What is Docker Compose ?

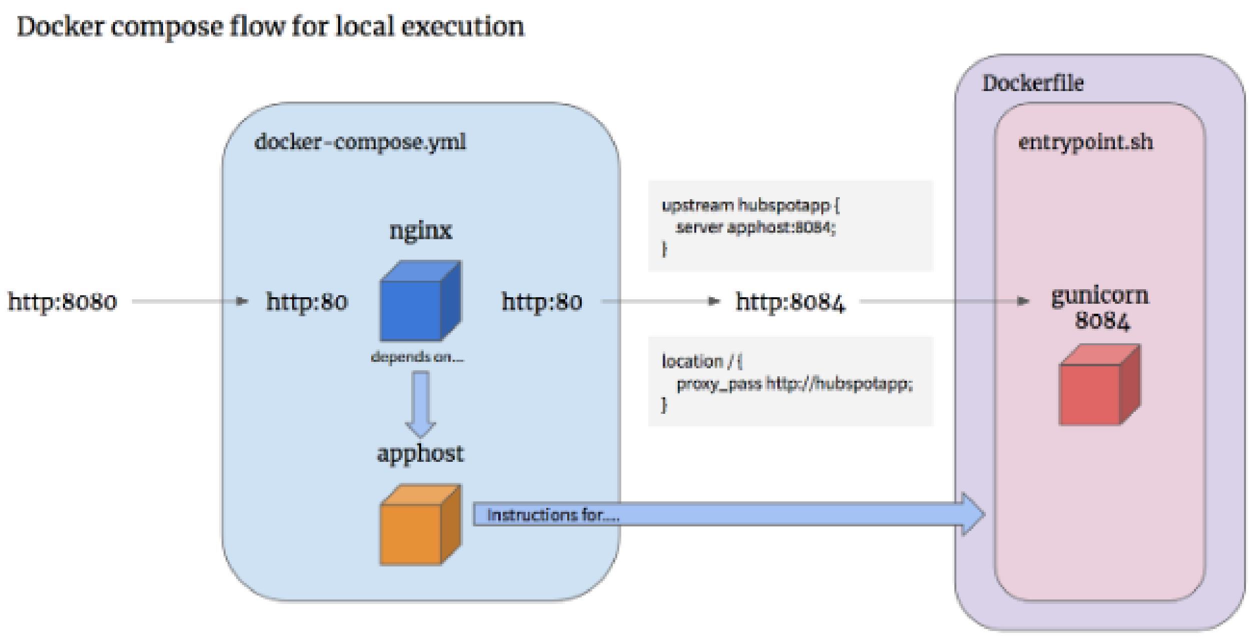
Docker Compose is a tool that allows you to define and **run multi-container Docker applications**. It uses a single YAML file (usually named docker-compose.yml) to specify how your application's containers, networks, and volumes should be configured and connected.

For example,

suppose you had an application which required **NGNIX and MySQL**, you could create one file which would start both the containers as a service without the need to start each one separately.

Why Use Docker Compose?

1. **Simplicity**: Docker Compose simplifies the process of setting up and running complex applications by defining all the components in a single file.
2. **Automation**: It automates the container creation and management process, reducing the need for manual setup and configuration.
3. **Isolation**: Containers in a Docker Compose application are isolated from the host system and each other, ensuring that applications run consistently across different environments.
4. **Reproducibility**: With a single configuration file, you can recreate the exact same environment on different machines, which makes development and testing more reliable.
5. **Scalability**: Docker Compose can define services that can be scaled up or down based on demand.

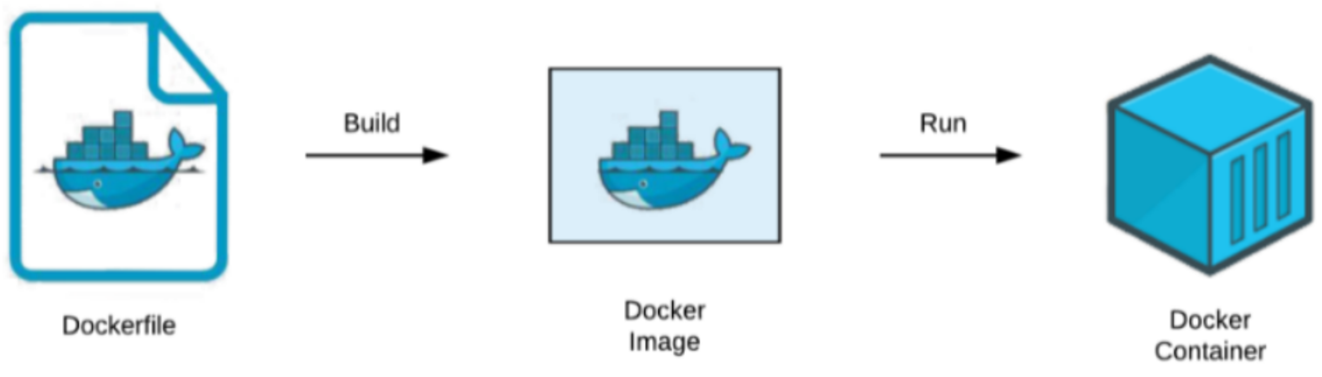


```
upstream hubspotapp {
   server apphost:8084;
}
```

```
location / {
   proxy_pass http://hubspotapp;
}
```

Basic Concepts:

1. **Services**: A service is a container defined in the `docker-compose.yml` file. It represents one component of your application, such as a web server, database, or application server.
2. **Containers**: These are instances of Docker images running based on the specifications provided in the Compose file. Each service can have one or more containers running.
3. **Volumes**: Volumes allow data to persist even if containers are stopped or removed. They can be used to share data between containers or with the host.
4. **Networks**: Docker Compose creates a network for the services in the Compose file, enabling communication between containers using service names as hostnames.



Commands:

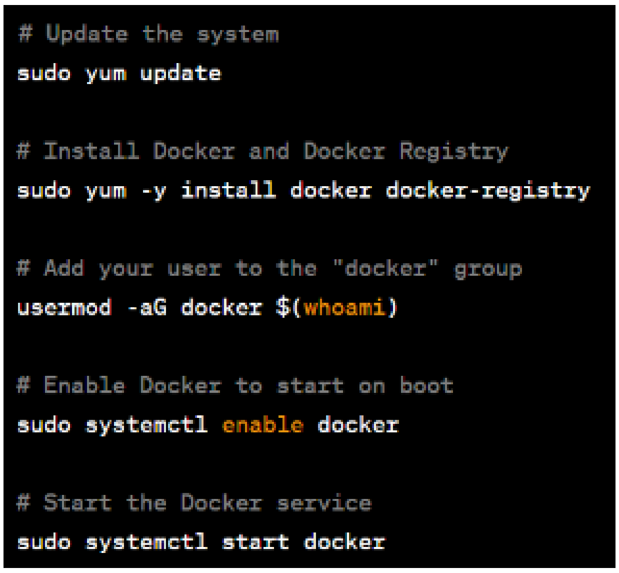
- **docker-compose up**: Creates and starts the services defined in the Compose file.
- **docker-compose down**: Stops and removes containers, networks, and volumes defined in the Compose file.
- **docker-compose ps**: Lists the status of services defined in the Compose file.

- **docker-compose exec** : Runs a command in a running container.
- **docker-compose up -d**: This command starts your services defined in the docker-compose.yml file in detached mode (-d). Detached mode means the containers run in the background, and you can continue using your terminal.
- **docker-compose ps**: This command lists the status of the services defined in the docker-compose.yml file. It shows whether they're running or stopped.
- **docker-compose stop**: This command stops the services defined in the docker-compose.yml file. It gracefully stops the containers and releases resources.
- **docker-compose config**: This command validates and displays the configuration of your docker-compose.yml file. It's useful for checking if there are any syntax errors or unexpected configurations.
- **docker-compose rm**: This command removes stopped service containers defined in the docker-compose.yml file. It cleans up containers that are no longer needed.
- **docker-compose down**: This command stops and removes all containers, networks, and volumes defined in the docker-compose.yml file. It's used to completely shut down and clean up your application.
- **docker-compose build**: This command builds the services defined in the docker-compose.yml file, using their respective Dockerfile definitions. It's used to create the images required for your services.
- **docker-compose logs**: This command displays the logs from the containers of the services defined in the docker-compose.yml file. You can use it to troubleshoot issues and monitor the output of your services.
- **docker-compose exec**: This command allows you to execute a command inside a running container. For example, you can use docker-compose exec to run a shell inside a service's container.
- **docker-compose top**: This command displays the running processes in your services' containers, similar to the top command on Linux.
- **docker-compose events**: This command displays real-time events from containers and services defined in the docker-compose.yml file.
- **docker-compose pause**: This command pauses all services defined in the docker-compose.yml file. It suspends the containers' processes but doesn't stop or remove them.
- **docker-compose unpause**: This command unpauses the services paused with docker-compose pause.
- **docker-compose kill**: This command sends a SIGKILL signal to the containers of the services defined in the docker-compose.yml file, forcibly stopping them.
- **docker-compose restart**: This command restarts the services defined in the docker-compose.yml file.

Usage:

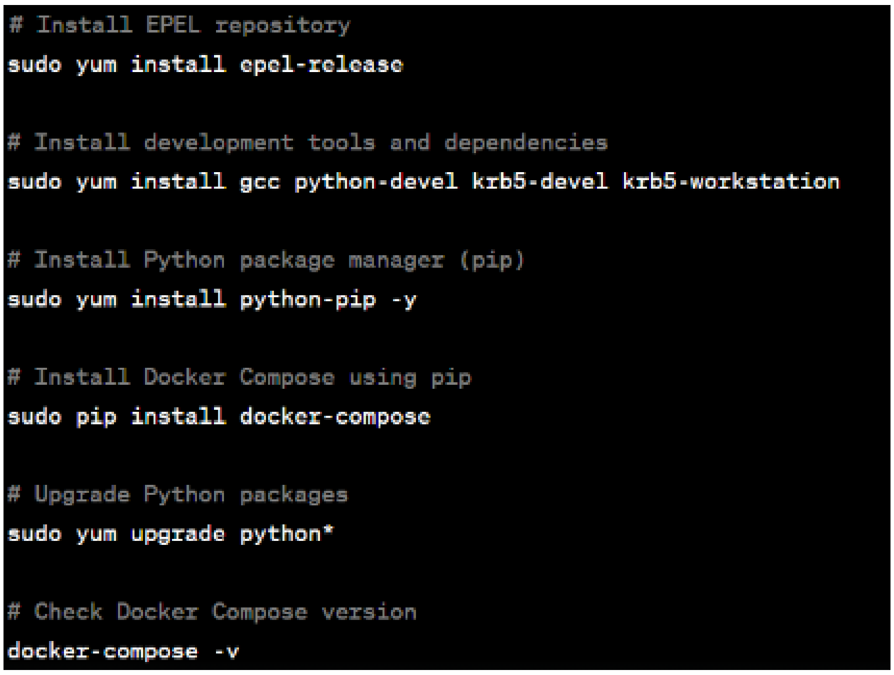
1. Create a `docker-compose.yml` file in your project directory.
2. Define your services, networks, and volumes in the YAML file.
3. Run `docker-compose up` to start your application.

Installing Docker



```
# Update the system
sudo yum update

# Install Docker and Docker Registry
sudo yum -y install docker docker-registry

# Add your user to the "docker" group
usermod -aG docker $(whoami)

# Enable Docker to start on boot
sudo systemctl enable docker

# Start the Docker service
sudo systemctl start docker
```

Installation of Docker Compose



```
# Install EPEL repository
sudo yum install epel-release

# Install development tools and dependencies
sudo yum install gcc python-devel krb5-devel krb5-workstation

# Install Python package manager (pip)
sudo yum install python-pip -y

# Install Docker Compose using pip
sudo pip install docker-compose

# Upgrade Python packages
sudo yum upgrade python*

# Check Docker Compose version
docker-compose -v
```

Creating a Docker-compose.yml file for a simple example

1. `mkdir hello-world` : This creates a directory named "hello-world" where you'll store your Docker Compose files.

2. `cd hello-world` : This changes your current directory to the newly created "hello-world" directory.

3. `vi docker-compose.yml` : This opens a text editor (vi) to create and edit the `docker-compose.yml` file.

4. Inside the `docker-compose.yml` file, you've added the following content:

   ```
   unixmen-compose-test:
     image: hello-world
   ```

This `docker-compose.yml` file defines a service named "unixmen-compose-test" that uses the "hello-world" image. The "hello-world" image is a simple image often used to test Docker setups.

After saving your changes to the `docker-compose.yml` file, you can use Docker Compose to start the service defined in it:

docker-compose up

This will pull the "hello-world" image (if not already present), create a container based on that image, and display a "Hello from Docker!" message.

Keep in mind that the provided `docker-compose.yml` file only defines one service. Docker Compose becomes even more powerful when managing applications with multiple interconnected services, such as databases, web servers, and more.

Troubleshooting

These steps will helpful for troubleshooting if `docker-compose` is not running and there are permission issues related to the Docker socket file. The Docker socket file is used by Docker to communicate with the Docker daemon.

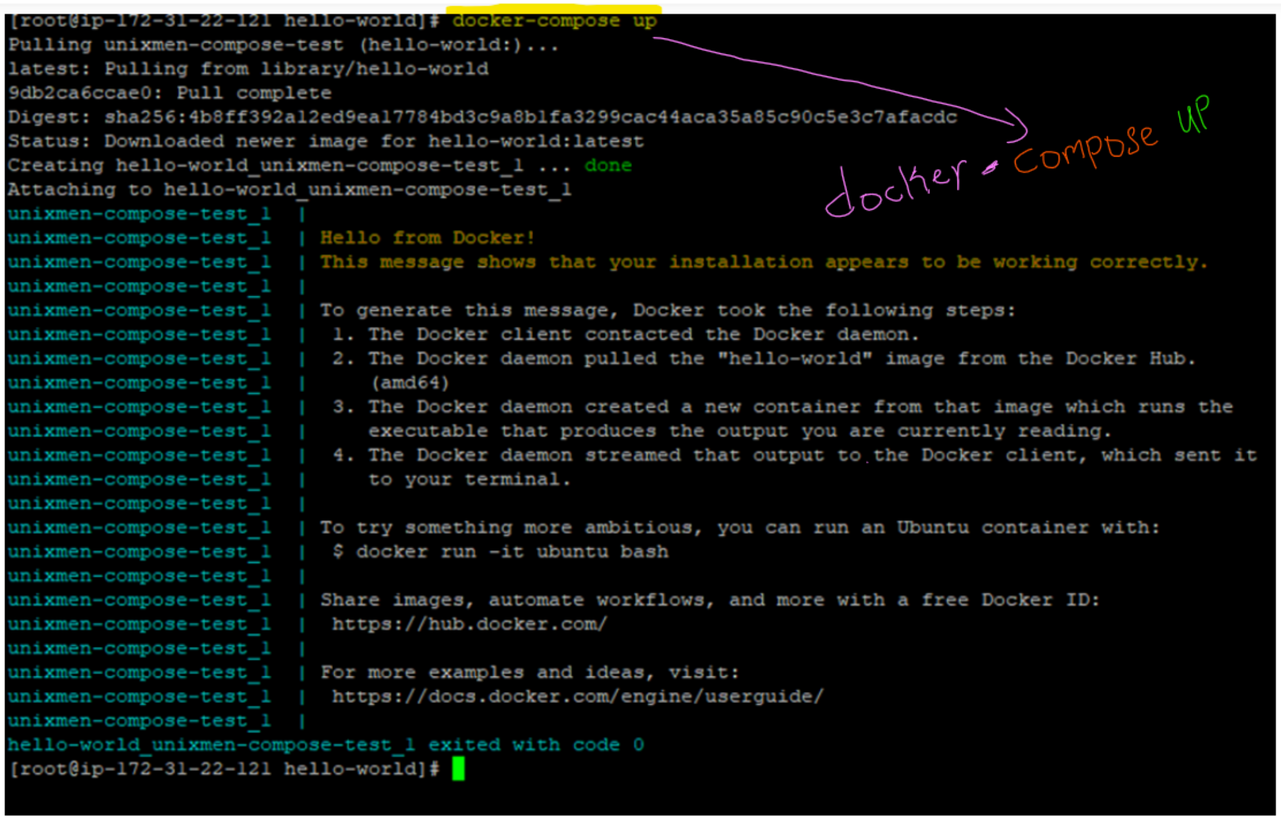
1. `sudo ls -la /var/run/docker.sock` : This command lists detailed information about the Docker socket file. The Docker socket is typically located at `/var/run/docker.sock` .

2. If you find that the owner of the Docker socket is `root` , it might cause permission issues when running Docker commands as a regular user. To resolve this:

`sudo chown ec2-user /var/run/docker.sock` : This command changes the ownership of the Docker socket to the user `ec2-user` . This can help ensure that the user has the necessary permissions to interact with the Docker daemon.

- The command `chown` stands for "change owner," and `/var/run/docker.sock` is the path to the Docker socket file. Adjust the username (`ec2-user` in this case) according to your system's user.

- By making this change, the user specified (e.g., `ec2-user`) will have the appropriate permissions to work with Docker, including running `docker-compose` without encountering permission-related issues.



```
[root@ip-172-31-22-121 hello-world]# docker-compose up
Pulling unixmen-compose-test (hello-world:)...
latest: Pulling from library/hello-world
9db2ca6ccae0: Pull complete
Digest: sha256:4b8ff392a12ed9ea17784bd3c9a8b1fa3299cac44aca35a85c90c5e3c7afacdc
Status: Downloaded newer image for hello-world:latest
Creating hello-world_unixmen-compose-test_1 ... done
Attaching to hello-world_unixmen-compose-test_1
unixmen-compose-test_1  |
unixmen-compose-test_1  | Hello from Docker!
unixmen-compose-test_1  | This message shows that your installation appears to be working correctly.
unixmen-compose-test_1  |
unixmen-compose-test_1  | To generate this message, Docker took the following steps:
unixmen-compose-test_1  |  1. The Docker client contacted the Docker daemon.
unixmen-compose-test_1  |  2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
unixmen-compose-test_1  |     (amd64)
unixmen-compose-test_1  |  3. The Docker daemon created a new container from that image which runs the
unixmen-compose-test_1  |     executable that produces the output you are currently reading.
unixmen-compose-test_1  |  4. The Docker daemon streamed that output to the Docker client, which sent it
unixmen-compose-test_1  |     to your terminal.
unixmen-compose-test_1  |
unixmen-compose-test_1  | To try something more ambitious, you can run an Ubuntu container with:
unixmen-compose-test_1  |  $ docker run -it ubuntu bash
unixmen-compose-test_1  |
unixmen-compose-test_1  | Share images, automate workflows, and more with a free Docker ID:
unixmen-compose-test_1  |  https://hub.docker.com/
unixmen-compose-test_1  |
unixmen-compose-test_1  | For more examples and ideas, visit:
unixmen-compose-test_1  |  https://docs.docker.com/engine/userguide/
unixmen-compose-test_1  |
hello-world_unixmen-compose-test_1 exited with code 0
[root@ip-172-31-22-121 hello-world]#
```

- **Environment Variables and Secrets:**

You can set environment variables for your services within the `docker-compose.yml` file. This is helpful for providing configuration to your containers. Additionally, Docker Compose allows you to manage sensitive information, like passwords, using Docker's secret management for better security.

```yaml
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    environment:
      MY_ENV_VARIABLE: my_value
    secrets:
      - my_secret
secrets:
  my_secret:
    file: ./my_secret.txt
```

In this example, an environment variable `MY_ENV_VARIABLE` is set for the `web` service, and a secret named `my_secret` is used. The secret's content is read from the `my_secret.txt` file.

- **Volume Mapping:**

Docker Compose enables you to map volumes between your host system and containers. This allows you to persist data even when containers are removed. Volumes can also be shared between services.

```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    volumes:
      - ./web-content:/usr/share/nginx/html
```

In this case, the `web` service's content is mapped from the local `./web-content` directory to the container's `/usr/share/nginx/html` directory.

- **Networking:**

Docker Compose creates a default network for your services, but you can also define custom networks to control communication between containers. Containers within the same Compose project can communicate with each other using service names as hostnames.

```
version: '3'
services:
  web:
    image: nginx:latest
    networks:
      - mynetwork
networks:
  mynetwork:
```

Here, the `web` service is connected to the `mynetwork` network.

- **Scaling Services:**

With Docker Compose, you can easily scale services to handle different levels of load. For instance, if you have a service that processes requests, you can scale it up to run multiple instances.

```
version: '3'
services:
  app:
    image: myapp:latest
    scale: 3
```

Running `docker-compose up` with this configuration will start three instances of the `app` service.

- **Extending Docker Compose:**

You can split your configuration into multiple Compose files to simplify management and enable reuse. This is done using the `-f` flag.

```
docker-compose -f docker-compose.yml -f docker-compose.prod.yml up
```

Here, both `docker-compose.yml` and `docker-compose.prod.yml` are used to define the application's configuration.

- **Docker Compose and Development:**

Docker Compose is particularly valuable during development. It allows you to set up consistent development environments across different machines, reducing the "it works on my machine" problem.

- **Healthchecks:**

Docker Compose allows you to define healthchecks for your services. Healthchecks determine the status of a container by periodically checking a specific command or endpoint.

```yaml
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
    healthcheck:
      test: ["CMD", "curl", "-f", "http://localhost"]
      interval: 30s
      timeout: 10s
      retries: 3
```

In this example, a healthcheck is defined for the `web` service. It runs the command `curl -f http://localhost` every 30 seconds, waiting up to 10 seconds for a response, and retrying up to 3 times before marking the container as unhealthy.

- **Dependencies and Initialization:**

Sometimes, you need to ensure that one service starts only after another service is ready. Docker Compose supports defining service dependencies and initialization scripts.

```yaml
version: '3'
services:
  db:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: mysecretpassword
  app:
    image: myapp:latest
    depends_on:
      - db
    command: ["./wait-for-db.sh", "db", "app-start-command"]
```

In this setup, the `app` service depends on the `db` service. It uses a custom script (`wait-for-db.sh`) to wait until the database is ready before starting its main command (`app-start-command`).

- **Override and Environment-specific Configuration:**

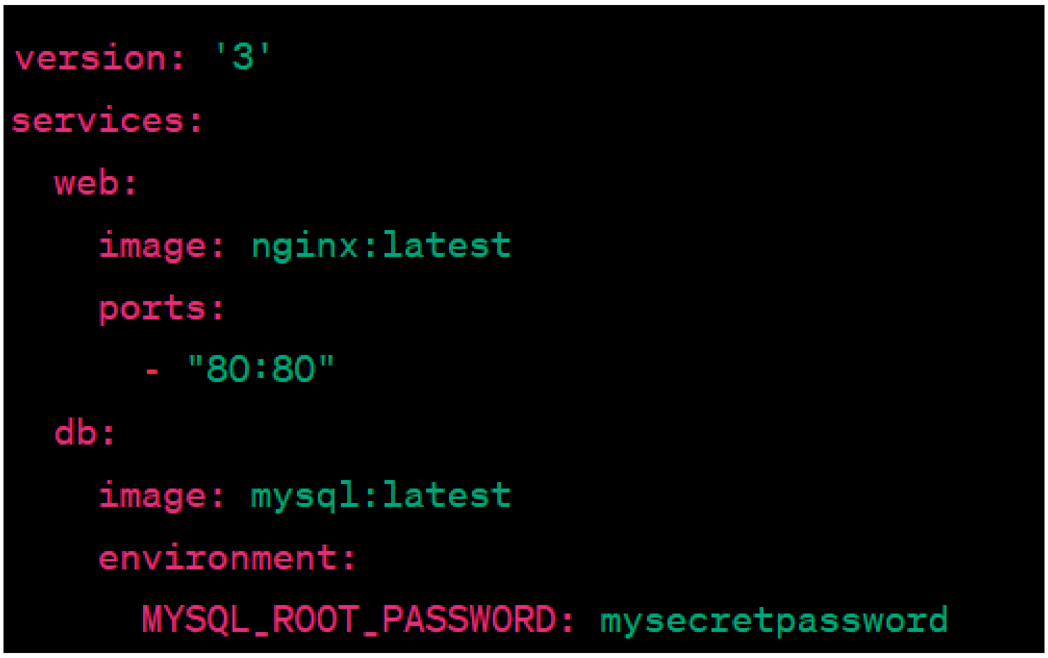
You can customize your Docker Compose setup for different environments (e.g., development, testing, production) by using environment variables and multiple `.env` files.

```
version: '3'
services:
  app:
    image: myapp:latest
    environment:
      ENVIRONMENT: ${ENVIRONMENT}
```

Docker Compose is an invaluable tool for managing multi-container applications, offering simplicity, automation, and portability. It streamlines the development, testing, and deployment of complex systems, enabling developers to focus on building and iterating on their applications without getting bogged down by container management intricacies.

Example : 1 (Nginx + MySQL)

In this example, we have two services: **web (NGINX) and db (MySQL)**. The web service uses the NGINX image and maps port 80. The db service uses the MySQL image and sets an environment variable for the root password.



```
version: '3'
services:
  web:
    image: nginx:latest
    ports:
      - "80:80"
  db:
    image: mysql:latest
    environment:
      MYSQL_ROOT_PASSWORD: mysecretpassword
```

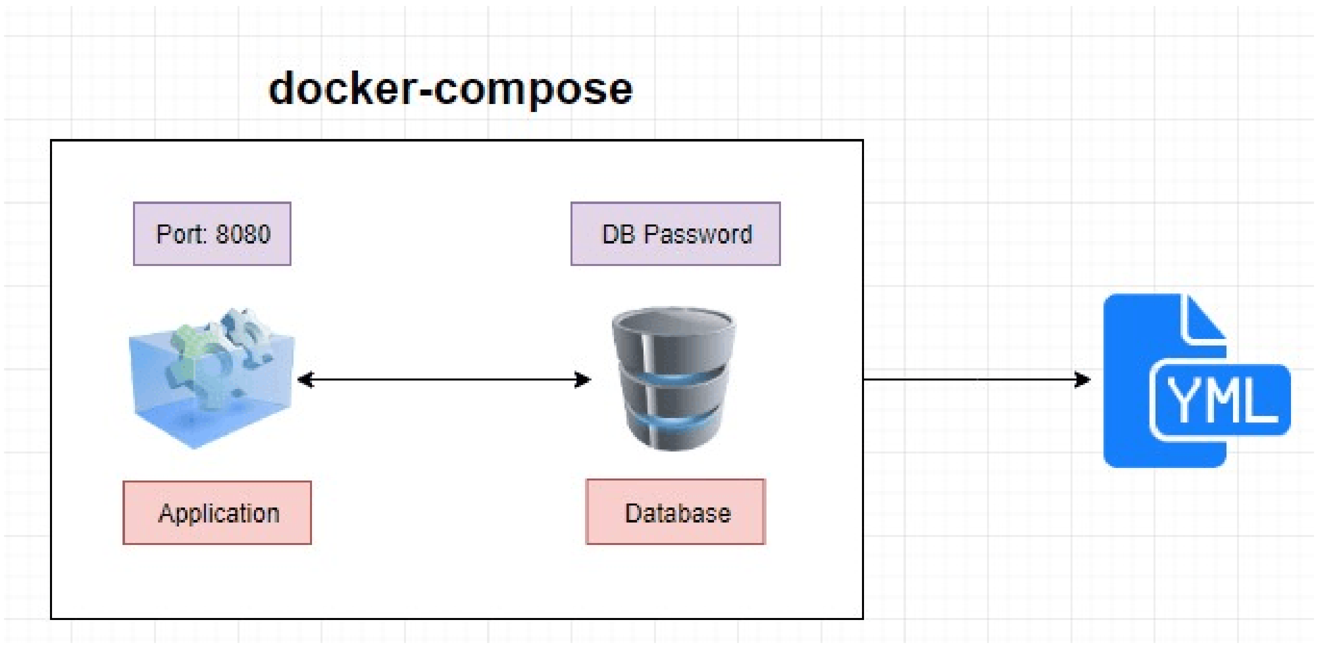
In this `docker-compose.yml` example, we have defined two services: `web` and `db` .

1. **Web Service (`web`):**

- The `web` service uses the official NGINX image (`nginx:latest`).
- The `ports` section maps port 80 from the host system to port 80 in the container. This means that when you access port 80 on your host, the traffic will be directed to the NGINX container.

2. **Database Service (`db`):**

- The `db` service uses the official MySQL image (`mysql:latest`).
- The `environment` section sets an environment variable `MYSQL_ROOT_PASSWORD` with the value `mysecretpassword` . This sets the root password for the MySQL database. m

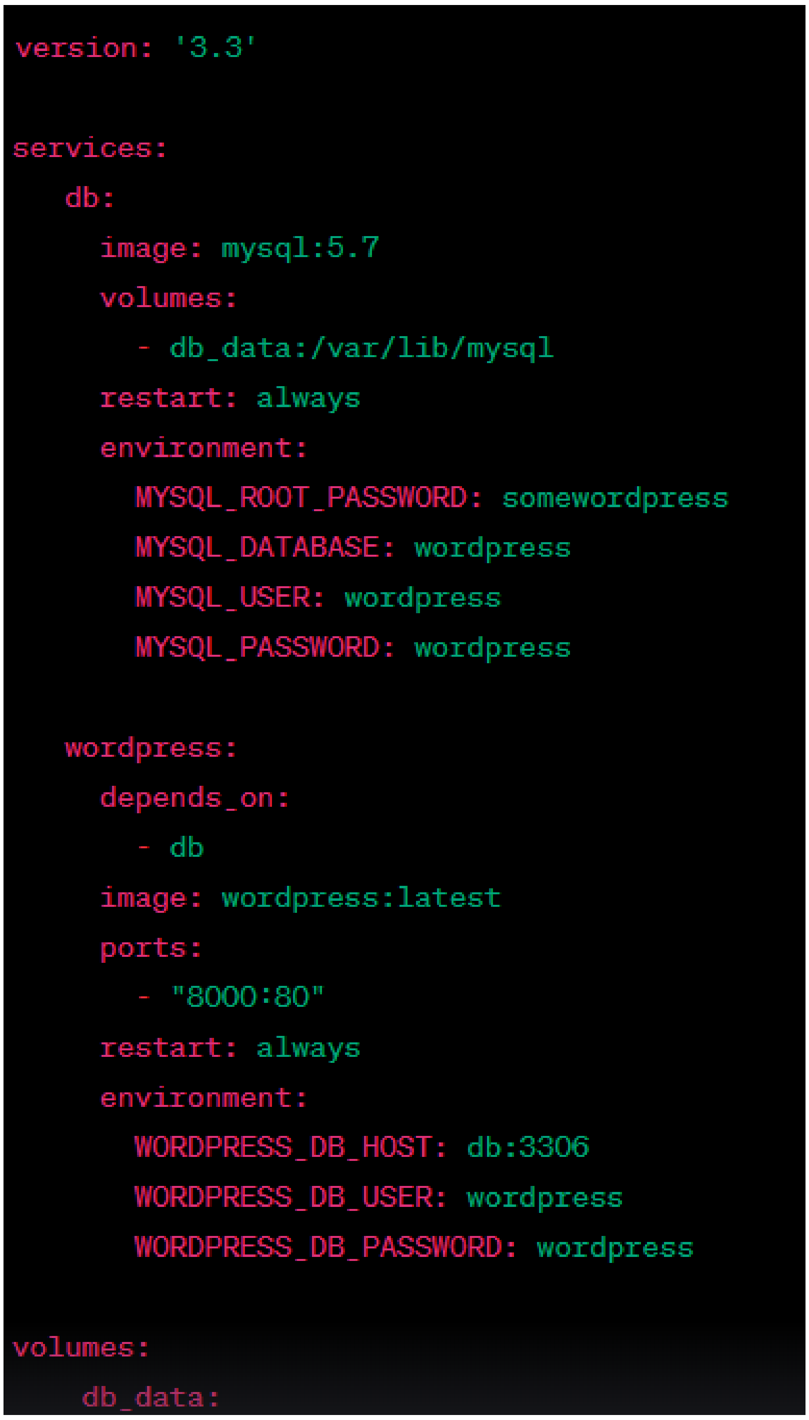


When you run `docker-compose up` , Docker Compose will create two containers: one running NGINX (`web`) and the other running MySQL (`db`). The NGINX container will be accessible on port 80 of your host system, and the MySQL container will have the root password set to "mysecretpassword".

This setup is particularly useful for quickly deploying multi-container applications with interconnected components. Docker Compose handles the creation, configuration, and networking of these containers, making it much simpler than manually managing each container individually.

Example: 2 (WordPress + MySQL)

A docker-compose.yml configuration file for **setting up a WordPress application along with a MySQL database**. This is a common use case for Docker Compose to manage multi-container applications. Let's go through your provided configuration:



```yaml
version: '3.3'

services:
   db:
     image: mysql:5.7
     volumes:
       - db_data:/var/lib/mysql
     restart: always
     environment:
       MYSQL_ROOT_PASSWORD: somewordpress
       MYSQL_DATABASE: wordpress
       MYSQL_USER: wordpress
       MYSQL_PASSWORD: wordpress

   wordpress:
     depends_on:
       - db
     image: wordpress:latest
     ports:
       - "8000:80"
     restart: always
     environment:
       WORDPRESS_DB_HOST: db:3306
       WORDPRESS_DB_USER: wordpress
       WORDPRESS_DB_PASSWORD: wordpress

volumes:
    db_data:
```

In this `docker-compose.yml` file:

1. I've specified the Compose file version as `3.3`.

2. I have defined two services: `db` (MySQL database) and `wordpress` (WordPress application).

3. The `db` service:

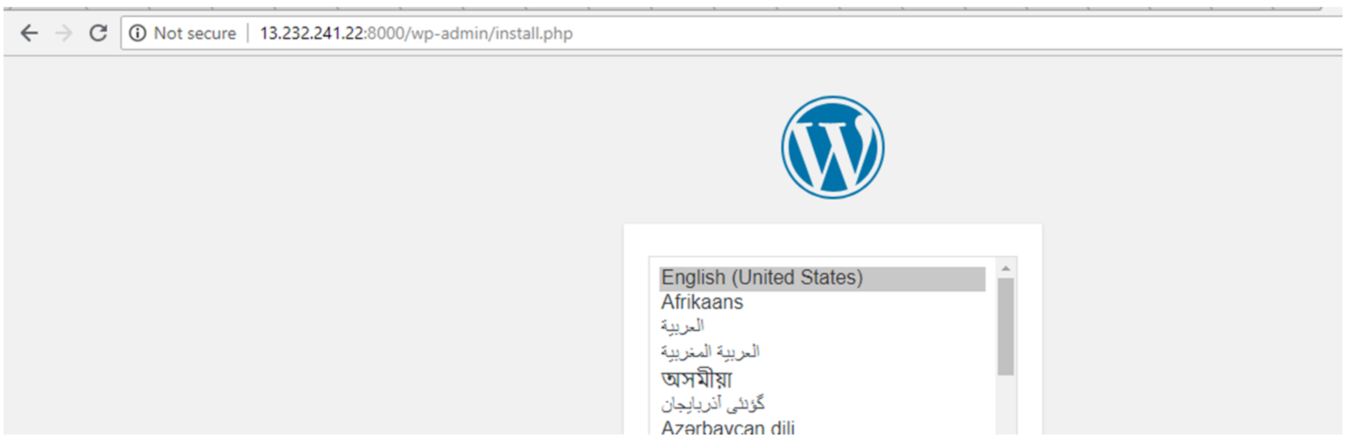
- Uses the MySQL 5.7 image.
- Defines a volume named `db_data` to persist MySQL data.
- Restarts the container automatically.
- Sets environment variables for the MySQL configuration, including root password, database name, user, and password.

4. The `wordpress` service:

- Depends on the `db` service to ensure the database is up before starting.
- Uses the latest WordPress image.
- Maps port 8000 on the host to port 80 in the container.
- Restarts the container automatically.
- Sets environment variables for WordPress to connect to the MySQL database, including host, user, and password.

5. The `volumes` section defines the volume `db_data` used by the `db` service to persist MySQL data1.

Verify in browser



When you run `docker-compose up` in the same directory as this `docker-compose.yml` file, Docker Compose will create and start the two services, setting up a WordPress instance that connects to a MySQL database.

Just ensure you have Docker and Docker Compose installed and then navigate to the directory containing this `docker-compose.yml` file in your terminal before running `docker-compose up` .

Example :3 (Python + Node.js + Redis + PostgreSQL + .NET)

A multi-component voting application architecture using different technologies for different purposes. Here's a breakdown of the technologies and their responsibilities:

1. Python (Backend Logic):

- Responsible for handling the backend logic of the application.
- Likely includes the business logic, data processing, and interaction with the database.
- Python is commonly used for its versatility and ease of use in building backend services.

2. Node.js (Frontend):

- Responsible for building the frontend part of the application.
- Handles the user interface, interactions, and presentation logic.
- Node.js is known for its asynchronous and event-driven capabilities, making it suitable for real-time applications.

3. Redis (Caching):

- Responsible for caching data to improve application performance.
- Redis is an in-memory data store that can significantly speed up data retrieval by storing frequently accessed data in memory.

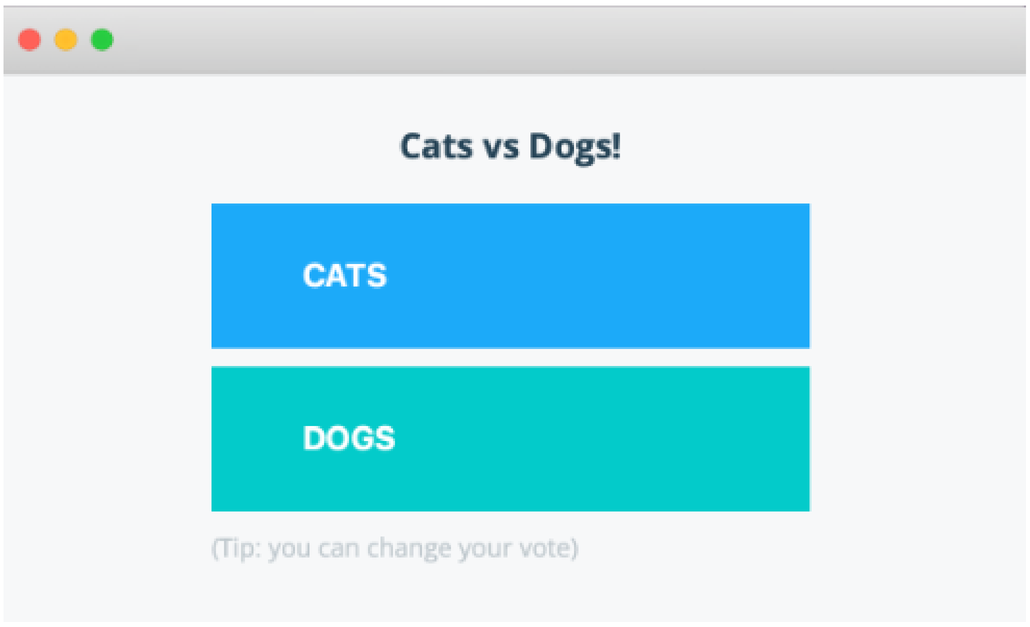
4. PostgreSQL (Database):

- Responsible for storing and managing the application's data.
- PostgreSQL is a powerful open-source relational database known for its advanced features and extensibility.

5. .NET (Application Component):

- Responsible for running specific aspects of the application.
- .NET is a framework that supports multiple programming languages and can be used for building various parts of an application.

This architecture outlines a common pattern in modern web development where different technologies are chosen for their strengths in different areas. For example, Python and Node.js are often used for backend and frontend respectively due to their strengths in those domains, while technologies like Redis and PostgreSQL offer specialized capabilities for caching and data storage.

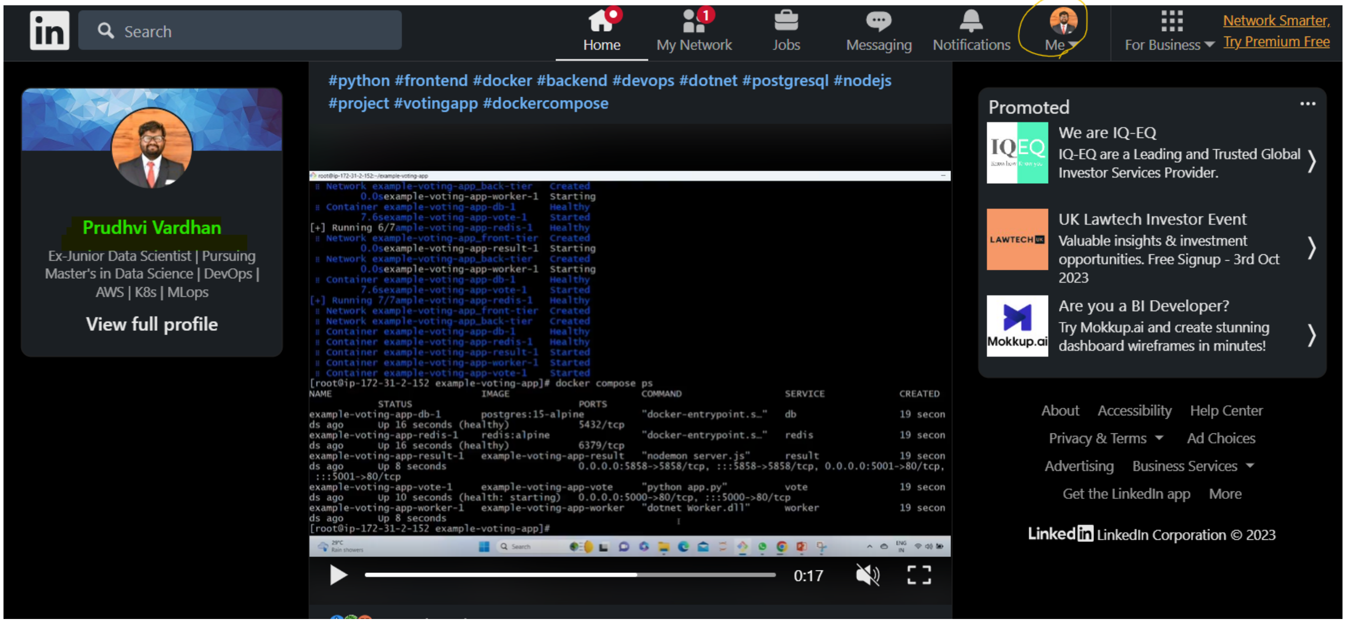


Dockerfile

```
# Use the official Python image as the base image
FROM python:3.8

# Set the working directory within the container
WORKDIR /app

# Copy the requirements file into the container at /app
COPY requirements.txt /app/

# Install any needed dependencies specified in requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copy the rest of the application code into the container at /app
COPY . /app/

# Set environment variables if needed
# ENV VARIABLE_NAME value

# Expose a port that the app will listen on
EXPOSE 5000

# Define the command to run the application
CMD ["python", "app.py"]
```

Check Out My LinkedIn Profile For Demo

https://www.linkedin.com/posts/prudhvi-vardhan_python-frontend-docker-activity-7044168454337105920-xfHh?utm_source=share&utm_medium=member_desktop (https://www.linkedin.com/posts/prudhvi-vardhan_python-frontend-docker-activity-7044168454337105920-xfHh?utm_source=share&utm_medium=member_desktop)


Search
Home
My Network
Jobs
Messaging
Notifications
Me
For Business
Network Smarter,
Try Premium Free
Prudhvi Vardhan
Ex-Junior Data Scientist | Pursuing Master's in Data Science | DevOps | AWS | K8s | MLops
View full profile
Promoted
We are IQ-EQ
IQ-EQ are a Leading and Trusted Global Investor Services Provider.
UK Lawtech Investor Event
Valuable insights & investment opportunities. Free Signup - 3rd Oct 2023
Are you a BI Developer?
Try Mokkup.ai and create stunning dashboard wireframes in minutes!
About
Accessibility
Help Center
Privacy & Terms
Ad Choices
Advertising
Business Services
Get the LinkedIn app
More
LinkedIn Corporation © 2023
0:17

#python #frontend #docker #backend #devops #dotnet #postgresql #nodejs #project #votingapp #dockercompose

```
root@ip-172-31-2-152:~/example-voting-app
 ⠿ Network example-voting-app_back-tier   Created
        0.0sexample-voting-app-worker-1  Starting
 ⠿ Container example-voting-app-db-1      Healthy
        7.6sexample-voting-app-vote-1    Started
[+] Running 6/7ample-voting-app-redis-1   Healthy
 ⠿ Network example-voting-app_front-tier  Created
        0.0sexample-voting-app-result-1  Starting
 ⠿ Network example-voting-app_back-tier   Created
        0.0sexample-voting-app-worker-1  Starting
 ⠿ Container example-voting-app-db-1      Healthy
        7.6sexample-voting-app-vote-1    Started
[+] Running 7/7ample-voting-app-redis-1   Healthy
 ⠿ Network example-voting-app_front-tier  Created
 ⠿ Network example-voting-app_back-tier   Created
 ⠿ Container example-voting-app-db-1      Healthy
 ⠿ Container example-voting-app-redis-1   Healthy
 ⠿ Container example-voting-app-result-1  Started
 ⠿ Container example-voting-app-worker-1  Started
 ⠿ Container example-voting-app-vote-1    Started
[root@ip-172-31-2-152 example-voting-app]# docker compose ps
NAME                        IMAGE                         COMMAND                    SERVICE        CREATED
            STATUS                              PORTS
example-voting-app-db-1     postgres:15-alpine            "docker-entrypoint.s…"    db             19 secon
ds ago      Up 16 seconds (healthy)             5432/tcp
example-voting-app-redis-1  redis:alpine                  "docker-entrypoint.s…"    redis          19 secon
ds ago      Up 16 seconds (healthy)             6379/tcp
example-voting-app-result-1 example-voting-app-result     "nodemon server.js"       result         19 secon
ds ago      Up 8 seconds                        0.0.0.0:5858->5858/tcp, :::5858->5858/tcp, 0.0.0.0:5001->80/tcp,
 :::5001->80/tcp
example-voting-app-vote-1   example-voting-app-vote       "python app.py"           vote           19 secon
ds ago      Up 10 seconds (health: starting)    0.0.0.0:5000->80/tcp, :::5000->80/tcp
example-voting-app-worker-1 example-voting-app-worker     "dotnet Worker.dll"       worker         19 secon
ds ago      Up 8 seconds
[root@ip-172-31-2-152 example-voting-app]#
```

-- Prudhvi Vardhan (LinkedIN)