configmap.yaml

echo -e "${CYAN}Creating Secret...${NC}"
kubectl apply -f k8s/config/secret.yaml

echo -e "${CYAN}Creating Deployment...${NC}"
kubectl apply -f k8s/base/deployment.yaml
#Defines how the application pods should run, their container image, replicas, and resource limits.

echo -e "${CYAN}Creating Service...${NC}"
kubectl apply -f k8s/networking/service.yaml
#Creates a Service to expose the application for external access.

echo -e "${CYAN}Creating HorizontalPodAutoscaler...${NC}"
kubectl apply -f k8s/monitoring/hpa.yaml || true
if [ $? -ne 0 ]; then
    echo -e "${YELLOW}Warning: Failed to create HorizontalPodAutoscaler. This is expected if metrics-server is not enabled.${NC}"
    echo -e "${YELLOW}The application will still work without auto-scaling.${NC}"
fi

echo -e "${GREEN}✓ All Kubernetes resources applied${NC}"
```

Output of Step 6:

```
[STEP 6] DEPLOYING TO KUBERNETES
Creating namespace...
namespace/k8s-demo unchanged
Creating ConfigMap and sample files...
configmap/app-files unchanged
Creating ConfigMap for application settings...
configmap/app-config unchanged
Creating Secret...
secret/app-secrets unchanged
Creating Deployment...
deployment.apps/k8s-master-app configured
Creating Service...
service/k8s-master-app unchanged
Creating HorizontalPodAutoscaler...
horizontalpodautoscaler.autoscaling/k8s-master-hpa configured
✓ All Kubernetes resources applied
```

Step7:

```
# Step 7: Wait for deployment to be ready
echo -e "${MAGENTA}[STEP 7] WAITING FOR DEPLOYMENT TO BE READY${NC}"
echo -e "${CYAN}This may take a minute or two...${NC}"

echo "Waiting for deployment to be ready..."
kubectl -n k8s-demo rollout status deployment/k8s-master-app --timeout=180s

if [ $? -ne 0 ]; then
    echo -e "${RED}Deployment failed to become ready within the timeout period.${NC}"
    echo -e "${YELLOW}Checking pod status...${NC}"

    kubectl -n k8s-demo get pods

    echo -e "${YELLOW}Checking pod logs...${NC}"
    POD=$(kubectl -n k8s-demo get pods -l app=k8s-master -o name | head -1)
    #-l (or --selector): Filters pods based on labels.
    #-o (or --output): Controls the format of the output.
# name: Returns the pod names in the format pod/<pod-name> instead of a detailed table.
#head -1: Extracts only the first line

    if [ ! -z "$POD" ]; then
    #!: Negates the condition,-z "$POD": Checks if POD is empty (i.e., "" or not set).
        kubectl -n k8s-demo logs $POD
    fi
else
    echo -e "${GREEN}✓ Deployment is ready${NC}"
fi
```

Output of Step 7:

```
[STEP 7] WAITING FOR DEPLOYMENT TO BE READY
This may take a minute or two...
Waiting for deployment to be ready...
deployment "k8s-master-app" successfully rolled out
✓ Deployment is ready
```

Step8:

$ deploy.sh

```
203
204   # Step 8: Set up port forwarding for easier access
205   echo -e "${MAGENTA}[STEP 8] SETTING UP PORT FORWARDING${NC}"
206   echo -e "${CYAN}This will make the application accessible on localhost${NC}"
207
208   # To avoid conflicts from multiple port-forward processes running at the same time.
209   # Checks if a kubectl port-forward process is already running for k8s-demo.
210   # If found, pgrep returns the process ID(s).
211   # If a process is found (> /dev/null means we discard output):
212   # pkill -f "kubectl.*port-forward.*k8s-demo"
213   # Kills any existing port-forwarding processes related to k8s-demo.
214   if pgrep -f "kubectl.*port-forward.*k8s-demo" > /dev/null; then
215       echo -e "${YELLOW}Port forwarding is already running. Stopping it...${NC}"
216       pkill -f "kubectl.*port-forward.*k8s-demo"
217   fi
218
219   # Start port forwarding in the background
220   #Forwards port 80 on the service k8s-master-app
221   # To port 8080 on localhost., & Runs the process in the background.
222   kubectl -n k8s-demo port-forward svc/k8s-master-app 8080:80 &
223   PORT_FORWARD_PID=$! #$!: Captures the process ID (PID) of the last background command,Stores it in PORT_FORWARD_PID for later checks.
224
225   # Give it a moment to start,Pauses execution for 2 seconds to allow port forwarding to stabilize.
226   sleep 2
227
228   # Check if port forwarding started successfully
229   #/dev/null is a special file in Linux that discards any data written to it.("BLACK HOLE")
230   #ensures only the exit status is checked, not the output.
231   #The script only cares whether the process exists (exit code), not the output.
232   # If process exists: ps -p $PORT_FORWARD_PID returns exit code 0 (success).
233   # If process doesn't exist: It returns a non-zero exit code (failure).
234   #By redirecting output to /dev/null, the script can check the exit code silently
235   if ! ps -p $PORT_FORWARD_PID > /dev/null; then
236       echo -e "${RED}Failed to start port forwarding.${NC}"
237   else
238       echo -e "${GREEN}✓ Port forwarding started on port 8080${NC}"
239   fi
240
```

Output of Step 8:

If facing this error:

```
[STEP 8] SETTING UP PORT FORWARDING
This will make the application accessible on localhost
Unable to listen on port 8080: Listeners failed to create with the following errors: [unable to create listener: Error listen tcp4 127.0.0.1:808
0: bind: address already in use unable to create listener: Error listen tcp6 [::1]:8080: bind: address already in use]
error: unable to listen on any of the requested ports: [{8080 5000}]
Failed to start port forwarding.
```

netstat -tulnp | grep :8080 #to get process id command to check what is running on 8080 port if there was earlier

sudo pkill -9 <PID> # to actually kill the process

```
[STEP 8] SETTING UP PORT FORWARDING
This will make the application accessible on localhost
Forwarding from 127.0.0.1:8080 -> 5000
Forwarding from [::1]:8080 -> 5000
```

Step9:

```
# Step 9: Display access information
echo -e "${MAGENTA}[STEP 9] DEPLOYMENT COMPLETE${NC}"
echo -e "${BLUE}================================================================================${NC}"
echo -e "${GREEN}Kubernetes Zero to Hero application has been deployed!${NC}"
echo -e "${BLUE}================================================================================${NC}"

echo -e "${YELLOW}Your application is accessible via multiple methods:${NC}"
echo ""
echo -e "${CYAN}1. Port Forwarding:${NC}"
echo "   URL: http://localhost:8080"
echo "   (This is running in the background with PID $PORT_FORWARD_PID)"
echo ""

# Get Minikube IP
MINIKUBE_IP=$(minikube ip)
echo -e "${CYAN}2. NodePort:${NC}"
echo "   URL: http://$MINIKUBE_IP:30080"
echo ""

echo -e "${CYAN}3. Minikube Service URL:${NC}"
echo "   Run: minikube service k8s-master-app -n k8s-demo"
echo ""
```

Output of Step 9:

```
[STEP 9] DEPLOYMENT COMPLETE
================================================================================
Kubernetes Zero to Hero application has been deployed!
================================================================================
Your application is accessible via multiple methods:

1. Port Forwarding:
   URL: http://localhost:8080
   (This is running in the background with PID 58472)

2. NodePort:
   URL: http://192.168.49.2:30080

3. Minikube Service URL:
   Run: minikube service k8s-master-app -n k8s-demo

================================================================================
USEFUL COMMANDS
```

CODE: for deploy.sh

```
#!/bin/bash
# Deployment script for the Kubernetes Zero to Hero application
# This script automates the entire deployment process to Minikube
# REVISED VERSION: Works around WSL2 mounting limitations

# Color definitions for better output
GREEN='\033[0;32m'
BLUE='\033[0;34m'
YELLOW='\033[0;33m'
RED='\033[0;31m'
CYAN='\033[0;36m'
MAGENTA='\033[0;35m'
NC='\033[0m' # No Color

echo -e
"${BLUE}========================================================================
${NC}"
echo -e "${BLUE}             KUBERNETES ZERO TO HERO -
DEPLOYMENT                    ${NC}"
echo -e
"${BLUE}========================================================================
${NC}"

# Function to check if a command exists
#/dev/null This prevents unnecessary output from appearing in the terminal.
command_exists() {
    command -v "$1" &> /dev/null
}

# Step 1: Check prerequisites
echo -e "${MAGENTA}[STEP 1] CHECKING PREREQUISITES${NC}"

# Check for required tools
for tool in minikube kubectl docker; do
    if ! command_exists $tool; then
        echo -e "${RED}Error: $tool is not installed. Please install it and
try again.${NC}"
        exit 1
    fi
    echo -e "${GREEN}✓ $tool is installed${NC}"
done

# Step 2: Ensure Minikube is running
echo -e "${MAGENTA}[STEP 2] ENSURING MINIKUBE IS RUNNING${NC}"

if ! minikube status | grep -q "host: Running"; then
    echo -e "${YELLOW}Minikube is not running. Starting Minikube...${NC}"
```

```
    # Start Minikube with appropriate configuration
    minikube start --cpus=2 --memory=4096 --disk-size=20g

    if [ $? -ne 0 ]; then #$? → Stores the exit status of the most recent
command
    #0 → The exit code 0 typically means success.
        echo -e "${RED}Failed to start Minikube. Trying with minimal
configuration...${NC}"

        # Fallback to minimal configuration
        minikube start --driver=docker #Uses Docker as the virtualization
driver , Runs Kubernetes inside a Docker container instead of a virtual
machine.

        if [ $? -ne 0 ]; then
            echo -e "${RED}Failed to start Minikube. Exiting.${NC}"
            exit 1
        fi
    fi
else
    echo -e "${GREEN}✓ Minikube is already running${NC}"
fi

# Step 3: Enable required Minikube addons
echo -e "${MAGENTA}[STEP 3] ENABLING MINIKUBE ADDONS${NC}"

# We'll handle addons more carefully, checking if they're already enabled
# and handling errors better

# Function to safely enable an addon
enable_addon() {
    local addon=$1
    local already_enabled=$(minikube addons list | grep $addon | grep -c
"enabled")

    if [ $already_enabled -eq 1 ]; then
        echo -e "${GREEN}✓ $addon addon is already enabled${NC}"
        return 0
    fi

    echo -e "${CYAN}Enabling $addon addon...${NC}"-
    minikube addons enable $addon

    if [ $? -ne 0 ]; then
        echo -e "${YELLOW}Warning: Failed to enable $addon addon. Continuing
without it.${NC}"
        return 1
```

```
    else
        echo -e "${GREEN}✓ $addon addon enabled${NC}"
        return 0
    fi
}

# Try to enable addons but don't fail if they don't work
enable_addon "dashboard" || true

echo -e "${YELLOW}Note: Skipping Ingress and Metrics Server addons as they may
not work in all environments${NC}"
echo -e "${YELLOW}The application will still work without these addons${NC}"

# Step 4: Configure Docker to use Minikube's Docker daemon
echo -e "${MAGENTA}[STEP 4] CONFIGURING DOCKER TO USE MINIKUBE${NC}"
echo -e "${CYAN}This allows us to build images directly into Minikube's
registry${NC}"

#minikube docker-env prints environment variables needed to redirect Docker
commands to Minikube’s internal Docker daemon.
eval $(minikube docker-env)
if [ $? -ne 0 ]; then
    echo -e "${RED}Failed to configure Docker to use Minikube. Exiting.${NC}"
    exit 1
fi
echo -e "${GREEN}✓ Docker configured to use Minikube's registry${NC}"

# Step 5: Build the Docker image
echo -e "${MAGENTA}[STEP 5] BUILDING DOCKER IMAGE${NC}"

echo -e "${CYAN}Building k8s-master-app:latest image...${NC}"
cd ~/k8s-master-app/app

# Added retry mechanism for Docker build with better network settings
MAX_ATTEMPTS=3
BUILD_SUCCESS=false

for ATTEMPT in $(seq 1 $MAX_ATTEMPTS); do
    echo -e "${YELLOW}Build attempt $ATTEMPT of $MAX_ATTEMPTS...${NC}"

    # Use host network for better connectivity in WSL
    docker build --network=host -t k8s-master-app:latest .
    # Builds an image named k8s-master-app:latest from the current directory
(.).
    # --network=host to avoid network-related issues.

    if [ $? -eq 0 ]; then
        BUILD_SUCCESS=true
```

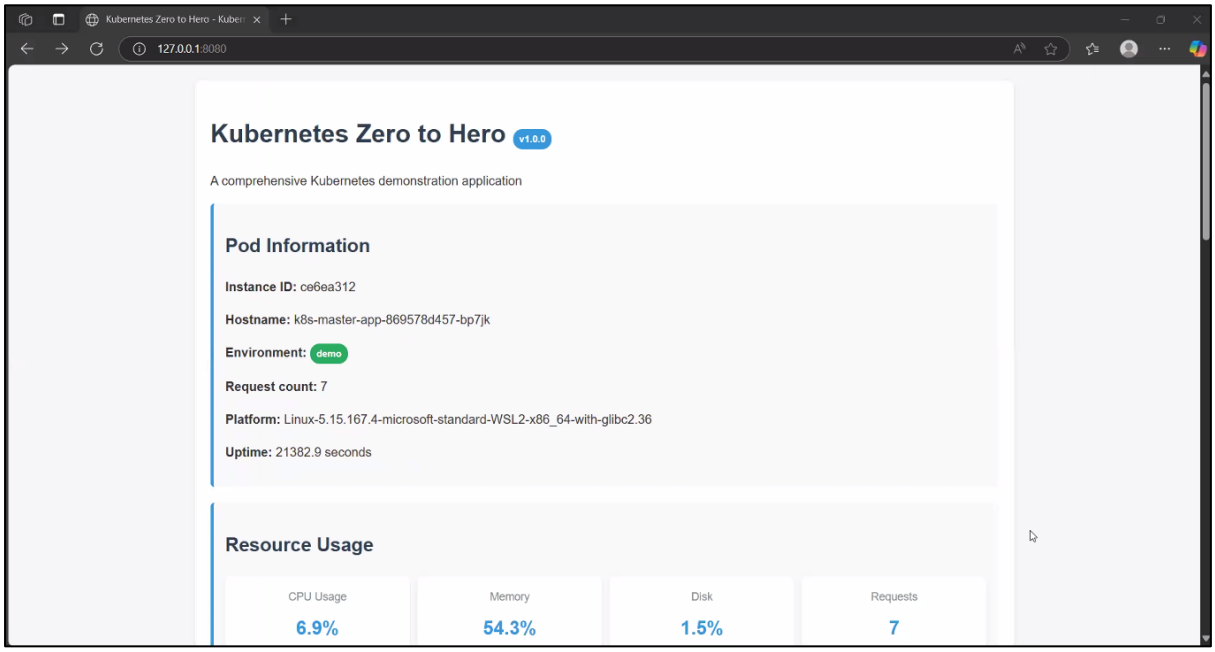
```
        break
    else
        echo -e "${YELLOW}Build attempt $ATTEMPT failed. Waiting before
retry...${NC}"
        sleep 5
    fi
done

if [ "$BUILD_SUCCESS" != "true" ]; then
    echo -e "${RED}Failed to build Docker image after $MAX_ATTEMPTS attempts.
Exiting.${NC}"
    exit 1
fi

echo -e "${GREEN}✓ Docker image built successfully${NC}"
docker images | grep k8s-master-app

# Step 6: Apply Kubernetes manifests
echo -e "${MAGENTA}[STEP 6] DEPLOYING TO KUBERNETES${NC}"
cd ~/k8s-master-app

echo -e "${CYAN}Creating namespace...${NC}"
kubectl apply -f k8s/base/namespace.yaml

echo -e "${CYAN}Creating ConfigMap and sample files...${NC}"
kubectl apply -f k8s/volumes/volumes.yaml

echo -e "${CYAN}Creating ConfigMap for application settings...${NC}"
kubectl apply -f k8s/config/configmap.yaml

echo -e "${CYAN}Creating Secret...${NC}"
kubectl apply -f k8s/config/secret.yaml

echo -e "${CYAN}Creating Deployment...${NC}"
kubectl apply -f k8s/base/deployment.yaml
#Defines how the application pods should run, their container image, replicas,
and resource limits.

echo -e "${CYAN}Creating Service...${NC}"
kubectl apply -f k8s/networking/service.yaml
#Creates a Service to expose the application for external access.

echo -e "${CYAN}Creating HorizontalPodAutoscaler...${NC}"
kubectl apply -f k8s/monitoring/hpa.yaml || true
if [ $? -ne 0 ]; then
    echo -e "${YELLOW}Warning: Failed to create HorizontalPodAutoscaler. This
is expected if metrics-server is not enabled.${NC}"
```

```
    echo -e "${YELLOW}The application will still work without auto-
scaling.${NC}"
fi

echo -e "${GREEN}✓ All Kubernetes resources applied${NC}"

# Step 7: Wait for deployment to be ready
echo -e "${MAGENTA}[STEP 7] WAITING FOR DEPLOYMENT TO BE READY${NC}"
echo -e "${CYAN}This may take a minute or two...${NC}"

echo "Waiting for deployment to be ready..."
kubectl -n k8s-demo rollout status deployment/k8s-master-app --timeout=180s

if [ $? -ne 0 ]; then
    echo -e "${RED}Deployment failed to become ready within the timeout
period.${NC}"
    echo -e "${YELLOW}Checking pod status...${NC}"

    kubectl -n k8s-demo get pods

    echo -e "${YELLOW}Checking pod logs...${NC}"
    POD=$(kubectl -n k8s-demo get pods -l app=k8s-master -o name | head -1)
    #-l (or --selector): Filters pods based on labels.
    #-o (or --output): Controls the format of the output.
# name: Returns the pod names in the format pod/<pod-name> instead of a
detailed table.
#head -1: Extracts only the first line

    if [ ! -z "$POD" ]; then
    #!: Negates the condition,-z "$POD": Checks if POD is empty (i.e., "" or
not set).
        kubectl -n k8s-demo logs $POD
    fi
else
    echo -e "${GREEN}✓ Deployment is ready${NC}"
fi

# Step 8: Set up port forwarding for easier access
echo -e "${MAGENTA}[STEP 8] SETTING UP PORT FORWARDING${NC}"
echo -e "${CYAN}This will make the application accessible on localhost${NC}"

# To avoid conflicts from multiple port-forward processes running at the same
time.
# Checks if a kubectl port-forward process is already running for k8s-demo.
# If found, pgrep returns the process ID(s).
# If a process is found (> /dev/null means we discard output):
# pkill -f "kubectl.*port-forward.*k8s-demo"
# Kills any existing port-forwarding processes related to k8s-demo.
```

```
if pgrep -f "kubectl.*port-forward.*k8s-demo" > /dev/null; then
    echo -e "${YELLOW}Port forwarding is already running. Stopping it...${NC}"
    pkill -f "kubectl.*port-forward.*k8s-demo"
fi

# Start port forwarding in the background
#Forwards port 80 on the service k8s-master-app
# To port 8080 on localhost., & Runs the process in the background.
kubectl -n k8s-demo port-forward svc/k8s-master-app 8080:80 &
PORT_FORWARD_PID=$! #$!: Captures the process ID (PID) of the last background
command,Stores it in PORT_FORWARD_PID for later checks.

# Give it a moment to start,Pauses execution for 2 seconds to allow port
forwarding to stabilize.
sleep 2

# Check if port forwarding started successfully
#/dev/null is a special file in Linux that discards any data written to
it.("BLACK HOLE")
#ensures only the exit status is checked, not the output.
#The script only cares whether the process exists (exit code), not the output.
# If process exists: ps -p $PORT_FORWARD_PID returns exit code 0 (success).
# If process doesn't exist: It returns a non-zero exit code (failure).
#By redirecting output to /dev/null, the script can check the exit code
silently
if ! ps -p $PORT_FORWARD_PID > /dev/null; then
    echo -e "${RED}Failed to start port forwarding.${NC}"
else
    echo -e "${GREEN}✓ Port forwarding started on port 8080${NC}"
fi

# Step 9: Display access information
echo -e "${MAGENTA}[STEP 9] DEPLOYMENT COMPLETE${NC}"
echo -e
"${BLUE}==========================================================================
${NC}"
echo -e "${GREEN}Kubernetes Zero to Hero application has been deployed!${NC}"
echo -e
"${BLUE}==========================================================================
${NC}"

echo -e "${YELLOW}Your application is accessible via multiple methods:${NC}"
echo ""
echo -e "${CYAN}1. Port Forwarding:${NC}"
echo "   URL: http://localhost:8080"
echo "   (This is running in the background with PID $PORT_FORWARD_PID)"
echo ""
```

```
# Get Minikube IP
MINIKUBE_IP=$(minikube ip)
echo -e "${CYAN}2. NodePort:${NC}"
echo "   URL: http://$MINIKUBE_IP:30080"
echo ""

echo -e "${CYAN}3. Minikube Service URL:${NC}"
echo "   Run: minikube service k8s-master-app -n k8s-demo"
echo ""

# Step 10: Display useful commands
echo -e
"${BLUE}========================================================================
${NC}"
echo -e "${YELLOW}USEFUL COMMANDS:${NC}"
echo -e
"${BLUE}========================================================================
${NC}"
echo -e "${CYAN}View the Kubernetes Dashboard:${NC}"
echo "   minikube dashboard"
echo ""
echo -e "${CYAN}View application logs:${NC}"
echo "   kubectl -n k8s-demo logs -l app=k8s-master"
echo ""
echo -e "${CYAN}Get a shell into a pod:${NC}"
echo "   kubectl -n k8s-demo exec -it $(kubectl -n k8s-demo get pods -l
app=k8s-master -o name | head -1) -- /bin/bash"
echo ""
echo -e "${CYAN}View all resources in the namespace:${NC}"
echo "   kubectl -n k8s-demo get all"
echo ""
echo -e "${CYAN}Check pod resource usage (if metrics-server is enabled):${NC}"
echo "   kubectl -n k8s-demo top pods"
echo ""
echo -e "${CYAN}Clean up all resources:${NC}"
echo "   ./scripts/cleanup.sh"
echo ""
echo -e "${CYAN}Stop port forwarding:${NC}"
echo "   kill $PORT_FORWARD_PID"
echo ""
echo -e
"${BLUE}========================================================================
${NC}"
echo -e "${GREEN}DEPLOYMENT SUCCESSFUL!${NC}"
echo -e
"${BLUE}========================================================================
${NC}"
echo -e "${YELLOW}Enjoy exploring your Kubernetes application!${NC}"
```

output after the main.py:




```
#Checks if it is readable (os.access(PATH, os.R_OK)).
# Stores this information in a dictionary under the volumes key.

            'config': {
                'path': CONFIG_PATH,
                'mounted': os.path.exists(CONFIG_PATH) and os.access(CONFIG_PATH, os.R_OK)
            },
            'logs': {
                'path': LOG_PATH,
                'mounted': os.path.exists(LOG_PATH) and os.access(LOG_PATH, os.R_OK)
            }
        },
        'timestamp': datetime.datetime.now().isoformat()
    })
```

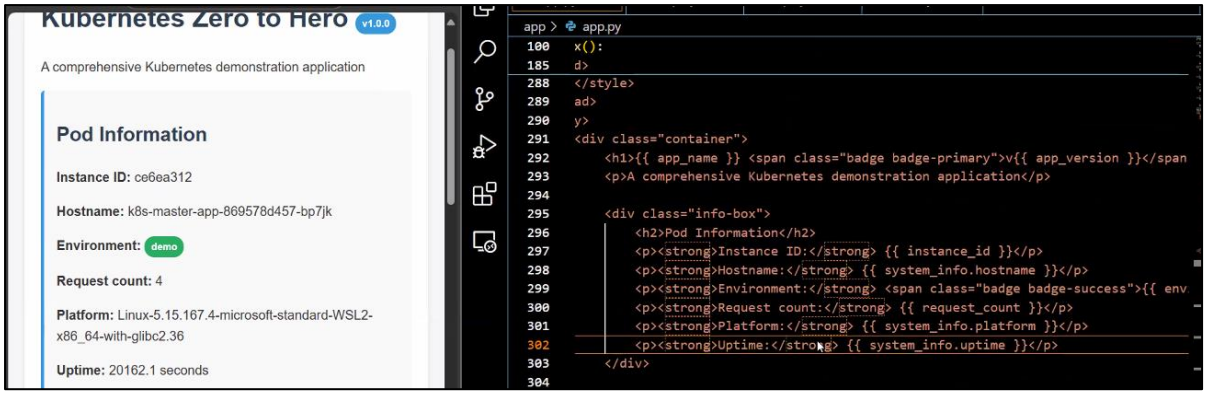
Example Output

- If `LOG_PATH` is set to `/var/log/myapp`, then:

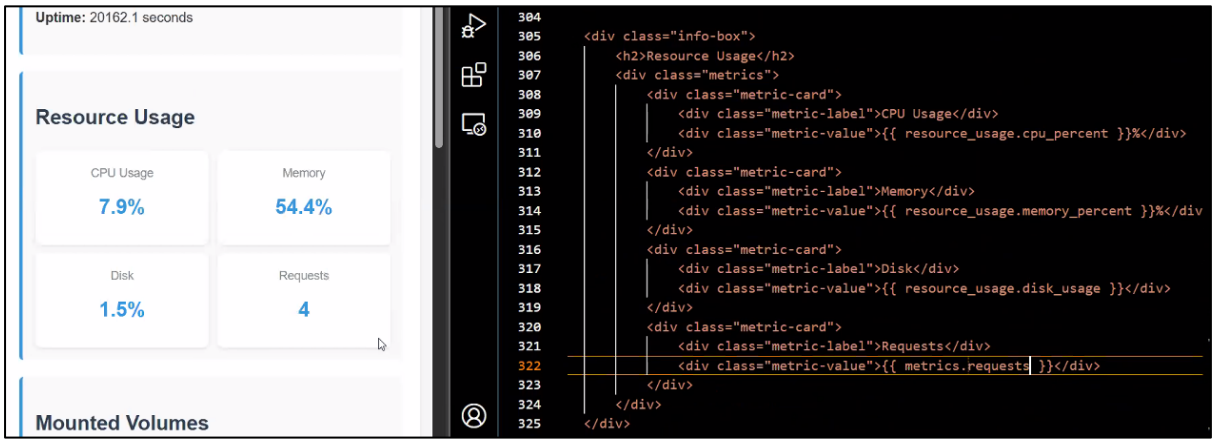
```python
Clog_file = "/var/log/myapp/app.log"
```

- If `LOG_PATH` is **not set**, then:

```python
Clog_file = "/logs/app.log"
```


Kubernetes Zero to Hero
v1.0.0
A comprehensive Kubernetes demonstration application
Pod Information
Instance ID: ce6ea312
Hostname: k8s-master-app-869578d457-bp7jk
Environment: demo
Request count: 4
Platform: Linux-5.15.167.4-microsoft-standard-WSL2-x86_64-with-glibc2.36
Uptime: 20162.1 seconds
app > app.py
<h1>{{ app_name }} v{{ app_version }}</span
A comprehensive Kubernetes demonstration application

<h2>Pod Information</h2>
Instance ID: {{ instance_id }}
Hostname: {{ system_info.hostname }}
Environment: {{ env
Request count: {{ request_count }}
Platform: {{ system_info.platform }}
Uptime: {{ system_info.uptime }}


Uptime: 20162.1 seconds
Resource Usage
CPU Usage
7.9%
Memory
54.4%
Disk
1.5%
Requests
4
Mounted Volumes

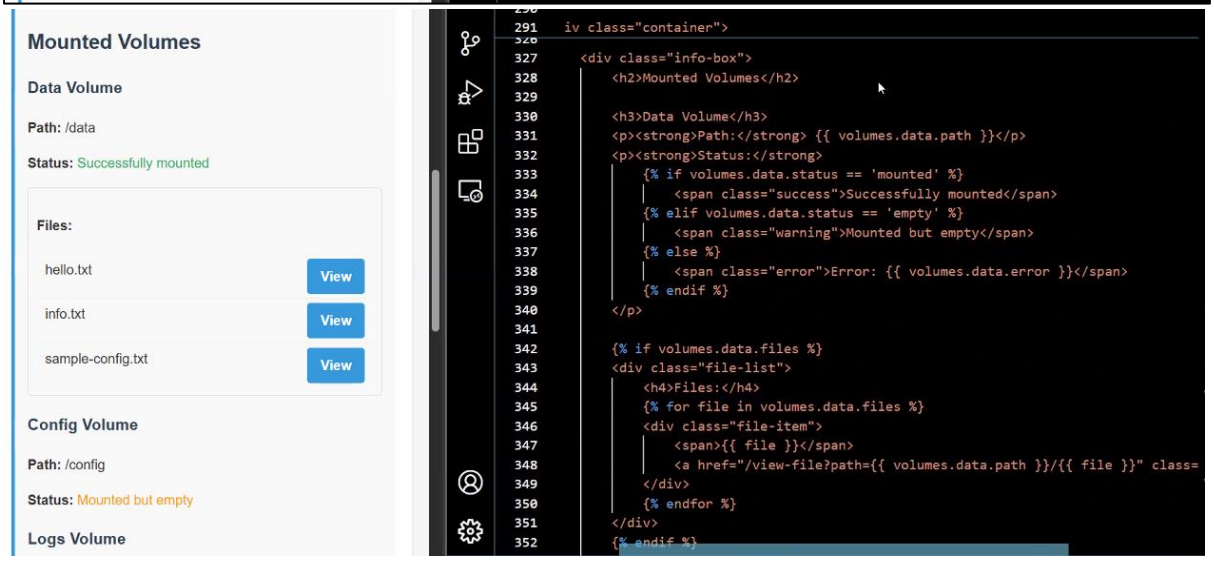
<h2>Resource Usage</h2>

CPU Usage
{{ resource_usage.cpu_percent }}%

Memory
{{ resource_usage.memory_percent }}%</div

Disk
{{ resource_usage.disk_usage }}

Requests
{{ metrics.requests }}


Mounted Volumes
Data Volume
Path: /data
Status: Successfully mounted
Files:
hello.txt
View
info.txt
View
sample-config.txt
View
Config Volume
Path: /config
Status: Mounted but empty
Logs Volume

<h2>Mounted Volumes</h2>
<h3>Data Volume</h3>
Path: {{ volumes.data.path }}
Status:
{% if volumes.data.status == 'mounted' %}
Successfully mounted
{% elif volumes.data.status == 'empty' %}
Mounted but empty
{% else %}
Error: {{ volumes.data.error }}
{% endif %}

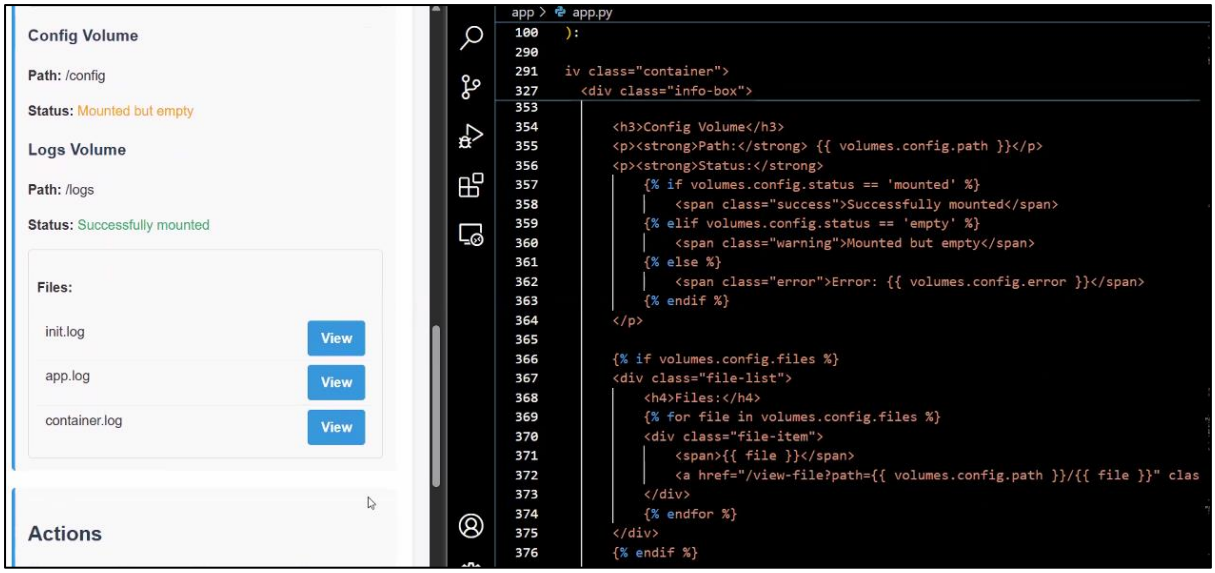
{% if volumes.data.files %}

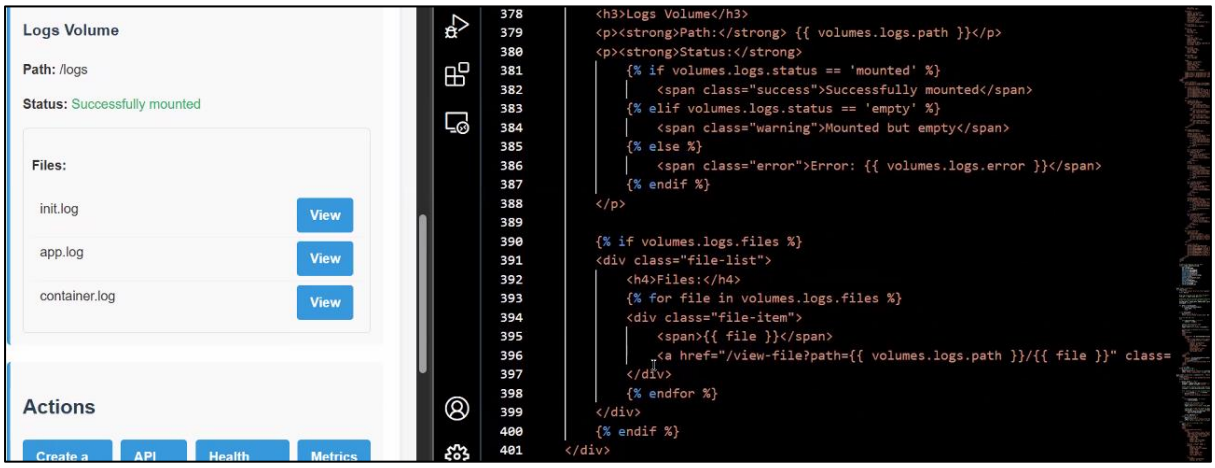
<h4>Files:</h4>
{% for file in volumes.data.files %}

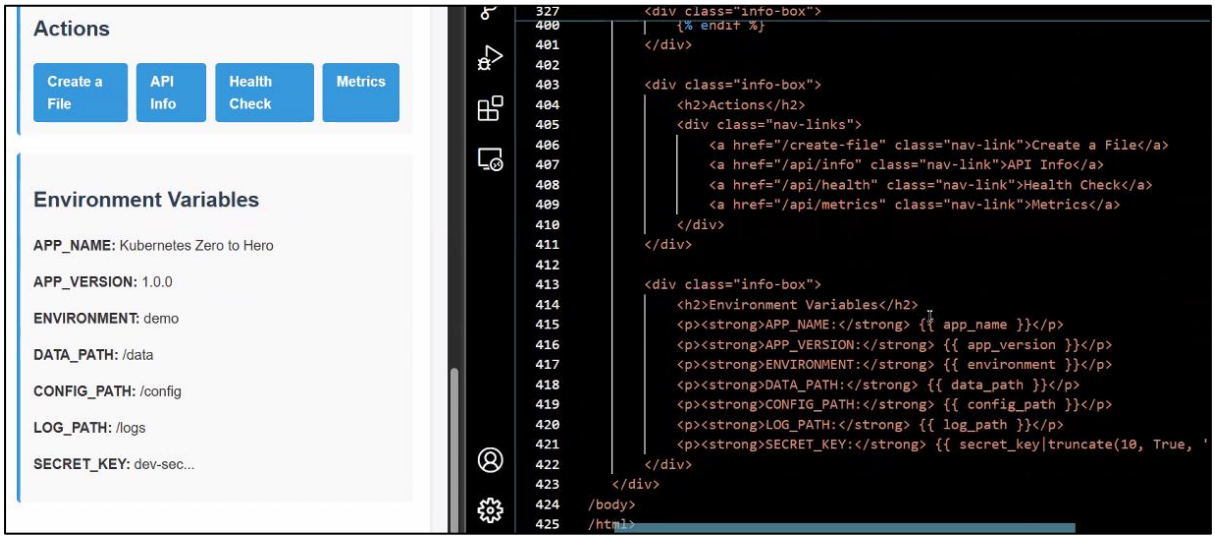
{{ file }}
<a href="/view-file?path={{ volumes.data.path }}/{{ file }}" class=

{% endfor %}

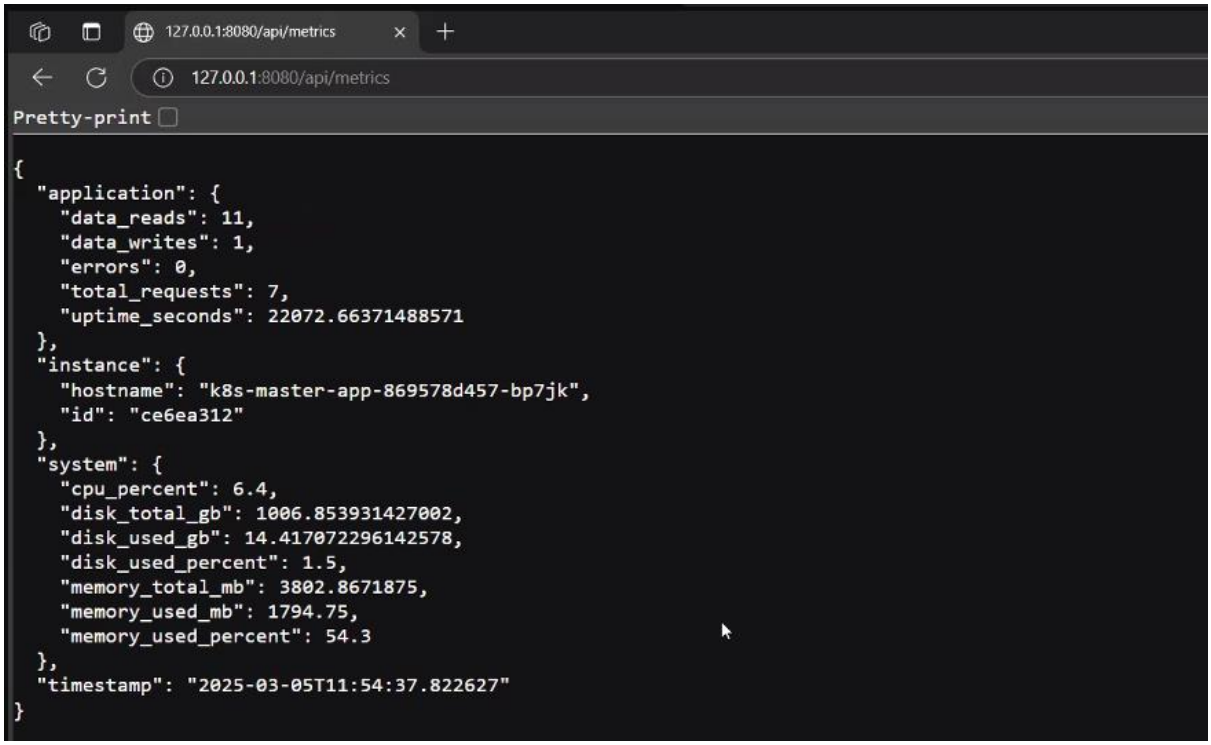
{% endif %}


Config Volume
Path: /config
Status: Mounted but empty
Logs Volume
Path: /logs
Status: Successfully mounted
Files:
init.log
View
app.log
View
container.log
View
Actions
app > app.py
100):
290
291 iv class="container">
327
353
354 <h3>Config Volume</h3>
355 Path: {{ volumes.config.path }}
356 Status:
357 {% if volumes.config.status == 'mounted' %}
358 Successfully mounted
359 {% elif volumes.config.status == 'empty' %}
360 Mounted but empty
361 {% else %}
362 Error: {{ volumes.config.error }}
363 {% endif %}
364
365
366 {% if volumes.config.files %}
367
368 <h4>Files:</h4>
369 {% for file in volumes.config.files %}
370
371 {{ file }}
372 <a href="/view-file?path={{ volumes.config.path }}/{{ file }}" clas
373
374 {% endfor %}
375
376 {% endif %}


Logs Volume
Path: /logs
Status: Successfully mounted
Files:
init.log
View
app.log
View
container.log
View
Actions
Create a
API
Health
Metrics
378 <h3>Logs Volume</h3>
379 Path: {{ volumes.logs.path }}
380 Status:
381 {% if volumes.logs.status == 'mounted' %}
382 Successfully mounted
383 {% elif volumes.logs.status == 'empty' %}
384 Mounted but empty
385 {% else %}
386 Error: {{ volumes.logs.error }}
387 {% endif %}
388
389
390 {% if volumes.logs.files %}
391
392 <h4>Files:</h4>
393 {% for file in volumes.logs.files %}
394
395 {{ file }}
396 <a href="/view-file?path={{ volumes.logs.path }}/{{ file }}" class=
397
398 {% endfor %}
399
400 {% endif %}
401


Actions
Create a File
API Info
Health Check
Metrics
Environment Variables
APP_NAME: Kubernetes Zero to Hero
APP_VERSION: 1.0.0
ENVIRONMENT: demo
DATA_PATH: /data
CONFIG_PATH: /config
LOG_PATH: /logs
SECRET_KEY: dev-sec...
327
400 {% endif %}
401
402
403
404 <h2>Actions</h2>
405
406 Create a File
407 API Info
408 Health Check
409 Metrics
410
411
412
413
414 <h2>Environment Variables</h2>
415 APP_NAME: {{ app_name }}
416 APP_VERSION: {{ app_version }}
417 ENVIRONMENT: {{ environment }}
418 DATA_PATH: {{ data_path }}
419 CONFIG_PATH: {{ config_path }}
420 LOG_PATH: {{ log_path }}
421 SECRET_KEY: {{ secret_key|truncate(10, True, '
422
423
424 /body>
425 /html>

```python
@app.route('/api/metrics')
def get_metrics():
    """API endpoint for application metrics - useful for monitoring systems"""
    # Get basic resource usage stats
    cpu_percent = psutil.cpu_percent()
    memory_info = psutil.virtual_memory()
    disk_info = psutil.disk_usage('/')

    # Collect all metrics
    all_metrics = {
        'system': {
            'cpu_percent': cpu_percent,
            'memory_used_percent': memory_info.percent,
            'memory_used_mb': memory_info.used / (1024 * 1024),
            'memory_total_mb': memory_info.total / (1024 * 1024),
            'disk_used_percent': disk_info.percent,
            'disk_used_gb': disk_info.used / (1024**3),
            'disk_total_gb': disk_info.total / (1024**3)
        },
        'application': {
            'uptime_seconds': time.time() - start_time,
            'total_requests': metrics['requests'],
            'data_reads': metrics['data_reads'],
            'data_writes': metrics['data_writes'],
            'errors': metrics['errors']
        },
        'instance': {
            'id': INSTANCE_ID,
            'hostname': socket.gethostname()
        },
        'timestamp': datetime.datetime.now().isoformat()
    }
```



```python
@app.route('/api/info')
def api_info():
    """API endpoint returning application information"""
    return jsonify({
        'app_name': APP_NAME,
        'version': APP_VERSION,
        'environment': ENVIRONMENT,
        'instance_id': INSTANCE_ID,
        'hostname': socket.gethostname(),
        'request_count': request_count,
        'uptime_seconds': time.time() - start_time,
# Kubernetes volumes allow persistent storage across container restarts.
# Volumes are mounted inside a container as directories.
# This lets the application store logs, config files, and data that persist beyond container restarts.
#Volumes are defined in your Kubernetes YAML files (deployment.yaml or volumes.yaml):
        'volumes': {
            'data': {
                'path': DATA_PATH,
                'mounted': os.path.exists(DATA_PATH) and os.access(DATA_PATH, os.R_OK)
            },
#Checks if it is readable (os.access(PATH, os.R_OK)).
# Stores this information in a dictionary under the volumes key.

            'config': {
                'path': CONFIG_PATH,
                'mounted': os.path.exists(CONFIG_PATH) and os.access(CONFIG_PATH, os.R_OK)
            },
            'logs': {
                'path': LOG_PATH,
                'mounted': os.path.exists(LOG_PATH) and os.access(LOG_PATH, os.R_OK)
            }
        },
        'timestamp': datetime.datetime.now().isoformat()
    })
```

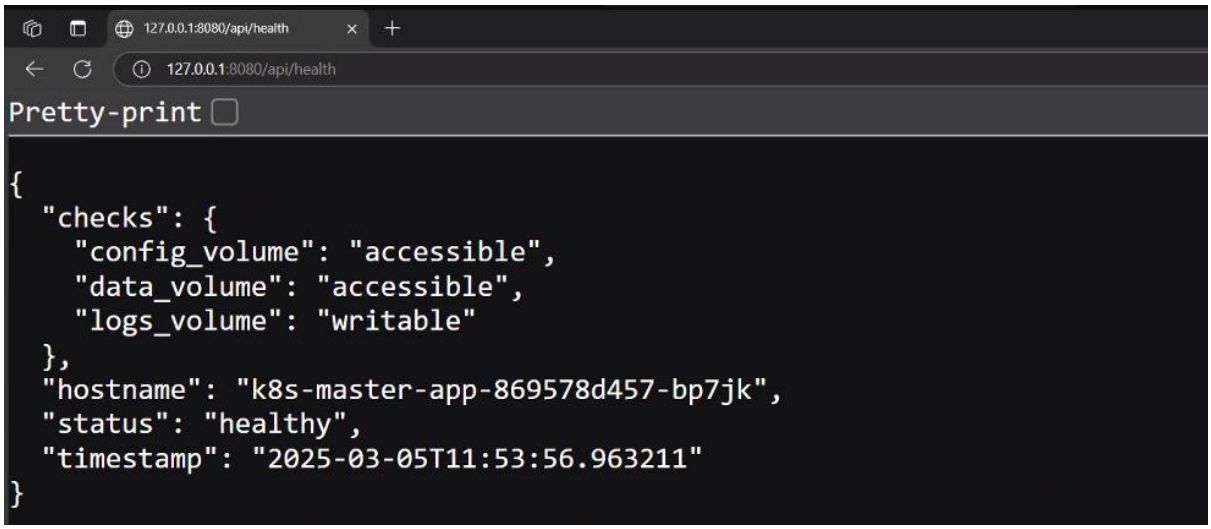
127.0.0.1:8080/api/info

127.0.0.1:8080/api/info

Pretty-print


```
{
  "app_name": "Kubernetes Zero to Hero",
  "environment": "demo",
  "hostname": "k8s-master-app-869578d457-bp7jk",
  "instance_id": "ce6ea312",
  "request_count": 7,
  "timestamp": "2025-03-05T11:54:18.452301",
  "uptime_seconds": 22053.293340444565,
  "version": "1.0.0",
  "volumes": {
    "config": {
      "mounted": true,
      "path": "/config"
    },
    "data": {
      "mounted": true,
      "path": "/data"
    },
    "logs": {
      "mounted": true,
      "path": "/logs"
    }
  }
}
```

```python
@app.route('/api/health')
def health_check():
    """Health check endpoint for Kubernetes liveness and readiness probes"""
    # Check if we can access our mounted volumes
    data_ok = os.path.exists(DATA_PATH) and os.access(DATA_PATH, os.R_OK)
    config_ok = os.path.exists(CONFIG_PATH) and os.access(CONFIG_PATH, os.R_OK)
    logs_ok = os.path.exists(LOG_PATH) and os.access(LOG_PATH, os.W_OK)

    # For a real application, you might check database connections,
    # cache availability, etc.

    # Overall health status
    is_healthy = data_ok and config_ok and logs_ok

    # Log health check results
    logger.info(f"Health check: {'PASS' if is_healthy else 'FAIL'}")

    response = {
        'status': 'healthy' if is_healthy else 'unhealthy',
        'checks': {
            'data_volume': 'accessible' if data_ok else 'inaccessible',
            'config_volume': 'accessible' if config_ok else 'inaccessible',
            'logs_volume': 'writable' if logs_ok else 'not writable'
        },
        'timestamp': datetime.datetime.now().isoformat(),
        'hostname': socket.gethostname()
    }

    # Set the HTTP status code based on health
    status_code = 200 if is_healthy else 503

    return jsonify(response), status_code
```



CODE OF APP.PY:

```
#!/usr/bin/env python3
"""
Kubernetes Master Application
=============================

This Flask application demonstrates:
1. Reading from and writing to mounted volumes
2. Working with environment variables (from ConfigMaps and Secrets)
3. Health checking
4. Resource usage reporting
5. Metrics collection

This showcases how a containerized application interacts with Kubernetes
features.
"""
# Flask: The core class to create a Flask web application.
# jsonify: Used to convert Python objects (like dictionaries) into JSON
responses.
# render_template_string: Allows rendering HTML templates from string inputs.
# request: Handles incoming HTTP requests (e.g., form data, query parameters).
# redirect, url_for: Helps redirect users to different routes within the
application.
from flask import Flask, jsonify, render_template_string, request, redirect,
url_for
import os #Interact with the operating system (e.g., environment variables,
file handling).
import socket # Handles network-related tasks (e.g., getting hostnames, IP
addresses).
import datetime #Works with date and time (e.g., logging timestamps,
calculating uptime).
import json #Enables JSON encoding and decoding.
import logging  #Manages logging messages (e.g., debugging, error tracking).
import uuid #Generates unique identifiers (e.g., session IDs, transaction
IDs).
import platform # Retrieves system information (OS, architecture, etc.).
import psutil  # For resource usage statistics , Monitors system resources
like CPU, memory, disk usage, etc.
import time # Provides time-related functions (e.g., sleep, timestamps).
import threading # Allows running background tasks asynchronously.
import sys #Provides system-specific functions (e.g., exiting the program,
checking Python version).

# Initialize Flask application
app = Flask(_name_)

log_file = os.path.join(os.environ.get('LOG_PATH', '/logs'), 'app.log')
logging.basicConfig(
```

```
    level=logging.INFO,#logging.INFO stores logs of levels INFO, WARNING,
ERROR, and CRITICAL will be recorded, but DEBUG messages will be ignored.
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',
    handlers=[
        logging.StreamHandler(),#outputs logs to the console.
        logging.FileHandler(log_file)#Saves logs to the specified file
(app.log).

    ]
)

logger = logging.getLogger('k8s-master-app')#Creates a named logger called
"k8s-master-app"

# Read configuration from environment variables (from ConfigMaps)
APP_NAME = os.environ.get('APP_NAME', 'k8s-master-app')
APP_VERSION = os.environ.get('APP_VERSION', '1.0.0')
ENVIRONMENT = os.environ.get('ENVIRONMENT', 'development')
DATA_PATH = os.environ.get('DATA_PATH', '/data')
CONFIG_PATH = os.environ.get('CONFIG_PATH', '/config')
LOG_PATH = os.environ.get('LOG_PATH', '/logs')

# Read secrets from environment variables (from Secrets)
# In a real app, these would be more sensitive values like API keys
SECRET_KEY = os.environ.get('SECRET_KEY', 'default-dev-key')
DB_PASSWORD = os.environ.get('DB_PASSWORD', 'default-password')

# Generate a (uui-Universally Unique Identifier)unique instance ID to
demonstrate statelessness ,each pod gets a unique INSTANCE_ID for
logging/debugging.
INSTANCE_ID = str(uuid.uuid4())[:8]

# Track request count and application metrics
# Performance Monitoring: Helps measure how many requests/errors occur.
# Can be exported to Prometheus or other monitoring tool
request_count = 0
start_time = time.time()
metrics = {
    'requests': 0,
    'errors': 0,
    'data_reads': 0,
    'data_writes': 0
}

# Simulate application load for resource usage demonstration,Runs a background
task that continuously performs some CPU work.
```

```
#Logs debug messages every 1000 iterations.(Useful for Kubernetes auto-scaling
)
def background_worker():
    """
    Simulate background work to demonstrate resource usage.
    In a real app, this might be processing tasks, etc.
    """
    logger.info("Background worker started")
    counter = 0
    while True:
        # Simple CPU work - calculate prime numbers
        counter += 1
        if counter % 1000 == 0:
            # Log occasionally to show activity
            logger.debug(f"Background worker tick: {counter}")
        time.sleep(0.1)  # Don't use too much CPU

# Start the background worker
worker_thread = threading.Thread(target=background_worker, daemon=True)
worker_thread.start()

@app.route('/')
def index():
    """Main page showing application status and mounted volume information"""
    global request_count, metrics
    request_count += 1
    metrics['requests'] += 1

    # Log the request
    logger.info(f"Request to index page from {request.remote_addr}")

    # Get system information
    system_info = {
        'hostname': socket.gethostname(),#Unique name of the instance.
        'platform': platform.platform(),# OS information.
        'python_version': platform.python_version(),
        'cpu_count': psutil.cpu_count(),#Number of CPU cores.
        'memory': f"{psutil.virtual_memory().total / (1024 * 1024):.1f}
MB",#Total RAM available.
        'uptime': f"{time.time() - start_time:.1f} seconds"# How long the
container has been running.

    }

    # Get resource usage Helps in troubleshooting high resource usage.
    resource_usage = {
        'cpu_percent': psutil.cpu_percent(),
        'memory_percent': psutil.virtual_memory().percent,
```

```
        'disk_usage': f"{psutil.disk_usage('/').percent}%"
    }

    # Get information about mounted volumes
    volumes = {}
    #   volumes allow data to persist across restarts

    # Check data volume
    try:
        data_files = os.listdir(DATA_PATH)
        volumes['data'] = {
            'path': DATA_PATH,#Tries to list files in the DATA_PATH volume.
            'files': data_files,#If successful, it stores the list of files.
            'status': 'mounted' if data_files else 'empty'#If an error occurs,
it logs the error and increments metrics['errors'].
        }
        metrics['data_reads'] += 1

    except Exception as e:
        volumes['data'] = {
            'path': DATA_PATH,
            'error': str(e),
            'status': 'error'
        }
        metrics['errors'] += 1

    # Check config volume
    try:
        config_files = os.listdir(CONFIG_PATH)
        volumes['config'] = {
            'path': CONFIG_PATH,
            'files': config_files,
            'status': 'mounted' if config_files else 'empty'
        }
    except Exception as e:
        volumes['config'] = {
            'path': CONFIG_PATH,
            'error': str(e),
            'status': 'error'
        }
        metrics['errors'] += 1

    # Check logs volume
    try:
        logs_files = os.listdir(LOG_PATH)
        volumes['logs'] = {
            'path': LOG_PATH,
            'files': logs_files,
```

```
            'status': 'mounted' if logs_files else 'empty'
        }
    except Exception as e:
        volumes['logs'] = {
            'path': LOG_PATH,
            'error': str(e),
            'status': 'error'
        }
        metrics['errors'] += 1

    # Build HTML content using a template
    html_content = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>{{ app_name }} - Kubernetes Master App</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            body {
                font-family: Arial, sans-serif;
                line-height: 1.6;
                margin: 0;
                padding: 20px;
                background-color: #f5f5f5;
                color: #333;
            }
            h1, h2, h3 { color: #2c3e50; }
            .container {
                max-width: 1000px;
                margin: 0 auto;
                background-color: white;
                padding: 20px;
                border-radius: 8px;
                box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            }
            .info-box {
                background-color: #f8f9fa;
                border-radius: 5px;
                padding: 15px;
                margin-bottom: 20px;
                border-left: 4px solid #3498db;
            }
            .success { color: #27ae60; }
            .error { color: #e74c3c; }
            .warning { color: #f39c12; }
            .info { color: #3498db; }
            .file-list {
```

```
            background-color: #f9f9f9;
            border-radius: 5px;
            padding: 10px;
            border: 1px solid #ddd;
        }
        .file-item {
            display: flex;
            justify-content: space-between;
            padding: 5px 10px;
            border-bottom: 1px solid #eee;
        }
        .file-item:last-child {
            border-bottom: none;
        }
        .nav-links {
            display: flex;
            gap: 10px;
            margin-top: 20px;
        }
        .nav-link {
            display: inline-block;
            padding: 8px 16px;
            background-color: #3498db;
            color: white;
            text-decoration: none;
            border-radius: 4px;
            font-weight: bold;
            transition: background-color 0.3s;
        }
        .nav-link:hover {
            background-color: #2980b9;
        }
        .metrics {
            display: flex;
            gap: 10px;
            flex-wrap: wrap;
        }
        .metric-card {
            flex: 1;
            min-width: 120px;
            background-color: #fff;
            padding: 15px;
            border-radius: 8px;
            box-shadow: 0 2px 4px rgba(0,0,0,0.1);
            text-align: center;
        }
        .metric-value {
            font-size: 24px;
```

```
                font-weight: bold;
                margin: 10px 0;
                color: #3498db;
            }
            .metric-label {
                font-size: 14px;
                color: #7f8c8d;
            }
            .badge {
                display: inline-block;
                padding: 3px 8px;
                border-radius: 12px;
                font-size: 12px;
                font-weight: bold;
                color: white;
                background-color: #95a5a6;
            }
            .badge-primary { background-color: #3498db; }
            .badge-success { background-color: #27ae60; }
            .badge-warning { background-color: #f39c12; }
            .badge-danger { background-color: #e74c3c; }
        </style>
    </head>
    <body>
        <div class="container">
            <h1>{{ app_name }} <span class="badge badge-primary">v{{
app_version }}</span></h1>
            <p>A comprehensive Kubernetes demonstration application</p>

            <div class="info-box">
                <h2>Pod Information</h2>
                <p><strong>Instance ID:</strong> {{ instance_id }}</p>
                <p><strong>Hostname:</strong> {{ system_info.hostname }}</p>
                <p><strong>Environment:</strong> <span class="badge badge-
success">{{ environment }}</span></p>
                <p><strong>Request count:</strong> {{ request_count }}</p>
                <p><strong>Platform:</strong> {{ system_info.platform }}</p>
                <p><strong>Uptime:</strong> {{ system_info.uptime }}</p>
            </div>

            <div class="info-box">
                <h2>Resource Usage</h2>
                <div class="metrics">
                    <div class="metric-card">
                        <div class="metric-label">CPU Usage</div>
                        <div class="metric-value">{{
resource_usage.cpu_percent }}%</div>
                    </div>
```

```
                    <div class="metric-card">
                        <div class="metric-label">Memory</div>
                        <div class="metric-value">{{
resource_usage.memory_percent }}%</div>
                    </div>
                    <div class="metric-card">
                        <div class="metric-label">Disk</div>
                        <div class="metric-value">{{ resource_usage.disk_usage
}}</div>
                    </div>
                    <div class="metric-card">
                        <div class="metric-label">Requests</div>
                        <div class="metric-value">{{ metrics.requests }}</div>
                    </div>
                </div>
            </div>

            <div class="info-box">
                <h2>Mounted Volumes</h2>

                <h3>Data Volume</h3>
                <p><strong>Path:</strong> {{ volumes.data.path }}</p>
                <p><strong>Status:</strong>
                    {% if volumes.data.status == 'mounted' %}
                        <span class="success">Successfully mounted</span>
                    {% elif volumes.data.status == 'empty' %}
                        <span class="warning">Mounted but empty</span>
                    {% else %}
                        <span class="error">Error: {{ volumes.data.error
}}</span>
                    {% endif %}
                </p>

                {% if volumes.data.files %}
                <div class="file-list">
                    <h4>Files:</h4>
                    {% for file in volumes.data.files %}
                    <div class="file-item">
                        <span>{{ file }}</span>
                        <a href="/view-file?path={{ volumes.data.path }}/{{
file }}" class="nav-link">View</a>
                    </div>
                    {% endfor %}
                </div>
                {% endif %}

                <h3>Config Volume</h3>
                <p><strong>Path:</strong> {{ volumes.config.path }}</p>
```

```
                <p><strong>Status:</strong>
                    {% if volumes.config.status == 'mounted' %}
                        <span class="success">Successfully mounted</span>
                    {% elif volumes.config.status == 'empty' %}
                        <span class="warning">Mounted but empty</span>
                    {% else %}
                        <span class="error">Error: {{ volumes.config.error
}}</span>
                    {% endif %}
                </p>

                {% if volumes.config.files %}
                <div class="file-list">
                    <h4>Files:</h4>
                    {% for file in volumes.config.files %}
                    <div class="file-item">
                        <span>{{ file }}</span>
                        <a href="/view-file?path={{ volumes.config.path }}/{{
file }}" class="nav-link">View</a>
                    </div>
                    {% endfor %}
                </div>
                {% endif %}

                <h3>Logs Volume</h3>
                <p><strong>Path:</strong> {{ volumes.logs.path }}</p>
                <p><strong>Status:</strong>
                    {% if volumes.logs.status == 'mounted' %}
                        <span class="success">Successfully mounted</span>
                    {% elif volumes.logs.status == 'empty' %}
                        <span class="warning">Mounted but empty</span>
                    {% else %}
                        <span class="error">Error: {{ volumes.logs.error
}}</span>
                    {% endif %}
                </p>

                {% if volumes.logs.files %}
                <div class="file-list">
                    <h4>Files:</h4>
                    {% for file in volumes.logs.files %}
                    <div class="file-item">
                        <span>{{ file }}</span>
                        <a href="/view-file?path={{ volumes.logs.path }}/{{
file }}" class="nav-link">View</a>
                    </div>
                    {% endfor %}
                </div>
```

```
                {% endif %}
            </div>

            <div class="info-box">
                <h2>Actions</h2>
                <div class="nav-links">
                    <a href="/create-file" class="nav-link">Create a File</a>
                    <a href="/api/info" class="nav-link">API Info</a>
                    <a href="/api/health" class="nav-link">Health Check</a>
                    <a href="/api/metrics" class="nav-link">Metrics</a>
                </div>
            </div>

            <div class="info-box">
                <h2>Environment Variables</h2>
                <p><strong>APP_NAME:</strong> {{ app_name }}</p>
                <p><strong>APP_VERSION:</strong> {{ app_version }}</p>
                <p><strong>ENVIRONMENT:</strong> {{ environment }}</p>
                <p><strong>DATA_PATH:</strong> {{ data_path }}</p>
                <p><strong>CONFIG_PATH:</strong> {{ config_path }}</p>
                <p><strong>LOG_PATH:</strong> {{ log_path }}</p>
                <p><strong>SECRET_KEY:</strong> {{ secret_key|truncate(10,
True, '...') }}</p>
            </div>
        </div>
    </body>
    </html>
    """

    # Render the template with our data (render_template_string library helps
rendering HTML templates from string inputs)
    return render_template_string(
        html_content,
        app_name=APP_NAME,
        app_version=APP_VERSION,
        environment=ENVIRONMENT,
        instance_id=INSTANCE_ID,
        system_info=system_info,
        resource_usage=resource_usage,
        volumes=volumes,
        request_count=request_count,
        metrics=metrics,
        data_path=DATA_PATH,
        config_path=CONFIG_PATH,
        log_path=LOG_PATH,
        secret_key=SECRET_KEY
    )
```

```
@app.route('/view-file')
def view_file():
    """View the contents of a file from a mounted volume"""
    global metrics

    # Get the file path from the query parameters
    file_path = request.args.get('path', '')

    # Security check to prevent directory traversal attacks
    # Only allow access to our mounted volumes
    allowed_paths = [DATA_PATH, CONFIG_PATH, LOG_PATH]
    valid_path = False

    for path in allowed_paths:
        if file_path.startswith(path):
            valid_path = True
            break

    if not valid_path:
        metrics['errors'] += 1
        return "Access denied: Invalid path", 403

    # Try to read the file
    try:
        with open(file_path, 'r') as f:
            content = f.read()

        # Record the successful read
        metrics['data_reads'] += 1
        logger.info(f"File viewed: {file_path}")

        # Simple HTML to display the file content
        html = f"""
        <!DOCTYPE html>
        <html>
        <head>
            <title>File: {os.path.basename(file_path)}</title>
            <style>
                body {{ font-family: Arial, sans-serif; line-height: 1.6;
padding: 20px; }}
                pre {{ background-color: #f8f9fa; padding: 15px; border-
radius: 5px; overflow-x: auto; }}
                .nav-link {{
                    display: inline-block;
                    padding: 8px 16px;
                    background-color: #3498db;
                    color: white;
                    text-decoration: none;
```

```
                    border-radius: 4px;
                    font-weight: bold;
                }}
            </style>
        </head>
        <body>
            <h1>File: {os.path.basename(file_path)}</h1>
            <p>Path: {file_path}</p>
            <pre>{content}</pre>
            <a href="/" class="nav-link">Back to Home</a>
        </body>
        </html>
        """
        return html
    except Exception as e:
        metrics['errors'] += 1
        logger.error(f"Error viewing file {file_path}: {str(e)}")
        return f"Error reading file: {str(e)}", 500

@app.route('/create-file', methods=['GET', 'POST'])
def create_file():
    """Create a new file in the mounted data volume"""
    global metrics

    if request.method == 'POST':
        filename = request.form.get('filename', '')
        content = request.form.get('content', '')

        # Only allow creating files in the data directory
        file_path = os.path.join(DATA_PATH, filename)

        # For security, don't allow directory traversal
        #This security check prevents directory traversal attacks by ensuring
the file is created only inside the allowed DATA_PATH directory.
        if '..' in filename or '/' in filename:
            metrics['errors'] += 1
            return "Invalid filename. Directory traversal not allowed.", 400

        try:
            with open(file_path, 'w') as f:
                f.write(content)

            # Record the successful write
            metrics['data_writes'] += 1
            logger.info(f"File created: {file_path}")

            # Also write to the log volume to demonstrate multiple volume
mounting
```

```
            log_message = f"File created: {filename} at
{datetime.datetime.now().isoformat()}\n"
            with open(os.path.join(LOG_PATH, 'file_operations.log'), 'a') as
log:
                log.write(log_message)

            return redirect('/')

        except Exception as e:
            metrics['errors'] += 1
            logger.error(f"Error creating file {file_path}: {str(e)}")
            return f"Error creating file: {str(e)}", 500
    else:
        # Show form for creating a file
        html = """
        <!DOCTYPE html>
        <html>
        <head>
            <title>Create File</title>
            <style>
                body { font-family: Arial, sans-serif; line-height: 1.6;
padding: 20px; }
                .form-group { margin-bottom: 15px; }
                label { display: block; margin-bottom: 5px; }
                input[type="text"], textarea {
                    width: 100%;
                    padding: 8px;
                    border: 1px solid #ddd;
                    border-radius: 4px;
                }
                textarea { height: 200px; }
                button {
                    padding: 8px 16px;
                    background-color: #3498db;
                    color: white;
                    border: none;
                    border-radius: 4px;
                    cursor: pointer;
                    font-weight: bold;
                }
                .nav-link {
                    display: inline-block;
                    padding: 8px 16px;
                    background-color: #95a5a6;
                    color: white;
                    text-decoration: none;
                    border-radius: 4px;
                    font-weight: bold;
```

```
                }
            </style>
        </head>
        <body>
            <h1>Create a New File</h1>
            <p>This file will be saved to the mounted data volume.</p>

            <form method="post">
                <div class="form-group">
                    <label for="filename">Filename:</label>
                    <input type="text" id="filename" name="filename" required
placeholder="example.txt">
                </div>

                <div class="form-group">
                    <label for="content">Content:</label>
                    <textarea id="content" name="content" required
placeholder="Enter file content here..."></textarea>
                </div>

                <div class="form-group">
                    <button type="submit">Create File</button>
                    <a href="/" class="nav-link">Cancel</a>
                </div>
            </form>
        </body>
        </html>
        """
        return html

@app.route('/api/info')
def api_info():
    """API endpoint returning application information"""
    return jsonify({
        'app_name': APP_NAME,
        'version': APP_VERSION,
        'environment': ENVIRONMENT,
        'instance_id': INSTANCE_ID,
        'hostname': socket.gethostname(),
        'request_count': request_count,
        'uptime_seconds': time.time() - start_time,
# Kubernetes volumes allow persistent storage across container restarts.
# Volumes are mounted inside a container as directories.
# This lets the application store logs, config files, and data that persist
beyond container restarts.
#Volumes are defined in your Kubernetes YAML files (deployment.yaml or
volumes.yaml):
        'volumes': {
```

```
            'data': {
                'path': DATA_PATH,
                'mounted': os.path.exists(DATA_PATH) and os.access(DATA_PATH,
os.R_OK)
            },
#Checks if it is readable (os.access(PATH, os.R_OK)).
# Stores this information in a dictionary under the volumes key.

            'config': {
                'path': CONFIG_PATH,
                'mounted': os.path.exists(CONFIG_PATH) and
os.access(CONFIG_PATH, os.R_OK)
            },
            'logs': {
                'path': LOG_PATH,
                'mounted': os.path.exists(LOG_PATH) and os.access(LOG_PATH,
os.R_OK)
            }
        },
        'timestamp': datetime.datetime.now().isoformat()
    })

@app.route('/api/health')
def health_check():
    """Health check endpoint for Kubernetes liveness and readiness probes"""
    # Check if we can access our mounted volumes
    data_ok = os.path.exists(DATA_PATH) and os.access(DATA_PATH, os.R_OK)
    config_ok = os.path.exists(CONFIG_PATH) and os.access(CONFIG_PATH,
os.R_OK)
    logs_ok = os.path.exists(LOG_PATH) and os.access(LOG_PATH, os.W_OK)

    # For a real application, you might check database connections,
    # cache availability, etc.

    # Overall health status
    is_healthy = data_ok and config_ok and logs_ok

    # Log health check results
    logger.info(f"Health check: {'PASS' if is_healthy else 'FAIL'}")

    response = {
        'status': 'healthy' if is_healthy else 'unhealthy',
        'checks': {
            'data_volume': 'accessible' if data_ok else 'inaccessible',
            'config_volume': 'accessible' if config_ok else 'inaccessible',
            'logs_volume': 'writable' if logs_ok else 'not writable'
        },
        'timestamp': datetime.datetime.now().isoformat(),
```

```
        'hostname': socket.gethostname()
    }

    # Set the HTTP status code based on health
    status_code = 200 if is_healthy else 503

    return jsonify(response), status_code
@app.route('/api/metrics')
def get_metrics():
    """API endpoint for application metrics - useful for monitoring systems"""
    # Get basic resource usage stats
    cpu_percent = psutil.cpu_percent()
    memory_info = psutil.virtual_memory()
    disk_info = psutil.disk_usage('/')

    # Collect all metrics
    all_metrics = {
        'system': {
            'cpu_percent': cpu_percent,
            'memory_used_percent': memory_info.percent,
            'memory_used_mb': memory_info.used / (1024 * 1024),
            'memory_total_mb': memory_info.total / (1024 * 1024),
            'disk_used_percent': disk_info.percent,
            'disk_used_gb': disk_info.used / (1024**3),
            'disk_total_gb': disk_info.total / (1024**3)
        },
        'application': {
            'uptime_seconds': time.time() - start_time,
            'total_requests': metrics['requests'],
            'data_reads': metrics['data_reads'],
            'data_writes': metrics['data_writes'],
            'errors': metrics['errors']
        },
        'instance': {
            'id': INSTANCE_ID,
            'hostname': socket.gethostname()
        },
        'timestamp': datetime.datetime.now().isoformat()
    }
    # Log metrics collection for demonstration
    logger.debug(f"Metrics collected: CPU: {cpu_percent}%, Memory:
{memory_info.percent}%")

    return jsonify(all_metrics)
# For local testing - this won't run in Kubernetes
if _name_ == '_main_':
    print(f"Starting {APP_NAME} v{APP_VERSION} in {ENVIRONMENT} mode")
    app.run(host='0.0.0.0', port=5000, debug=True)
```