Docker Persistent Logging & Jupyter Lab Exercises

Exercise 1: Persistent Data Logging

The goal is to run a container that generates log files and ensures those logs are not lost when the container is removed.

1. Launch a Container with a Data Volume:
Start an Alpine Linux container. While launching, mount a directory from your host machine called log_storage to the /var/log directory inside the container. Provide a screenshot of the docker run command you used.

2. Create a Logging Script:
Inside the running container, create a script at /usr/local/bin/data_logger.sh. This script should loop indefinitely, writing a new line with the current date and a "system status OK" message to /var/log/app.log every two seconds. Show the contents of your script with the cat command and provide a screenshot.

3. Run in Background:
Execute the data_logger.sh script in the background. Then, detach from the container's interactive session without stopping it. Capture a screenshot of your terminal showing the commands used to achieve this.

4. Verify on Host:
On your local machine (the host), use the tail -f command to monitor the app.log file within your log_storage directory. This will prove that the data written inside the container is persisting on your host. Provide a screenshot of the tail command's output showing the log entries appearing in real-time.

Exercise 2: Running Jupyter Lab in a Container

This task involves running a Jupyter Lab environment in a Docker container and making it accessible from your local machine.

1. Find an Image:
Search Docker Hub for an image such as jupyter/base-notebook or jupyter/scipy-notebook.

2. Run with Port Forwarding:
Run a container from the Jupyter image. Map port 8888 inside the container to port 9090 on your host machine. The command structure for port mapping is:
$> docker run -p 9090:8888 jupyter/base-notebook

3. Access Jupyter Lab:
Once the container is running, copy the URL with the token from the terminal output. Then open your web browser and navigate to http://localhost:9090/?token= to access Jupyter Lab.

4. Verify Operation:
Create a new notebook, run a simple Python command (like print("Hello, Docker!")), and provide a screenshot of your browser showing Jupyter Lab running successfully.

Exercise 3: Configurable Logging via Environment Variables

Here, you'll modify the script from Exercise 1 to make its behavior configurable when the container starts.

1. Parameterize the Script:
Rewrite your data_logger.sh script. Instead of a fixed message, it should now read a message prefix from an environment variable called LOG_PREFIX. The new log line should look like: [DATE]: [LOG_PREFIX] - system status OK.

2. Launch with an Environment Variable:
Run your container, this time setting the LOG_PREFIX environment variable to WebApp1. The syntax is:
$> docker run --env =

3. Verify Output:
Check the app.log file on your host machine to confirm that the new log lines correctly include the "WebApp1" prefix.

Exercise 4: Building a Robust Script

This task focuses on making your script more reliable by handling missing configuration.

1. Add a Default Value:
Modify data_logger.sh again. If the LOG_PREFIX environment variable is not set when the container runs, the script should use a default value of GENERIC.

2. Test the Fallback:
Prove that your modification works. First, run the container without setting the LOG_PREFIX variable and show that the log file contains the "GENERIC" prefix. Then, run it again with an invalid value (e.g., --env LOG_PREFIX="") and show that the default still applies.

Exercise 5: Creating a Custom Docker Image

The final exercise is to package your script into a new, portable Docker image.

1. Create a Dockerfile:
In a new directory, create a file named Dockerfile. This file should contain instructions to:
- Start from the alpine base image.
- Copy your final, robust data_logger.sh script from your local machine into the image's /usr/local/bin/ directory.
- Make the script executable.
- Set data_logger.sh as the default command to run when a container is started from this image.

2. Build the Image:
Use the docker build command to create a new image from your Dockerfile. Tag this image as custom-logger:latest.

3. Run Your Custom Image:
Launch a container from your new custom-logger image. Verify that it automatically starts logging to a mounted volume as intended. Provide a screenshot of your Dockerfile's contents and the docker run command used to test your new image.