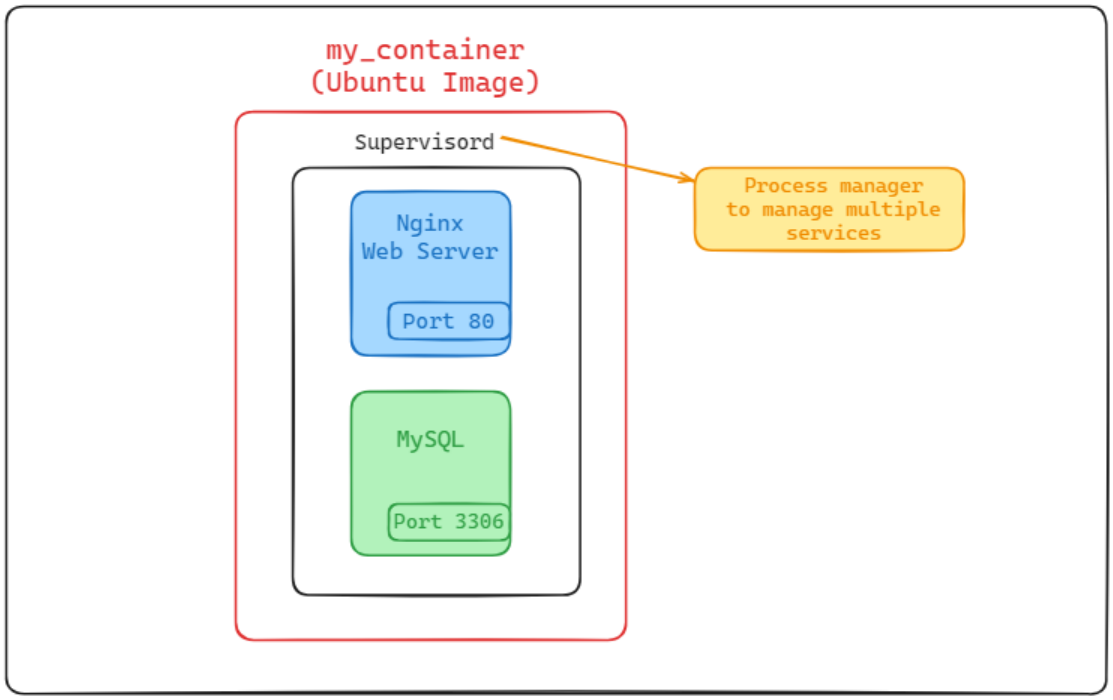
Setting Up a Host-Like Environment Using Docker Containers

This documentation provides a step-by-step guide to create a Docker container that mimics a traditional host environment. The container will be capable of running multiple processes and services using supervisord as the process manager.

Features

- Nginx: Web server.
- MySQL: Database server.
- Supervisord: Process manager to manage multiple services.



Host-Like Environment Setup

1. Create Dockerfile

Create a Dockerfile with the following content:

[Docker file code here, include ase ai folder]

Explanation:

- **Base Image:** ubuntu:latest provides a minimal Ubuntu environment.
- **Environment Variables:** DEBIAN_FRONTEND=noninteractive ensures package installations don't prompt for user input.
- **Package Installation:** Installs nginx, mysql-server, and supervisor, followed by cleanup to reduce image size.
- **Configuration:** Copies a custom supervisord configuration file into the container.
- **Ports:** Exposes necessary ports for Nginx and MySQL.
- **Entrypoint:** Uses supervisord to manage services within the container.

2. Create supervisord Configuration

Create a supervisord.conf file with the following content:

[supervisord]

nodaemon=true

[program:nginx]

command=/usr/sbin/nginx -g "daemon off;"

autorestart=true

[program:mysql]

command=/usr/sbin/mysqld

autorestart=true

Explanation:

- **supervisord:** Runs in the foreground (nodaemon=true).
- **Nginx:** Configured to run in the foreground (daemon off;) and restart automatically if it fails.
- **MySQL:** Configured to restart automatically if it fails.

3. Build the Docker Image

Build the Docker image using the Dockerfile and supervisord configuration:

```
docker build -t my_host_like_env .
```

4. Run the Docker Container

Run the Docker container based on the image you just built:

```
docker run -d --name my_container -p 80:80 -p 3306:3306 my_host_like_env
```

Check the container is running using:

```
docker ps
```

The output will look something similar like this:

```
CONTAINER ID   IMAGE          COMMAND                  CREATED         STATUS         PORTS
                NAMES
30af6e1bfab8   my_host_like_env   "/usr/bin/supervisord"   9 seconds ago   Up 6 seconds
0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:3306->3306/tcp, :
::3306->3306/tcp   my_container
```

docker exec -it my_container /bin/bash

This will open a new terminal inside the container. You can now verify that Nginx and MySQL process running using:

ps -ef

The ps -ef command is commonly used for system monitoring and troubleshooting to check which processes are running, their resource usage, and the commands used to start them.

```
term@ubuntu-p5sbbg-759d89bb87-nn9m4:~$ docker exec -it my_container /bin/bash
root@30af6e1bfab8:/# ps -ef
UID          PID    PPID  C STIME TTY          TIME CMD
root           1       0  0 22:56 ?        00:00:00 /usr/bin/python3 /usr/bin/supervisord
mysql          7       1  3 22:56 ?        00:00:01 /usr/sbin/mysqld
root           8       1  0 22:56 ?        00:00:00 nginx: master process /usr/sbin/nginx -g daemon off;
www-data       9       8  0 22:56 ?        00:00:00 nginx: worker process
www-data      10       8  0 22:56 ?        00:00:00 nginx: worker process
www-data      11       8  0 22:56 ?        00:00:00 nginx: worker process
www-data      12       8  0 22:56 ?        00:00:00 nginx: worker process
www-data      13       8  0 22:56 ?        00:00:00 nginx: worker process
```

Exit the container using exit command.

5. Access Services

Accessing Nginx Web Server

Access Nginx via curl or wget:

curl http://localhost:80

If Nginx is running correctly, this command will fetch the default Nginx welcome page.

```
term@ubuntu-p5sbbg-759d89bb87-nn9m4:~$ curl http://localhost:80
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
```

Access Nginx logs:

docker exec -it my_container tail -f /var/log/nginx/access.log

This command will show the Nginx access log. You can replace access.log with error.log for error logs.

```
term@ubuntu-p5sbbg-759d89bb87-nn9m4:~$ docker exec -it my_container tail -f /var/log/nginx/access.log
172.17.0.1 - - [03/Jun/2024:23:09:46 +0000] "GET / HTTP/1.1" 200 615 "-" "curl/7.81.0"
```

Accessing MySQL Database Server:

Access MySQL via MySQL Client:

Connect to MySQL server running inside the container

docker exec -it my_container mysql -uroot -p

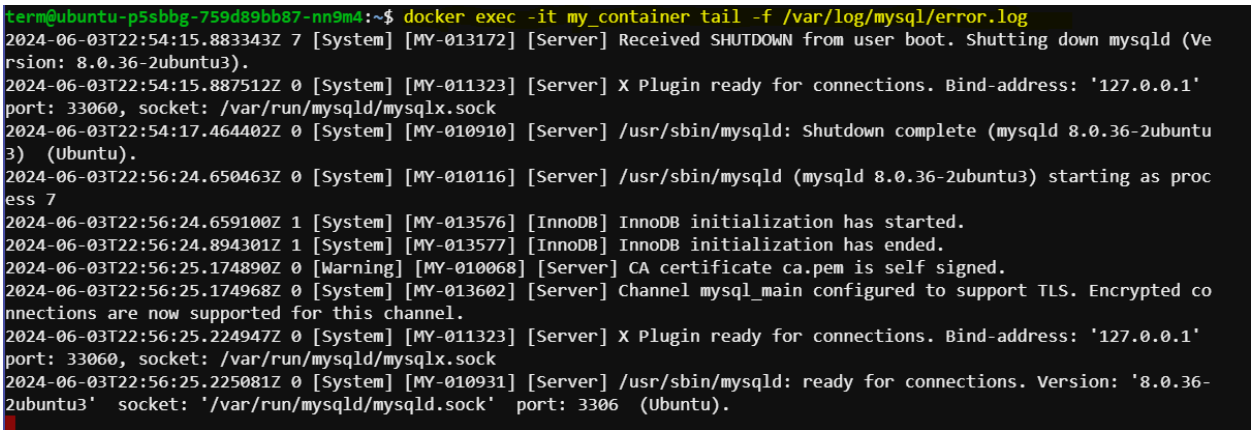
If prompted for a password, the default password is empty (just press Enter). Once connected, you can interact with MySQL as you would on a standard MySQL server.

```
term@ubuntu-p5sbbg-759d89bb87-nn9m4:~$ docker exec -it my_container mysql -uroot -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 8
Server version: 8.0.36-2ubuntu3 (Ubuntu)

Copyright (c) 2000, 2024, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.01 sec)
```

Access MySQL logs:

docker exec -it my_container tail -f /var/log/mysql/error.log

This command will show the MySQL error log. Replace error.log with query.log for the query log, if configured.



Usage

Managing Services

Inside the running container, you can manage services using supervisorctl:

Check status of services

supervisorctl status

Stop a service

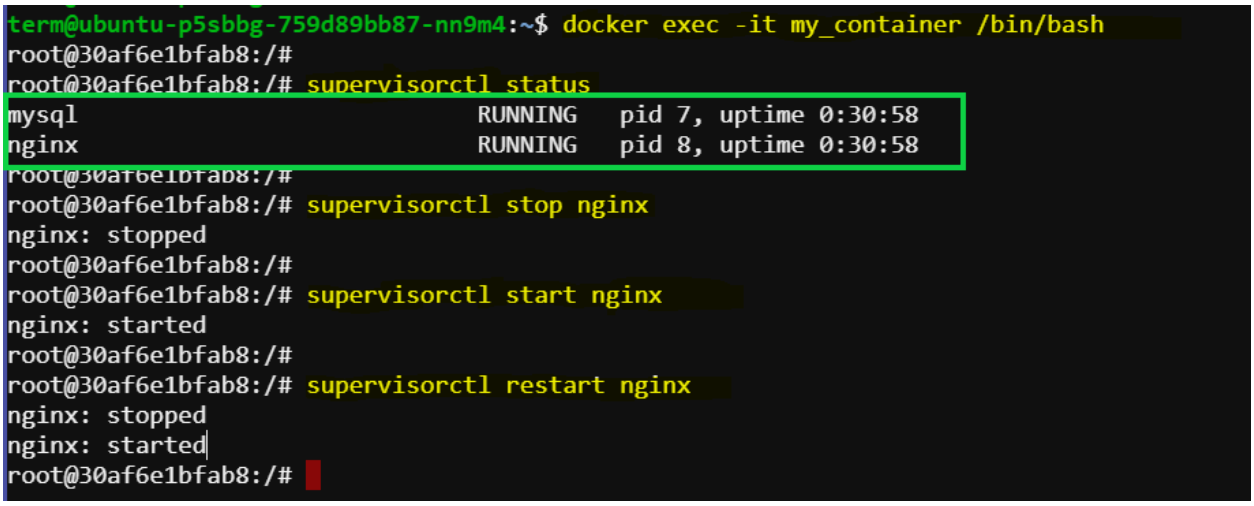
supervisorctl stop nginx

Start a service

supervisorctl start nginx

Restart a service

supervisorctl restart nginx



Conclusion

You now have a Docker container that simulates a traditional host environment capable of running multiple processes and services. You can extend this setup by adding more services or customizing configurations as per your requirements.