Docker demon
-

[Learning to read a script]

1️⃣ Defining Colors for Terminal Output

bash
CopyEdit

```
GREEN='\033[0;32m'   # Green color for success messages
BLUE='\033[0;34m'    # Blue for headings
YELLOW='\033[0;33m'  # Yellow for warnings
RED='\033[0;31m'     # Red for errors
CYAN='\033[0;36m'    # Cyan for informational messages
MAGENTA='\033[0;35m' # Magenta for step headings
NC='\033[0m'         # No color (resets text color)
```

- These **ANSI escape codes** change text color when printed in the terminal.
- Useful for **clear visibility** when running the script.

2️⃣ Displaying a Header Message

bash
CopyEdit

```
echo -e
"${BLUE}================================================================
========${NC}"
echo -e "${BLUE}             KUBERNETES ZERO TO HERO - REVISED SCRIPT
${NC}"
echo -e
"${BLUE}================================================================
========${NC}"
```

- **`echo -e`** enables interpretation of escape sequences (like `\n` for new lines).
- **`${BLUE}`** ensures the text appears in **blue**, and `${NC}` resets the color.
- This creates a **formatted heading** in the terminal.

3️⃣ Setting Up the Project Directory

bash
CopyEdit

```
echo -e "${MAGENTA}[STEP 1] SETTING UP PROJECT DIRECTORY
STRUCTURE${NC}"
```

- Prints **Step 1** in **magenta** for clarity.

bash
CopyEdit

```
PROJECT_DIR=~/k8s-master-app
echo -e "${CYAN}Creating project directory at ${PROJECT_DIR}...${NC}"
```

- Defines the **base directory** (`~/k8s-master-app`).

- Uses `${CYAN}` to make the message **stand out**.

4️⃣ Creating the Required Directory Structure

bash
CopyEdit

```
mkdir -p
${PROJECT_DIR}/{app,k8s/{base,volumes,networking,config,monitoring},sc
ripts,data,config,logs}
```

- **`mkdir -p`** ensures that directories are **created recursively** if they don't exist.
- **Breakdown of the structure:**
  - `app/` → Contains the Flask application.
  - `k8s/` → Stores Kubernetes-related YAML configurations.
    - `base/` → Core Kubernetes configurations.
    - `volumes/` → Persistent storage configurations.
    - `networking/` → Service and ingress configurations.
    - `config/` → ConfigMaps and secrets.
    - `monitoring/` → Prometheus, Grafana, or other monitoring setups.
  - `scripts/` → Holds helper scripts.
  - `data/` → Stores app-related data (mounted volumes).
  - `config/` → General configuration files.
  - `logs/` → Stores application logs.

5️⃣ Handling WSL2 Directory Mounting Issues

bash
CopyEdit

```
echo -e "${CYAN}Creating local data directories instead of host
mounts...${NC}"
```

- Since **WSL2 (Windows Subsystem for Linux 2)** has issues with mounting host directories into Kubernetes,
  - Instead of using **external mounts**, the script creates directories **inside the project**.

6️⃣ Creating Sample Configuration and Data Files

bash
CopyEdit

```
echo "This is a sample configuration file for our Kubernetes app" >
${PROJECT_DIR}/config/sample-config.txt
```

- Creates a **sample configuration file** inside `config/`.

bash
CopyEdit

```
echo "Hello from the volume!" > ${PROJECT_DIR}/data/hello.txt
```

- Creates a **test file** inside the `data/` directory.

bash

CopyEdit
```
echo "This file demonstrates volume mounting in Kubernetes" >
${PROJECT_DIR}/data/info.txt
```

- Another sample **text file** to test volume mounting.

bash
CopyEdit
```
echo -e "${GREEN}✓ Project directory structure created${NC}"
```

- Prints a **success message** in **green**.

7️⃣ Step 2: Creating the Flask Application

bash
CopyEdit
```
echo -e "${MAGENTA}[STEP 2] CREATING APPLICATION FILES${NC}"
echo -e "${CYAN}Building a Flask application that demonstrates volume
mounting...${NC}"
```

- Prints a heading in **magenta** and a message in **cyan**.

8️⃣ Creating the `app.py` File Using `cat`

bash
CopyEdit
```
cat > ${PROJECT_DIR}/app/app.py << 'EOL'
```

- **Writes content into `app.py`** dynamically.
- **`<< 'EOL'`** → Everything between `EOL` markers is **treated as input** and written to the file.

Summary

This script: ✅ **Creates a structured Kubernetes project folder**
✅ **Handles WSL2 limitations by using local directories**
✅ **Generates sample config and data files**
✅ **Prepares the environment for a Flask application**
Would you like a **detailed explanation** of the Flask app that follows this? 😃

Full Breakdown of This Code:

python
CopyEdit
```
# Import required libraries
from flask import Flask, jsonify, render_template_string, request,
redirect, url_for
import os  # Used to read environment variables
import socket  # Helps retrieve host-related information
import datetime  # For working with date and time
import json  # JSON formatting and responses
import logging  # Logging system for debugging and monitoring
```

```
import uuid  # Generates unique identifiers
import platform  # Gets system/platform information
import psutil  # Monitors system resource usage (CPU, memory, etc.)
import time  # Provides time-related functions
import threading  # Enables running background tasks
import sys  # System-specific parameters and functions
```

- **These libraries** help build a **Kubernetes-compatible** Flask app.
- They provide **logging, system monitoring, HTTP handling, and configuration management.**

Flask App Initialization

python
CopyEdit

```
# Initialize Flask application
app = Flask(__name__)
```

- **Creates a Flask web application** that serves HTTP requests.

Logging Configuration

python
CopyEdit

```
# Set up logging to print to console and file
logging.basicConfig(
    level=logging.INFO,  # Log messages with INFO level or higher
    format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',  #
Standard log format
    handlers=[
        logging.StreamHandler(sys.stdout),  # Print logs to console
(stdout)
        logging.FileHandler(os.environ.get('LOG_PATH',
'/app/app.log'))  # Save logs to a file
    ]
)
logger = logging.getLogger('k8s-master-app')
```

- **Logs are important in Kubernetes** to debug and monitor applications.
- This config:
  - Logs messages **both to the console and a file**.
  - The log file path is dynamically set using the **`LOG_PATH` environment variable** (defaults to `/app/app.log`).

Reading Configuration from Environment Variables

python
CopyEdit

```
APP_NAME = os.environ.get('APP_NAME', 'k8s-master-app')
```

- Reads the **`APP_NAME`** from environment variables.
- Defaults to `'k8s-master-app'` if not provided.

python
CopyEdit
```
APP_VERSION = os.environ.get('APP_VERSION', '1.0.0')
```

- Fetches the **app version**, defaulting to `'1.0.0'`.
- Helps in **tracking deployments** in Kubernetes.

python
CopyEdit
```
ENVIRONMENT = os.environ.get('ENVIRONMENT', 'development')
```

- Retrieves **deployment environment** (development, staging, production).
- Useful for **loading different configurations** based on the environment.

python
CopyEdit
```
DATA_PATH = os.environ.get('DATA_PATH', '/data')
```

- Defines where **app-related data** is stored.
- Kubernetes **volumes** will mount data at this location.

python
CopyEdit
```
CONFIG_PATH = os.environ.get('CONFIG_PATH', '/config')
```

- Defines the **configuration file path**.
- ConfigMaps in Kubernetes **inject configuration files** here.

python
CopyEdit
```
LOG_PATH = os.environ.get('LOG_PATH', '/logs')
```

- Specifies the **log storage directory**.
- Helps in **centralized logging** in Kubernetes.

Host & System Information

python
CopyEdit
```
HOSTNAME = socket.gethostname()
IP_ADDRESS = socket.gethostbyname(HOSTNAME)
SYSTEM_INFO = platform.system() + " " + platform.release()
```

- **HOSTNAME**: Gets the container's hostname inside Kubernetes.
- **IP_ADDRESS**: Fetches the container's internal IP address.
- **SYSTEM_INFO**: Retrieves the OS type and version (e.g., `"Linux 5.10.16"`).

Unique Instance ID

python
CopyEdit
```
INSTANCE_ID = str(uuid.uuid4())
```

- **Generates a unique ID** for each running instance.
- Useful for **tracking logs and debugging** across multiple replicas.

Flask Routes

Health Check Route

python
CopyEdit

```
@app.route('/health', methods=['GET'])
def health_check():
    return jsonify({'status': 'healthy', 'timestamp':
datetime.datetime.utcnow()}), 200
```

- Kubernetes **liveness & readiness probes** will **call this endpoint**.
- Returns `200 OK` if the app is running fine.

Root Route (Welcome Page)

python
CopyEdit

```
@app.route('/', methods=['GET'])
def home():
    return jsonify({
        'message': f'Welcome to {APP_NAME}!',
        'version': APP_VERSION,
        'instance_id': INSTANCE_ID,
        'hostname': HOSTNAME,
        'ip_address': IP_ADDRESS,
        'system_info': SYSTEM_INFO
    })
```

- Displays **app details, hostname, and IP** when you visit `/`.
- Helps debug **which Kubernetes pod is serving a request**.

Resource Usage API

python
CopyEdit

```
@app.route('/resources', methods=['GET'])
def system_resources():
    return jsonify({
        'cpu_usage': psutil.cpu_percent(),
        'memory_usage': psutil.virtual_memory().percent,
        'disk_usage': psutil.disk_usage(DATA_PATH).percent
    })
```

- Reports **CPU, Memory, and Disk Usage**.
- Useful for **monitoring pod resource consumption** in Kubernetes.

Background Log Writer

python
CopyEdit

```
def write_logs():
    while True:
        logger.info("Application is running...")
        time.sleep(30)  # Logs every 30 seconds
```

- Runs **in a separate thread** to continuously log messages.
- Helps in **monitoring pod activity**.

python
CopyEdit

```
threading.Thread(target=write_logs, daemon=True).start()
```

- **Starts the background log writer** when the app runs.

Running the Flask App

python
CopyEdit

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000)
```

- Runs the app on **port 5000**, listening on all interfaces (`0.0.0.0`).
- Kubernetes will **map this port to a service**.

Final Summary

This **Flask app**:

1. **Reads config from environment variables** → Makes it **dynamic** for Kubernetes.
2. **Logs messages to file & console** → Helps in **troubleshooting Kubernetes pods**.
3. **Has health check & monitoring endpoints** → Used by Kubernetes **probes**.
4. **Reports system resource usage** → Helps in **monitoring container performance**.
5. **Handles requests in a scalable way** → Kubernetes can run **multiple replicas**.

1️⃣ Tracking Request Count & Application Metrics

python
CopyEdit

```
request_count = 0
start_time = time.time()
```

- `request_count = 0` → **Tracks the total number of requests** received by the Flask app.
- `start_time = time.time()` → **Records the time when the application starts running** (in seconds since epoch).

python
CopyEdit

```
metrics = {
    'requests': 0,
    'errors': 0,
```

```
    'data_reads': 0,
    'data_writes': 0
}
```

- **Dictionary `metrics` stores key performance indicators (KPIs)** of the app:
  - `'requests'`: Total number of HTTP requests received.
  - `'errors'`: Number of errors encountered.
  - `'data_reads'`: Count of times data is read from a file or database.
  - `'data_writes'`: Count of times data is written to a file or database.

2️⃣ Defining a Background Worker Function

python
CopyEdit

```
def background_worker():
```

- Defines a function named `background_worker()`, which will run **continuously in the background** to simulate workload.

python
CopyEdit

```
    """
    Simulate background work to demonstrate resource usage.
    In a real app, this might be processing tasks, etc.
    """
```

- A **docstring** explaining that this function simulates **background activity**, such as:
  - Handling scheduled tasks.
  - Processing queued jobs.
  - Running periodic checks.

3️⃣ Logging That the Worker Has Started

python
CopyEdit

```
    logger.info("Background worker started")
```

- **Logs an informational message** indicating that the background worker has started.
- `logger.info()` writes messages to **both the console and the log file**.

4️⃣ Simulating CPU Load

python
CopyEdit

```
    counter = 0
    while True:
```

- `counter = 0` → Initializes a **counter** variable.
- `while True:` → Runs an **infinite loop**, ensuring the worker **never stops** unless the app is shut down.

python

CopyEdit

```
        # Simple CPU work - calculate prime numbers
         counter += 1
```

- **Increments `counter` on every loop iteration** to track worker progress.
- Comment suggests that **actual work** could be added, such as **prime number calculations**.

5️⃣ Logging Activity at Intervals

python
CopyEdit

```
        if counter % 1000 == 0:
             # Log occasionally to show activity
             logger.debug(f"Background worker tick: {counter}")
```

- **Every 1000 iterations**, it logs a debug message to show that the worker is active.
- `logger.debug()` logs the message at **debug level**, which is **less critical than `info` or `error`**.

6️⃣ Adding a Delay to Reduce CPU Usage

python
CopyEdit

```
        time.sleep(0.1)  # Don't use too much CPU
```

- `time.sleep(0.1)` → Pauses execution for **0.1 seconds (100ms)** to **reduce CPU load**.
- Without this, the loop would run **continuously at full speed**, consuming **100% CPU**.
- This **throttling technique** makes it more **realistic for background tasks**.

📌 Summary

This function runs **in the background** to: ✅ Simulate **workload**
✅ Track **resource usage**
✅ Log **progress periodically**
✅ Prevent CPU **overload** using `time.sleep()`
Would you like an **example of how to run this function in a separate thread?** 🚀

1️⃣ Starting the Background Worker Thread

python
CopyEdit

```
worker_thread = threading.Thread(target=background_worker,
daemon=True)
worker_thread.start()
```

- **Creates and starts a separate background thread** to run `background_worker()`.
- `daemon=True` → Marks it as a **daemon thread**, meaning:
  - It will **automatically stop** when the main process exits.
  - No need to manually terminate it.

2️⃣ Defining the Main Route (`/`)

python
CopyEdit

```
@app.route('/')
def index():
```

- **Defines the main route (`/`)** in the Flask app.
- When someone visits the homepage, this function will run.

python
CopyEdit

```
    """Main page showing application status and mounted volume
information"""
```

- A **docstring** explaining what this function does.

3️⃣ Updating Request Count & Logging Requests

python
CopyEdit

```
    global request_count, metrics
     request_count += 1
     metrics['requests'] += 1
```

- Uses the **global variables** `request_count` and `metrics`.
- **Increments the request count** for tracking total hits.

python
CopyEdit

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

- Logs the **IP address** of the visitor.

4️⃣ Getting System Information

python
CopyEdit

```
    system_info = {
         'hostname': socket.gethostname(),
         'platform': platform.platform(),
         'python_version': platform.python_version(),
         'cpu_count': psutil.cpu_count(),
         'memory': f"{psutil.virtual_memory().total / (1024 *
1024):.1f} MB",
         'uptime': f"{time.time() - start_time:.1f} seconds"
     }
```

- Gathers system details:
  - `'hostname'`: The **machine name** running this app.

- `'platform'`: The **OS and version** (e.g., `"Linux-5.15.0-86-generic"`).
- `'python_version'`: The **Python version**.
- `'cpu_count'`: The **number of CPU cores** available.
- `'memory'`: The **total system memory** in **MB**.
- `'uptime'`: **Time since the app started running**.

5️⃣ Getting Resource Usage

python
CopyEdit

```
resource_usage = {
    'cpu_percent': psutil.cpu_percent(),
    'memory_percent': psutil.virtual_memory().percent,
    'disk_usage': f"{psutil.disk_usage('/').percent}%"
}
```

- Uses the **`psutil` library** to get:
  - `'cpu_percent'`: **CPU usage percentage**.
  - `'memory_percent'`: **RAM usage percentage**.
  - `'disk_usage'`: **Disk space usage percentage**.

6️⃣ Checking Mounted Volumes

python
CopyEdit

```
volumes = {}
```

- Initializes an **empty dictionary** to store volume details.

Checking the `data` Volume

python
CopyEdit

```
try:
    data_files = os.listdir(DATA_PATH)
    volumes['data'] = {
        'path': DATA_PATH,
        'files': data_files,
        'status': 'mounted' if data_files else 'empty'
    }
    metrics['data_reads'] += 1
except Exception as e:
    volumes['data'] = {
        'path': DATA_PATH,
        'error': str(e),
        'status': 'error'
    }
    metrics['errors'] += 1
```

- Tries to **list the files** in `DATA_PATH`.
- If successful:
  - It stores the **path** and **file list** in `volumes['data']`.
  - If **files exist**, status is `'mounted'`; otherwise, `'empty'`.
  - Increments `metrics['data_reads']`.
- If an **error occurs**:
  - It logs the error message.
  - Increments `metrics['errors']`.

Checking the `config` Volume

python
CopyEdit

```
    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
```

- Same logic as the **data volume** but checks `CONFIG_PATH`.

Checking the `logs` Volume

python
CopyEdit

```
    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
```

- Same logic as **data & config volumes**, but for `LOG_PATH`.

📌 Summary

✔️ **Runs a background worker** in a separate thread.
✔️ **Handles requests** and logs visitor details.
✔️ **Collects system resource usage** (CPU, RAM, disk).
✔️ **Checks if volumes are properly mounted**.
✔️ **Handles errors gracefully** and tracks failures.

Background Worker Setup

python
CopyEdit

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

- This creates and starts a **background thread** that runs the `background_worker` function.
- The `daemon=True` makes sure the thread will stop when the main application exits.

Flask Route - `/` (Home Page)

python
CopyEdit

```
@app.route('/')
def index():
```

- This defines the main route (`/`) for the Flask web application.
- When a user visits the root URL, this function gets executed.

Tracking Request Count & Metrics

python
CopyEdit

```
    global request_count, metrics
     request_count += 1
     metrics['requests'] += 1
```

- Increments the `request_count` every time the endpoint is accessed.
- Updates the `metrics` dictionary, which stores various statistics.

Logging Incoming Requests

python
CopyEdit

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

- Logs the visitor's IP address (`request.remote_addr`).

- Helps in tracking traffic and debugging.

System Information Retrieval

python
CopyEdit

```
system_info = {
    'hostname': socket.gethostname(),
    'platform': platform.platform(),
    'python_version': platform.python_version(),
    'cpu_count': psutil.cpu_count(),
    'memory': f"{psutil.virtual_memory().total / (1024 *
1024):.1f} MB",
    'uptime': f"{time.time() - start_time:.1f} seconds"
}
```

- This dictionary gathers system-level details:
  - `hostname`: Name of the machine.
  - `platform`: OS details (e.g., Linux-5.15.0-91-generic).
  - `python_version`: Running Python version.
  - `cpu_count`: Number of CPU cores.
  - `memory`: Total system RAM.
  - `uptime`: Time since the application started.

Resource Usage (CPU, Memory, Disk)

python
CopyEdit

```
resource_usage = {
    'cpu_percent': psutil.cpu_percent(),
    'memory_percent': psutil.virtual_memory().percent,
    'disk_usage': f"{psutil.disk_usage('/').percent}%"
}
```

- Retrieves:
  - **CPU usage (%)** using `psutil.cpu_percent()`
  - **Memory usage (%)** using `psutil.virtual_memory().percent`
  - **Disk usage (%)** using `psutil.disk_usage('/')`
- Useful for monitoring **resource utilization**.

Checking Mounted Volumes

python
CopyEdit

```
volumes = {}
```

- Initializes a dictionary to store volume information.

Checking `data` volume

python
CopyEdit

```
    try:
        data_files = os.listdir(DATA_PATH)
        volumes['data'] = {
            'path': DATA_PATH,
            'files': data_files,
            'status': 'mounted' if data_files else 'empty'
        }
        metrics['data_reads'] += 1
     except Exception as e:
        volumes['data'] = {
            'path': DATA_PATH,
            'error': str(e),
            'status': 'error'
        }
        metrics['errors'] += 1
```

- **Tries** to list files in `DATA_PATH`:
  - If successful, it updates the `volumes` dictionary.
  - If there are **no files**, it marks the volume as **empty**.
  - If an **error occurs** (e.g., directory missing), it logs the error.

Checking `config` volume

python
CopyEdit

```
    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
```

- Same logic as above, but for **configuration files**.

Checking `logs` volume

python
CopyEdit

```
    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
```

- Same approach for **logs directory**.

Building the HTML Response

python
CopyEdit

```
   html_content = """
    <!DOCTYPE html>
    <html>
    <head>
        <title>{{ app_name }} - Kubernetes Master App</title>
```

- Starts an HTML response.
- Uses `{{ app_name }}` to insert **dynamic data**.

Basic Styling

html
CopyEdit

```
       <style>
            body {
                font-family: Arial, sans-serif;
                background-color: #f5f5f5;
            }
```

- Defines **CSS styles** to improve UI.

Displaying System Info

html
CopyEdit

```
           <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>
```

- Displays **app name, version, and system details** dynamically.

Displaying Resource Usage

html
CopyEdit

```
            <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>
```

- Shows CPU, memory, and disk **usage in real time**.

Displaying Mounted Volumes

html
CopyEdit

```
            <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>
```

- Checks if **volumes are successfully mounted** or if there are **errors**.

File List Display

html
CopyEdit

```
                {% 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 %}
```

- **Lists files** available in mounted directories.
- Provides **links to view** individual files.

Summary

This **Flask app** provides:

1. **System monitoring** (CPU, RAM, Disk).
2. **Volume checks** (data, config, logs).
3. **Request tracking** (counts every visit).
4. **A simple UI** to display everything.

Config Volume Display

html
CopyEdit

```
<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>
```

- Displays the **configuration volume** information.
- Checks if the volume is:
    - ✅ **Mounted successfully**
    - ⚠️ **Mounted but empty**
    - ❌ **Error (if any)**

If Configuration Files Exist

html
CopyEdit

```
{% 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 %}
```

- Lists **all config files** if available.
- Each file has a **"View"** link to open the file.

Logs Volume Display

html
CopyEdit

```
<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>
```

- Similar to **config volume**, but for **logs**.

If Log Files Exist

html
CopyEdit

```
{% 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 %}
```

- Lists **log files** if available.
- Each file has a **"View"** button.

Actions Section

html
CopyEdit

```
<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>
```

- **Provides quick access** to important routes:
  - Create a new file.
  - Get API information.
  - Perform a **health check**.
  - Fetch **metrics**.

Environment Variables Display

html
CopyEdit

```
<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>
```

- Displays **critical environment variables**.
- `SECRET_KEY` is **truncated for security** (only first 10 characters shown).

Rendering the Template

python
CopyEdit

```
# Render the template with our data
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
)
```

- **Dynamically injects data** into the HTML template.
- Uses `render_template_string()` to send the **generated HTML** as a response.

File Viewing Route (`/view-file`)

python
CopyEdit

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

- Defines a **Flask route** to allow users to **view file contents**.
- **Updates metrics** to track file accesses.

Extracting the File Path

python
CopyEdit

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

- Retrieves the **file path** from the URL.

Example:
bash

CopyEdit
`http://localhost:5000/view-file?path=/config/settings.json`

- would retrieve `/config/settings.json`.

1. Security Check for Directory Traversal

python
CopyEdit

```
# 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
```

- This initializes a list of allowed paths where files are stored (`DATA_PATH`, `CONFIG_PATH`, `LOG_PATH`).
- A `valid_path` flag is set to `False` to track whether the requested file path is valid.

python
CopyEdit

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

- Iterates over `allowed_paths` to check if the requested `file_path` starts with one of them.
- If a match is found, it sets `valid_path = True` and exits the loop.

python
CopyEdit

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

- If the `file_path` is outside allowed directories, it logs an error in `metrics` and returns a **403 Forbidden** response.

---

2. Reading and Displaying a File

python
CopyEdit

```
try:
    with open(file_path, 'r') as f:
        content = f.read()
```

- Opens the requested file in **read mode (`'r'`)** and reads its content into a variable.

python
CopyEdit

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

- Increments the `data_reads` counter in `metrics` to track successful file reads.
- Logs the file access event for auditing.

python
CopyEdit

```
# 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>
"""
```

- Creates an HTML page that:
  - Displays the **file name** and **path**.
  - Shows file content inside a `<pre>` block for proper formatting.
  - Provides a **Back to Home** button.

python
CopyEdit

```
return html
```

- Returns the generated HTML to the client.

python

CopyEdit

```
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
```

- If any error occurs while reading the file:
  - Increments the `errors` counter.
  - Logs the error message.
  - Returns a **500 Internal Server Error** response.

---

3. Creating a New File

python
CopyEdit

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

- Defines a Flask route `/create-file` that supports both **GET** and **POST** requests.
- Uses a docstring to describe the function.
- Declares `metrics` as `global` to track metrics.

python
CopyEdit

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

- Checks if the request is a **POST** request.
- Retrieves the **filename** and **content** from the form data.

python
CopyEdit

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

- Constructs the full file path by joining `DATA_PATH` with the `filename`.

python
CopyEdit

```
# For security, don't allow directory traversal
if '..' in filename or '/' in filename:
    metrics['errors'] += 1
    return "Invalid filename. Directory traversal not allowed.", 400
```

- Prevents **directory traversal attacks** (`../` or `/`) by rejecting such filenames.
- Returns a **400 Bad Request** error if detected.

python
CopyEdit

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

- Opens the file in **write mode (`'w'`)** and writes the content to it.

python
CopyEdit

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

- Increments the `data_writes` counter.
- Logs the file creation event.

python
CopyEdit

```
# 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)
```

- Appends a log entry to `file_operations.log` in the log volume (`LOG_PATH`).

python
CopyEdit

```
return redirect('/')
```

- Redirects the user **back to the home page** after file creation.

python
CopyEdit

```
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
```

- If an error occurs, logs it and returns a **500 Internal Server Error** response.

---

4. HTML Form for Creating a File

python
CopyEdit

```
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
```

- If the request is **GET**, it returns an HTML form to create a new file.
- Includes:
  - **Filename input** (`<input type="text">`)
  - **Content textarea** (`<textarea>`)
  - **Submit button** and **Cancel button**

Summary

- Implements **security measures** (directory traversal prevention).
- Supports **viewing and creating files** via Flask routes.
- Uses **logging and metrics tracking**.
- Serves **HTML responses dynamically**.

API Info Endpoint (`/api/info`)

Function Definition & Docstring

python
CopyEdit

```
@app.route('/api/info')
def api_info():
    """API endpoint returning application information"""
```

- Registers the endpoint `/api/info` with the Flask app.
- Defines a function `api_info()` that returns application metadata.
- The docstring explains that this endpoint returns application info.

Returning JSON Response

python
CopyEdit

```
return jsonify({
```

- The function returns a JSON response using `jsonify()`.

Application Metadata

python
CopyEdit

```
    'app_name': APP_NAME,
    'version': APP_VERSION,
```

```
    'environment': ENVIRONMENT,
    'instance_id': INSTANCE_ID,
```

- Includes application-specific metadata (name, version, environment, and instance ID).

Host and Instance Info

python
CopyEdit

```
    'hostname': socket.gethostname(),
    'request_count': request_count,
    'uptime_seconds': time.time() - start_time,
```

- `socket.gethostname()`: Gets the machine's hostname.
- `request_count`: Tracks the number of requests handled.
- `uptime_seconds`: Computes the server uptime since `start_time`.

Mounted Volumes Check

python
CopyEdit

```
    'volumes': {
        'data': {
            'path': DATA_PATH,
            'mounted': os.path.exists(DATA_PATH) and
os.access(DATA_PATH, os.R_OK)
        },
        '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)
        }
    },
```

- Defines a dictionary for mounted volumes:
  - Checks if each path (`DATA_PATH`, `CONFIG_PATH`, `LOG_PATH`) exists and is readable (`os.R_OK`).
  - Returns a boolean indicating if the volume is accessible.

Timestamp

python
CopyEdit

```
    'timestamp': datetime.datetime.now().isoformat()
})
```

- Includes the current timestamp in ISO format.

Health Check Endpoint (`/api/health`)

Function Definition & Docstring

python
CopyEdit

```
@app.route('/api/health')
def health_check():
    """Health check endpoint for Kubernetes liveness and readiness
probes"""
```

- Registers the `/api/health` route.
- Defines `health_check()`, which is used by Kubernetes for liveness/readiness probes.

Checking Volume Accessibility

python
CopyEdit

```
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)
```

- Checks if:
  - `DATA_PATH` and `CONFIG_PATH` exist and are **readable**.
  - `LOG_PATH` exists and is **writable**.

Setting Health Status

python
CopyEdit

```
is_healthy = data_ok and config_ok and logs_ok
```

- Determines overall health based on volume access checks.

Logging the Health Check Result

python
CopyEdit

```
logger.info(f"Health check: {'PASS' if is_healthy else 'FAIL'}")
```

- Logs whether the health check **passed** or **failed**.

Constructing the Response

python
CopyEdit

```
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()
}
```

- Sets the health status to `"healthy"` or `"unhealthy"`.
- Provides detailed status for each volume.
- Includes the current timestamp and hostname.

Setting HTTP Status Code

python
CopyEdit

```
status_code = 200 if is_healthy else 503
```

- Returns **`200 OK`** if healthy.
- Returns **`503 Service Unavailable`** if unhealthy.

Returning JSON Response

python
CopyEdit

```
return jsonify(response), status_code
```

- Sends the health check response with the appropriate HTTP status code.

API Metrics Endpoint (`/api/metrics`)

Function Definition & Docstring

python
CopyEdit

```
@app.route('/api/metrics')
def get_metrics():
    """API endpoint for application metrics - useful for monitoring
systems"""
```

- Registers the `/api/metrics` route.
- Defines `get_metrics()`, which provides system and application performance metrics.
- The docstring explains that this is useful for monitoring.

Fetching System Resource Usage

python
CopyEdit

```
cpu_percent = psutil.cpu_percent()
memory_info = psutil.virtual_memory()
disk_info = psutil.disk_usage('/')
```

- **`psutil.cpu_percent()`**: Retrieves CPU usage as a percentage.
- **`psutil.virtual_memory()`**: Fetches memory details.
- **`psutil.disk_usage('/')`**: Retrieves disk usage for the root (`/`) partition.

Structuring the Metrics Response

python
CopyEdit

```
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)
        },
```

- **System metrics:**
  - CPU usage (`cpu_percent`).
  - Memory usage (percentage, used memory in MB, total memory in MB).
  - Disk usage (percentage, used disk in GB, total disk in GB).

python
CopyEdit

```
    'application': {
            'uptime_seconds': time.time() - start_time,
            'total_requests': metrics['requests'],
            'data_reads': metrics['data_reads'],
            'data_writes': metrics['data_writes'],
            'errors': metrics['errors']
        },
```

- **Application metrics:**
  - `uptime_seconds`: Server uptime.
  - `total_requests`: Tracks total HTTP requests.
  - `data_reads`, `data_writes`: Track file operations.
  - `errors`: Counts application errors.

python
CopyEdit

```
    'instance': {
            'id': INSTANCE_ID,
            'hostname': socket.gethostname()
        },
        'timestamp': datetime.datetime.now().isoformat()
}
```

- **Instance metadata:**
  - `id`: Unique instance ID.
  - `hostname`: Machine's hostname.
- **Timestamp:** Current time in ISO format.

Logging Metrics Collection

python
CopyEdit

```
logger.debug(f"Metrics collected: CPU: {cpu_percent}%, Memory:
{memory_info.percent}%")
```

- Logs collected CPU and memory usage for debugging.

Returning JSON Response

python
CopyEdit
```
return jsonify(all_metrics)
```

- Returns the structured metrics as a JSON response.

Running the Flask App (Local Testing)

python
CopyEdit
```
# 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)
```

- If the script is executed directly, it:
  - Prints startup information.
  - Runs Flask on **`0.0.0.0:5000`** with **debug mode enabled**.
- **Note:** This is for local development and won't be used in Kubernetes.

Summary

1. **`/api/metrics`**
   - Returns system stats (CPU, memory, disk usage).
   - Includes application metrics (uptime, requests, errors).
   - Provides instance metadata (hostname, ID).
2. **Local Testing (`__main__`)**
   - Runs the Flask app locally on port **5000**.

Step 1: Creating `requirements.txt`

bash
CopyEdit
```
cat > ${PROJECT_DIR}/app/requirements.txt << 'EOL'
Flask==2.2.3
Werkzeug==2.2.3
psutil==5.9.5
EOL
```

- Creates a `requirements.txt` file inside `${PROJECT_DIR}/app/` with necessary dependencies:
  - **Flask** (for the web framework)
  - **Werkzeug** (for request handling)
  - **psutil** (for system resource monitoring)

Step 2: Creating the Dockerfile

bash
CopyEdit
```
cat > ${PROJECT_DIR}/app/Dockerfile << 'EOL'
```

- Writes the Dockerfile to `${PROJECT_DIR}/app/Dockerfile`.

Base Image
dockerfile
CopyEdit
```
FROM python:3.9
```

- Uses **Python 3.9** as the base image (non-slim version for pre-installed utilities).

Metadata Labels
dockerfile
CopyEdit
```
LABEL maintainer="k8s-zero-hero@example.com"
LABEL version="1.0.0"
LABEL description="Kubernetes Zero to Hero Master Application"
```

- Adds metadata such as **maintainer, version, and description**.

Working Directory & Volumes
dockerfile
CopyEdit
```
WORKDIR /app

RUN mkdir -p /data /config /logs && \
    chmod 777 /data /config /logs
```

- Sets `/app` as the working directory.
- Creates mount points for **data, config, and logs**, with **read/write access**.

Install Dependencies
dockerfile
CopyEdit
```
COPY requirements.txt .
RUN pip install --no-cache-dir --upgrade pip && \
    pip install --no-cache-dir -r requirements.txt
```

- Copies `requirements.txt` and installs dependencies **without caching**.

Copy Application Code
dockerfile
CopyEdit
```
COPY app.py .
RUN chmod +x app.py
```

- Copies `app.py` into the container and makes it **executable**.

Expose Application Port
dockerfile
CopyEdit
```
EXPOSE 5000
```

- Opens **port 5000** for external access.

Health Check Script
dockerfile

CopyEdit
```
RUN echo '#!/bin/sh' > /healthcheck.sh && \
    echo 'curl -s http://localhost:5000/api/health || exit 1' >>
/healthcheck.sh && \
    chmod +x /healthcheck.sh
```

- Creates a shell script to **check container health** via `/api/health`.

Security: Running as Non-Root User
dockerfile
CopyEdit
```
RUN useradd -m appuser && \
    chown -R appuser:appuser /app /data /config /logs

USER appuser
```

- **Creates a non-root user (`appuser`)** and assigns ownership to avoid running as root.

Set Environment Variables
dockerfile
CopyEdit
```
ENV PYTHONDONTWRITEBYTECODE=1 \
    PYTHONUNBUFFERED=1 \
    APP_NAME="K8s Master App" \
    APP_VERSION="1.0.0" \
    ENVIRONMENT="production" \
    DATA_PATH="/data" \
    CONFIG_PATH="/config" \
    LOG_PATH="/logs"
```

- Disables Python bytecode writing and buffering.
- Sets **application metadata and paths**.

Container Startup Command
dockerfile
CopyEdit
```
CMD ["python", "app.py"]
```

- Starts the Flask application.

Health Check Instruction
dockerfile
CopyEdit
```
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3
CMD /healthcheck.sh
```

- Runs **every 30s** with a **3s timeout**.
- Starts after **5s** and retries **3 times**.

Step 3: Creating Kubernetes Manifest Files

bash

CopyEdit

```
echo -e "${MAGENTA}[STEP 3] CREATING KUBERNETES MANIFESTS${NC}"
echo -e "${CYAN}Creating Kubernetes configuration files with helpful
analogies...${NC}"
```

- Displays messages to indicate Kubernetes manifest creation.

Create Namespace (`namespace.yaml`)

bash

CopyEdit

```
cat > ${PROJECT_DIR}/k8s/base/namespace.yaml << 'EOL'
```

- Creates a **namespace** to isolate resources.

yaml

CopyEdit

```
apiVersion: v1
kind: Namespace
metadata:
  name: k8s-demo
  labels:
    name: k8s-demo
    environment: demo
    app: k8s-master
```

- Namespace is **`k8s-demo`**, labeled for easy identification.

Create ConfigMap (`configmap.yaml`)

bash

CopyEdit

```
cat > ${PROJECT_DIR}/k8s/config/configmap.yaml << 'EOL'
```

- Stores **non-sensitive configuration** separately.

yaml

CopyEdit

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-config
  namespace: k8s-demo
data:
  APP_NAME: "Kubernetes Zero to Hero"
  APP_VERSION: "1.0.0"
  ENVIRONMENT: "demo"
  DATA_PATH: "/data"
  CONFIG_PATH: "/config"
  LOG_PATH: "/logs"
  app-settings.json: |
    {
```

```
    "logLevel": "info",
    "enableMetrics": true,
    "maxFileSizeMB": 10
  }
```

- Stores environment variables and an embedded JSON config file.

Summary

✅ **Creates `requirements.txt`** for dependencies.
✅ **Builds Dockerfile** for a **secure and efficient** container.
✅ **Defines Kubernetes manifests** for **namespace isolation and configuration management**.

Step 4: Creating Kubernetes Secrets

bash
CopyEdit

```
cat > ${PROJECT_DIR}/k8s/config/secret.yaml << 'EOL'
```

- Creates a Kubernetes **Secret** named `app-secrets` inside the **`k8s-demo`** namespace.

Secret.yaml

yaml
CopyEdit

```
apiVersion: v1
kind: Secret
metadata:
  name: app-secrets
  namespace: k8s-demo
type: Opaque
data:
  # Values must be base64 encoded
  # echo -n "dev-secret-key-12345" | base64
  SECRET_KEY: ZGV2LXNlY3JldC1rZXktMTIzNDU=
  # echo -n "password123" | base64
  DB_PASSWORD: cGFzc3dvcmQxMjM=
```

🔹 **How Secrets Work:**

- **Type:** `Opaque` → Generic secret storage.
- **Values:** Must be **base64 encoded**.

Example encoding:

bash
CopyEdit

```
echo -n "dev-secret-key-12345" | base64
```

  - Output: `ZGV2LXNlY3JldC1rZXktMTIzNDU=`
  - Kubernetes will decode these when injected into pods.

🔹 **Why Use Secrets?**

- Prevents sensitive values from being **hardcoded**.
- Unlike `ConfigMaps`, secrets are **not easily visible** in logs/UI.

- **In production**: Use a **secrets manager** like **HashiCorp Vault, AWS Secrets Manager, or Kubernetes External Secrets**.

Step 5: Setting Up Persistent Volumes (Using `emptyDir`)

bash
CopyEdit

```
cat > ${PROJECT_DIR}/k8s/volumes/volumes.yaml << 'EOL'
```

- Defines **temporary storage volumes** for the app.

ConfigMap-based Volume (`volumes.yaml`)

yaml
CopyEdit

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: app-files
  namespace: k8s-demo
data:
  hello.txt: |
    Hello from the Kubernetes volume!
    This file is loaded from a ConfigMap, not a host mount.

  info.txt: |
    This file demonstrates how to use ConfigMaps to provide files to
pods.
    In a real application, you might use PersistentVolumes backed by
cloud storage.

  sample-config.txt: |
    # Sample Configuration
    log_level=info
    max_connections=100
    timeout=30
```

🔹 **Why `emptyDir` Instead of `hostPath`?**

- `emptyDir` creates a **temporary volume** that:
  - Exists **only as long as the pod is running**.
  - Is **wiped** when the pod is deleted.
- **Avoids WSL2 issues:** `hostPath` (mounting local directories) causes problems in WSL2.

🔹 **How to Use This Volume in a Pod?**

yaml
CopyEdit

```
volumes:
  - name: app-files-volume
```

```
  configMap:
    name: app-files
```

- This mounts the **ConfigMap** as a volume so that pods can access its files.

🔹 Summary

✅ **`secret.yaml`** → Stores **sensitive credentials** securely in base64.
✅ **`volumes.yaml`** → Uses **ConfigMap-based storage** instead of `hostPath`, preventing WSL2 issues.
✅ **Uses `emptyDir` volumes** → Ideal for temporary storage inside Kubernetes.

1️⃣ Metadata: Defining the Deployment

yaml
CopyEdit

```
apiVersion: apps/v1
kind: Deployment
```

- `apiVersion: apps/v1` → Specifies that we are using the **apps/v1 API** for **Deployments**.
- `kind: Deployment` → Defines that this is a **Deployment** resource.

yaml
CopyEdit

```
metadata:
  name: k8s-master-app
  namespace: k8s-demo
  labels:
    app: k8s-master
```

- `name: k8s-master-app` → Names the **Deployment** as `"k8s-master-app"`.
- `namespace: k8s-demo` → Places the deployment in the `"k8s-demo"` namespace.
- `labels: app: k8s-master` → Tags it with a label for **identification**.

2️⃣ Spec: Defining the Deployment Behavior

yaml
CopyEdit

```
spec:
  replicas: 2
```

- `replicas: 2` → Maintains **2 identical pods** to ensure availability.

Deployment Strategy

yaml
CopyEdit

```
 strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 1
```

- `type: RollingUpdate` → Updates pods **one at a time** for zero-downtime.
- `maxUnavailable: 1` → **Max 1 pod can be down** during an update.
- `maxSurge: 1` → **1 extra pod** can be created while updating.

Selector: Matching Pods

yaml
CopyEdit

```
 selector:
    matchLabels:
      app: k8s-master
```

- Ensures this **Deployment only manages** pods with `app: k8s-master`.

3️⃣ Pod Template: How Pods Should Look

yaml
CopyEdit

```
 template:
    metadata:
      labels:
        app: k8s-master
```

- Labels the pod with `app: k8s-master`.

Annotations (For Prometheus Monitoring)

yaml
CopyEdit

```
     annotations:
        prometheus.io/scrape: "true"
        prometheus.io/path: "/api/metrics"
        prometheus.io/port: "5000"
```

- **Prometheus Monitoring Annotations**:
  - `prometheus.io/scrape: "true"` → Enables Prometheus to **collect metrics**.
  - `prometheus.io/path: "/api/metrics"` → Exposes metrics on `/api/metrics`.
  - `prometheus.io/port: "5000"` → Metrics are available on **port 5000**.

4️⃣ Pod Specifications

yaml
CopyEdit

```
   spec:
      containers:
      - name: k8s-master-app
        image: k8s-master-app:latest
        imagePullPolicy: Never
```

- `name: k8s-master-app` → Name of the container.

- `image: k8s-master-app:latest` → Uses `k8s-master-app:latest` image.
- `imagePullPolicy: Never` → **Prevents pulling from a registry** (useful for local development).

Exposing Ports

yaml
CopyEdit

```
ports:
 - containerPort: 5000
   name: http
```

- `containerPort: 5000` → Exposes port **5000** inside the container.
- `name: http` → Labels this port as `"http"`.

---

5 Environment Variables

yaml
CopyEdit

```
envFrom:
 - configMapRef:
     name: app-config
```

- Loads environment variables **from a ConfigMap** named `"app-config"`.

yaml
CopyEdit

```
- secretRef:
     name: app-secrets
```

- Loads **sensitive data** (like passwords) from `"app-secrets"` Secret.

---

6 Storage: Mounting Volumes

yaml
CopyEdit

```
volumeMounts:
 - name: data-volume
   mountPath: /data
   readOnly: false
 - name: config-volume
   mountPath: /config
   readOnly: true
 - name: logs-volume
   mountPath: /logs
   readOnly: false
 - name: config-files
   mountPath: /config-files
   readOnly: true
```

- Mounts the following:

- `data-volume` → **Writable**, stores dynamic app data.
- `config-volume` → **Read-only**, holds config files.
- `logs-volume` → **Writable**, stores logs.
- `config-files` → **Read-only**, contains sample config files.

7️⃣ Resource Management

yaml
CopyEdit

```yaml
resources:
  requests:
    cpu: "100m"
    memory: "128Mi"
  limits:
    cpu: "500m"
    memory: "512Mi"
```

- Ensures **efficient scheduling**:
  - Requests **100m CPU (0.1 core)**, **128MB memory**.
  - Limits to **500m CPU (0.5 core)**, **512MB memory**.

8️⃣ Health Checks

Liveness Probe (Restart if Unresponsive)

yaml
CopyEdit

```yaml
livenessProbe:
  httpGet:
    path: /api/health
    port: http
  initialDelaySeconds: 30
  periodSeconds: 30
  timeoutSeconds: 5
  failureThreshold: 3
```

- **Checks if the app is alive**:
  - Calls `/api/health` on **port 5000**.
  - Waits **30 seconds** before the first check.
  - Runs **every 30 seconds**.
  - **If it fails 3 times, the container restarts**.

Readiness Probe (Check Before Accepting Traffic)

yaml
CopyEdit

```yaml
readinessProbe:
  httpGet:
    path: /api/health
    port: http
  initialDelaySeconds: 15
  periodSeconds: 10
```

```
          timeoutSeconds: 3
```

- **Checks if the app is ready to serve traffic**:
  - Calls `/api/health` every **10 seconds**.
  - Starts checking after **15 seconds**.

9️⃣ Lifecycle Hooks (Logging Events)

yaml
CopyEdit

```
       lifecycle:
          postStart:
            exec:
              command: ["/bin/sh", "-c", "echo 'Container started' >
/logs/container.log"]
          preStop:
            exec:
              command: ["/bin/sh", "-c", "echo 'Container stopping' >>
/logs/container.log"]
```

- **Logs when the container starts/stops**:
  - `postStart`: Writes `"Container started"` to `/logs/container.log`.
  - `preStop`: Writes `"Container stopping"` before shutdown.

🔟 Volumes (Storage Configuration)

yaml
CopyEdit

```
      volumes:
       - name: data-volume
         emptyDir: {}
       - name: config-volume
         emptyDir: {}
       - name: logs-volume
         emptyDir: {}
       - name: config-files
         configMap:
           name: app-files
```

- Uses `emptyDir` instead of **PersistentVolumes** to avoid WSL2 issues.

🔟 Init Containers (Pre-setup)

yaml
CopyEdit

```
      initContainers:
       - name: init-volumes
         image: busybox
```

```
        command: ["/bin/sh", "-c", "cp /config-files/* /data/ && echo
'Volumes initialized' > /logs/init.log"]
        volumeMounts:
        - name: data-volume
          mountPath: /data
        - name: logs-volume
          mountPath: /logs
        - name: config-files
          mountPath: /config-files
```

- **Prepares volumes before app starts**:
  - Copies sample files from `config-files` to `/data`.
  - Writes `"Volumes initialized"` to `/logs/init.log`.

✅ Summary

- Uses **RollingUpdate** for safe deployment.
- Uses **emptyDir** instead of `PersistentVolumes` (Fixes WSL2 issues).
- Uses **Init Container** to prepare data.
- Adds **health checks** for auto-restarts if the app crashes.
- Logs startup and shutdown events.

1️⃣ Metadata

yaml
CopyEdit

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
```

- `apiVersion: autoscaling/v2` → Uses **autoscaling API version v2** (latest stable).
- `kind: HorizontalPodAutoscaler` → Defines this as an **HPA resource**.

yaml
CopyEdit

```
metadata:
  name: k8s-master-hpa
  namespace: k8s-demo
```

- `name: k8s-master-hpa` → The **name** of the HPA.
- `namespace: k8s-demo` → The **namespace** where the HPA applies.

2️⃣ Target Deployment

yaml
CopyEdit

```
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: k8s-master-app
```

- `scaleTargetRef` → Specifies **which deployment** this HPA controls.
- **It auto-scales the `k8s-master-app` deployment**.

3️⃣ Scaling Limits

yaml
CopyEdit

```
 minReplicas: 1
  maxReplicas: 5
```

- **`minReplicas: 1`** → The deployment **will always have at least 1 pod**.
- **`maxReplicas: 5`** → **Scales up to 5 pods max** when load increases.

4️⃣ Scaling Metrics

📌 CPU-Based Scaling

yaml
CopyEdit

```
 - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 50
```

- **If CPU usage exceeds 50% per pod**, **more pods** will be created.
- If CPU usage drops **below 50%**, extra pods will be removed.

📌 Memory-Based Scaling

yaml
CopyEdit

```
 - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 50
```

- **If memory usage exceeds 50% per pod**, more pods will be **created**.
- If memory usage drops **below 50%**, extra pods will be **removed**.

5️⃣ Scaling Behavior (Prevent Rapid Scaling)

yaml
CopyEdit

```
 behavior:
    scaleUp:
      stabilizationWindowSeconds: 60
    scaleDown:
      stabilizationWindowSeconds: 120
```

- `scaleUp.stabilizationWindowSeconds: 60` → Waits **60s before scaling up** (prevents rapid scaling up).
- `scaleDown.stabilizationWindowSeconds: 120` → Waits **120s before scaling down** (prevents rapid downscaling).

✅ Summary

- **Auto-scales** `k8s-master-app` between **1 to 5 pods**.
- Scales **based on CPU & memory** (`50%` utilization threshold).
- **Prevents unnecessary scaling** by setting stabilization windows.

🔍 Add Logging to a File for Debugging

Store logs in a file (`deploy.log`) for easier debugging:

bash

CopyEdit

```
exec > >(tee deploy.log) 2>&1
```

- Add this line at the beginning after `#!/bin/bash`.

🚀 Verify Kubernetes Namespace Before Creating It

Instead of blindly applying `namespace.yaml`, check if the namespace exists:

bash

CopyEdit

```
if ! kubectl get namespace k8s-demo &> /dev/null; then
    echo -e "${CYAN}Creating namespace...${NC}"
    kubectl apply -f k8s/base/namespace.yaml
else
    echo -e "${GREEN}✓ Namespace already exists${NC}"
fi
```

-

🔄 Improve Docker Build Process with Caching

Use `--progress=plain` to debug builds and optimize caching:

bash

CopyEdit

```
docker build --network=host --progress=plain -t k8s-master-app:latest
.
```

-

📡 Verify Minikube Status Before Restarting

Instead of checking only `host: Running`, check for all components:

bash

CopyEdit

```
minikube_status=$(minikube status | grep -E "host: Running|kubelet:
Running|apiserver: Running" | wc -l)
if [ "$minikube_status" -lt 3 ]; then
    echo -e "${YELLOW}Minikube is not fully running.
Restarting...${NC}"
    minikube start --cpus=2 --memory=4096 --disk-size=20g
--driver=docker
else
    echo -e "${GREEN}✓ Minikube is fully running${NC}"
```

```
fi
```

-

🔄 **Retry Mechanism for `kubectl apply`**
Avoid transient failures in Kubernetes resource creation:
bash
CopyEdit

```
retry_kubectl() {
    local resource=$1
    local max_attempts=3
    for attempt in $(seq 1 $max_attempts); do
        kubectl apply -f $resource && return 0
        echo -e "${YELLOW}Retrying $resource (Attempt
$attempt/$max_attempts)...${NC}"
        sleep 5
    done
    echo -e "${RED}Failed to apply $resource after $max_attempts
attempts.${NC}"
    return 1
}

retry_kubectl "k8s/base/deployment.yaml"
retry_kubectl "k8s/networking/service.yaml"
```

-

✨ Key Benefits of These Changes:

✔️ **Better Debugging** → Logs stored in `deploy.log`
✔️ **More Robust Minikube Checks** → Ensures all components are running
✔️ **Efficient Docker Builds** → Uses build caching
✔️ **Resilient Kubernetes Deployments** → Adds retry mechanism

tep 7: Wait for Deployment to be Ready

bash
CopyEdit

```
echo -e "${MAGENTA}[STEP 7] WAITING FOR DEPLOYMENT TO BE READY${NC}"
echo -e "${CYAN}This may take a minute or two...${NC}"
```

- Prints a message indicating that the script is waiting for the Kubernetes deployment to be ready.
- Uses ANSI escape codes (`${MAGENTA}` and `${CYAN}`) for colored text.

bash
CopyEdit

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

- Uses `kubectl rollout status` to check the status of the deployment `k8s-master-app` in the `k8s-demo` namespace.

- The `--timeout=180s` flag ensures it waits up to 180 seconds before timing out.

bash
CopyEdit
```
if [ $? -ne 0 ]; then
```

- Checks the exit status (`$?`) of the previous command.
- If it's **non-zero**, it means the deployment failed.

bash
CopyEdit
```
   echo -e "${RED}Deployment failed to become ready within the timeout period.${NC}"
```

- Prints an error message in **red** if the deployment is not ready.

bash
CopyEdit
```
   echo -e "${YELLOW}Checking pod status...${NC}"
    kubectl -n k8s-demo get pods
```

- Prints a message indicating that the script is checking the status of pods.
- Runs `kubectl get pods` to list all pods in the `k8s-demo` namespace.

bash
CopyEdit
```
   echo -e "${YELLOW}Checking pod logs...${NC}"
    POD=$(kubectl -n k8s-demo get pods -l app=k8s-master -o name | head -1)
```

- Tries to get the name of a pod associated with `app=k8s-master` by filtering the pods using the label selector (`-l app=k8s-master`).
- `-o name` ensures it only returns the pod names.
- `head -1` selects the first pod in the list.

bash
CopyEdit
```
   if [ ! -z "$POD" ]; then
        kubectl -n k8s-demo logs $POD
    fi
```

- If a pod name was found (`! -z "$POD"` ensures it's not empty), it fetches the logs of that pod using `kubectl logs`.

bash
CopyEdit
```
else
    echo -e "${GREEN}✓ Deployment is ready${NC}"
fi
```

- If the deployment was successful, it prints a **success message** in **green**.

Step 8: Set Up Port Forwarding

bash
CopyEdit
```
echo -e "${MAGENTA}[STEP 8] SETTING UP PORT FORWARDING${NC}"
echo -e "${CYAN}This will make the application accessible on
localhost${NC}"
```

- Prints a message indicating the script is setting up port forwarding.

bash
CopyEdit
```
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
```

- Checks if a `kubectl port-forward` process is already running.
- If found, it **kills** the process to avoid conflicts.

bash
CopyEdit
```
kubectl -n k8s-demo port-forward svc/k8s-master-app 8080:80 &
PORT_FORWARD_PID=$!
```

- Starts **port forwarding** from the `k8s-master-app` service **inside the cluster** (port `80`) to **localhost** (`8080`).
- Runs the process **in the background** (`&`).
- Stores the **process ID (PID)** in `PORT_FORWARD_PID`.

bash
CopyEdit
```
sleep 2
```

- Waits **2 seconds** to allow the process to start.

bash
CopyEdit
```
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
```

- Checks if the port-forwarding process is running (`ps -p $PORT_FORWARD_PID`).
- Prints **error** if it failed, otherwise prints **success message**.

Step 9: Deployment Completion Message

bash
CopyEdit
```
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}"
```

- Prints a **final success message**.

Step 10: Display Access Information

bash
CopyEdit

```
echo -e "${YELLOW}Your application is accessible via multiple
methods:${NC}"
```

- Prints information on how to access the deployed application.

1. Port Forwarding

bash
CopyEdit

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

- Displays **port forwarding** access.

2. NodePort Access

bash
CopyEdit

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

- Gets the **Minikube IP** and displays the **NodePort access URL**.

3. Minikube Service URL

bash
CopyEdit

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

- Provides an alternative way to access the app using `minikube service`.

Step 11: Display Useful Kubernetes Commands

bash
CopyEdit

```
echo -e
"${BLUE}================================================================
========${NC}"
echo -e "${YELLOW}USEFUL COMMANDS:${NC}"
echo -e
"${BLUE}================================================================
========${NC}"
```

- Prints a section with **useful commands**.

View Kubernetes Dashboard

bash

CopyEdit

```
echo -e "${CYAN}View the Kubernetes Dashboard:${NC}"
echo "   minikube dashboard"
```

- Shows how to open the **Kubernetes dashboard**.

View Application Logs

bash

CopyEdit

```
echo -e "${CYAN}View application logs:${NC}"
echo "   kubectl -n k8s-demo logs -l app=k8s-master"
```

- Command to view logs for all pods matching `app=k8s-master`.

Get a Shell into a Pod

bash

CopyEdit

```
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"
```

- Provides a command to open a **shell** in the first available pod.

View All Resources in the Namespace

bash

CopyEdit

```
echo -e "${CYAN}View all resources in the namespace:${NC}"
echo "   kubectl -n k8s-demo get all"
```

- Displays all **Kubernetes resources** in `k8s-demo`.

Check Pod Resource Usage

bash

CopyEdit

```
echo -e "${CYAN}Check pod resource usage (if metrics-server is
enabled):${NC}"
echo "   kubectl -n k8s-demo top pods"
```

- Shows **CPU & memory usage** of pods (requires `metrics-server`).

Clean Up All Resources

bash
CopyEdit
```
echo -e "${CYAN}Clean up all resources:${NC}"
echo "   ./scripts/cleanup.sh"
```

- Provides command to run the **cleanup script**.

Stop Port Forwarding
bash
CopyEdit
```
echo -e "${CYAN}Stop port forwarding:${NC}"
echo "   kill $PORT_FORWARD_PID"
```

- Command to **stop port forwarding**.

Final Deployment Success Message
bash
CopyEdit
```
echo -e
"${BLUE}============================================================
========${NC}"
echo -e "${GREEN}DEPLOYMENT SUCCESSFUL!${NC}"
echo -e
"${BLUE}============================================================
========${NC}"
echo -e "${YELLOW}Enjoy exploring your Kubernetes application!${NC}"
```

- Final **success message**.

Summary
This script: ✅ Deploys an application
✅ Waits for readiness
✅ Sets up **port forwarding**
✅ Provides multiple access options
✅ Displays useful Kubernetes commands

Section 1: API Testing Function
bash
CopyEdit
```
test_api() {
    echo -e "${CYAN}Running API tests...${NC}"
```

- Defines a function called `test_api` to test the application's API endpoints.
- `echo -e` prints a message to indicate that API tests are starting. The `${CYAN}` and `${NC}` are color codes for formatting text.

bash
CopyEdit
```
   run_test_request "/" "Homepage"
    run_test_request "/api/health" "Health check endpoint"
```

```
    run_test_request "/api/info" "Info endpoint"
    run_test_request "/api/metrics" "Metrics endpoint"
```

- Calls `run_test_request` (assumed to be a defined function elsewhere) to send HTTP requests to different API endpoints:
  - `/` → Tests the homepage.
  - `/api/health` → Tests if the health check API is functioning.
  - `/api/info` → Retrieves application information.
  - `/api/metrics` → Fetches monitoring metrics.

bash
CopyEdit

```
   echo -e "${CYAN}API tests completed${NC}"
}
```

- Prints a message indicating API tests are completed.

Section 2: Getting Kubernetes Pod Info

bash
CopyEdit

```
get_pod_info() {
    echo -e "${CYAN}Getting pod information...${NC}"
```

- Defines a function `get_pod_info` that retrieves information about Kubernetes pods.

bash
CopyEdit

```
   kubectl -n k8s-demo get pods -l app=k8s-master
```

- Runs a `kubectl` command to list all pods in the `k8s-demo` namespace that match the label `app=k8s-master`.

bash
CopyEdit

```
   POD_NAME=$(kubectl -n k8s-demo get pods -l app=k8s-master -o name |
head -1)
```

- Stores the name of the first pod (from the filtered list) into the `POD_NAME` variable.

bash
CopyEdit

```
   if [ ! -z "$POD_NAME" ]; then
```

- Checks if `POD_NAME` is not empty (i.e., if a pod exists).

bash
CopyEdit

```
       echo -e "${YELLOW}Detailed info for $POD_NAME:${NC}"
        kubectl -n k8s-demo describe $POD_NAME
    fi
}
```

- If a pod is found, prints its name and runs `kubectl describe` to fetch detailed information.

Section 3: Running Tests and Displaying Final Messages

bash
CopyEdit

```
test_api
get_pod_info
```

- Calls the `test_api` and `get_pod_info` functions to run API tests and fetch pod info.

bash
CopyEdit

```
echo -e
"${BLUE}================================================================
========${NC}"
echo -e "${GREEN}TESTS COMPLETE!${NC}"
echo -e
"${BLUE}================================================================
========${NC}"
echo -e "${YELLOW}Your Kubernetes application is up and running.${NC}"
```

- Prints a message indicating that tests are complete and the Kubernetes application is running.

Section 4: Making Scripts Executable

bash
CopyEdit

```
chmod +x ${PROJECT_DIR}/scripts/deploy.sh
chmod +x ${PROJECT_DIR}/scripts/cleanup.sh
chmod +x ${PROJECT_DIR}/scripts/test-app.sh
```

- Changes file permissions to make the deployment, cleanup, and testing scripts executable.

bash
CopyEdit

```
echo -e "${GREEN}✓ Deployment scripts created${NC}"
```

- Prints a success message confirming the scripts have been set up.

Section 5: Creating Documentation

bash
CopyEdit

```
echo -e "${MAGENTA}[STEP 5] CREATING DOCUMENTATION${NC}"
echo -e "${CYAN}Creating README and documentation files...${NC}"
```

- Prints messages indicating that documentation is being created.

Creating `README.md`

bash
CopyEdit

```
cat > ${PROJECT_DIR}/README.md << 'EOL'
# Kubernetes Zero to Hero Application (WSL2-Compatible Version)
```

- Creates a `README.md` file for project documentation.
- Explains Kubernetes concepts like:
  - ConfigMaps & Secrets
  - EmptyDir volumes (avoiding WSL2 mounting issues)
  - Networking methods
  - Health checks and resource management

Creating `kubernetes-cheatsheet.md`

bash
CopyEdit

```
cat > ${PROJECT_DIR}/kubernetes-cheatsheet.md << 'EOL'
# Kubernetes Commands Cheat Sheet
```

- Creates a cheat sheet with common `kubectl` commands for managing namespaces, resources, logs, debugging, and Minikube operations.

Section 6: Finalizing Setup

bash
CopyEdit

```
chmod +x ${PROJECT_DIR}/scripts/deploy.sh
chmod +x ${PROJECT_DIR}/scripts/cleanup.sh
chmod +x ${PROJECT_DIR}/scripts/test-app.sh
```

- Ensures all scripts are executable.

bash
CopyEdit

```
echo -e
"${BLUE}============================================================
========${NC}"
echo -e "${GREEN}KUBERNETES ZERO TO HERO PROJECT CREATED
SUCCESSFULLY!${NC}"
echo -e
"${BLUE}============================================================
========${NC}"
```

- Prints a message confirming the successful creation of the Kubernetes project.

bash
CopyEdit

```
echo -e "${CYAN}To deploy the application, run:${NC}"
echo -e "cd ${PROJECT_DIR}"
echo -e "./scripts/deploy.sh"
```

- Instructions for deploying the application.

bash
CopyEdit
```
echo -e "${CYAN}To clean up after you're done, run:${NC}"
echo -e "./scripts/cleanup.sh"
```

- Instructions for cleaning up resources.

bash
CopyEdit
```
echo -e "${CYAN}To test the application functionality, run:${NC}"
echo -e "./scripts/test-app.sh"
```

- Instructions for testing the application.

Key Takeaways

- The script sets up, deploys, and tests a Kubernetes application with WSL2 compatibility.
- It avoids `hostPath` volume mounting (a common WSL2 issue).
- It ensures smooth deployment using `EmptyDir` volumes and ConfigMaps.
- Comprehensive documentation and cheat sheets are generated.
- The project includes debugging utilities and best practices for Kubernetes.

Namespace is also called a smaller cluster to run bigger cluster
Anything you do is stored in config map
Adv of config map:
Persistent volume clain:
Hpa .yaml- max

```
class Dog:
    """
    A simple class representing a dog.

    This demonstrates the most basic class definition in Python.
    """
    # Class attribute - shared by all instances
    species = "Canis familiaris"
    # The _init_ method is a special method called when an object is created
    # It's similar to a constructor in other programming languages
    def _init_(self, name, age):
        """Initialize a new Dog object.

        Args:
            name (str): The dog's name
```

```
        age (int): The dog's age in years
        """
        # Instance attributes - unique to each instance
        self.name = name  # 'self' refers to the instance being created
        self.age = age
    # Instance method - defines behavior
    def bark(self):
        """The dog makes a sound."""
        return f"{self.name} says Woof!"
    # Another instance method
    def get_info(self):
        """Return a string with the dog's information."""
        return f"{self.name} is {self.age} years old."

# Creating objects (instances) of the Dog class
fido = Dog("Fido", 3)
bella = Dog("Bella", 5)

# Accessing attributes
print(fido.name)       # Output: Fido
print(bella.age)       # Output: 5
print(fido.species)    # Output: Canis familiaris

# Calling methods
print(fido.bark())     # Output: Fido says Woof!
print(bella.get_info()) # Output: Bella is 5 years old
```