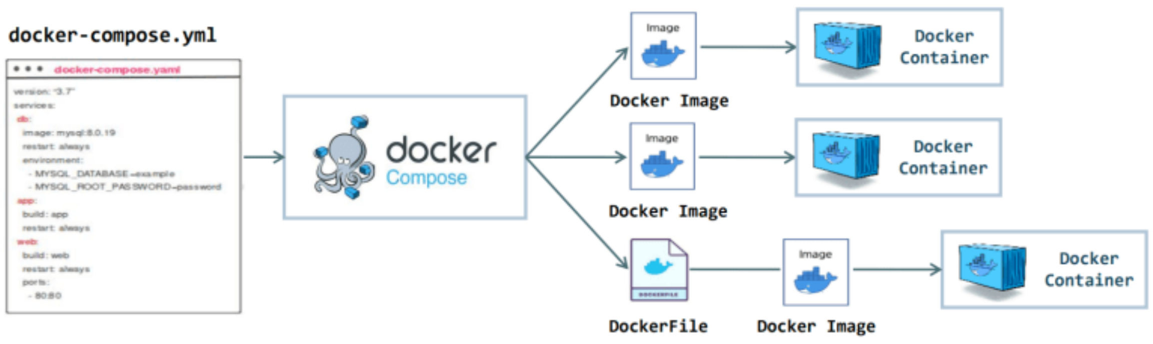
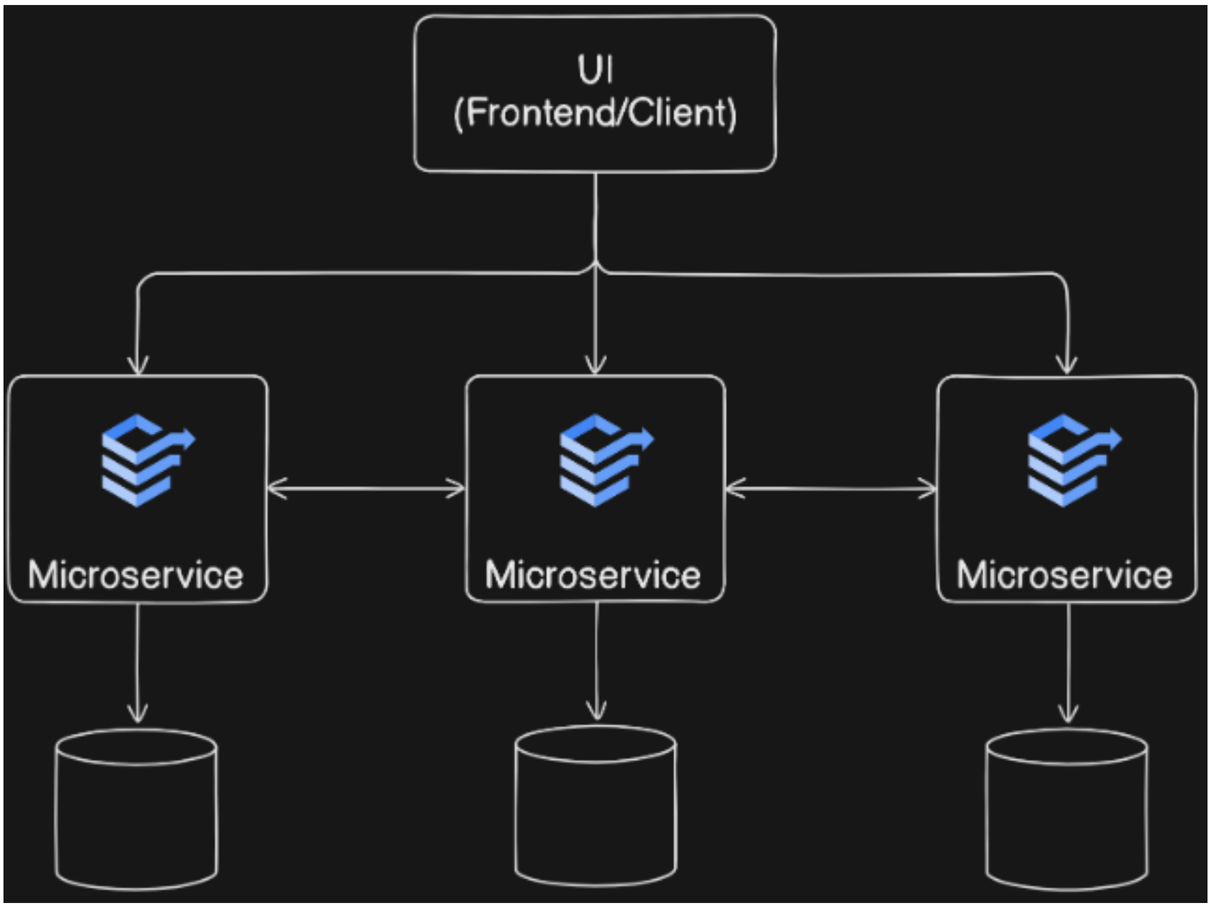
Docker Compose

What are Multi-Container Applications?
Modern cloud-native applications are often composed of multiple smaller services that work together to form a complete application



```
version: "3.7"
services:
 db:
  image: mysql:8.0.19
  restart: always
  environment:
   - MYSQL_DATABASE=example
   - MYSQL_ROOT_PASSWORD=password
 app:
  build: app
  restart: always
 web:
  build: web
  restart: always
  ports:
   - 80:80
```

This is known as the microservices pattern. These could include:



Web front-end
Database

Authentication service
and so on.

Challenges with Microservices:
Managing and deploying multiple microservices can be complex and cumbersome, requiring careful orchestration of each service.

What is Docker Compose?
Docker Compose is a tool that allows you to define and manage multi-container Docker applications. It uses a declarative configuration file, typically in YAML format, to specify the services, networks, and volumes required for the application.

Benefits of Docker Compose:
Simplifies the orchestration of multi-container applications.
Allows for a single configuration file to define and manage all services.
Integrates with version control systems for better management.
Basic Docker Compose Commands
Check Installation:

```
docker compose version
```

Start an Application:

```
docker compose up
```

Stop an Application:

```
docker compose down
```

View Container Status:

```
docker compose ps
```

Simple Docker Compose Example
Let's create a basic Docker Compose setup with a web server using nginx:alpine image and a redis service using redis:alpine image.

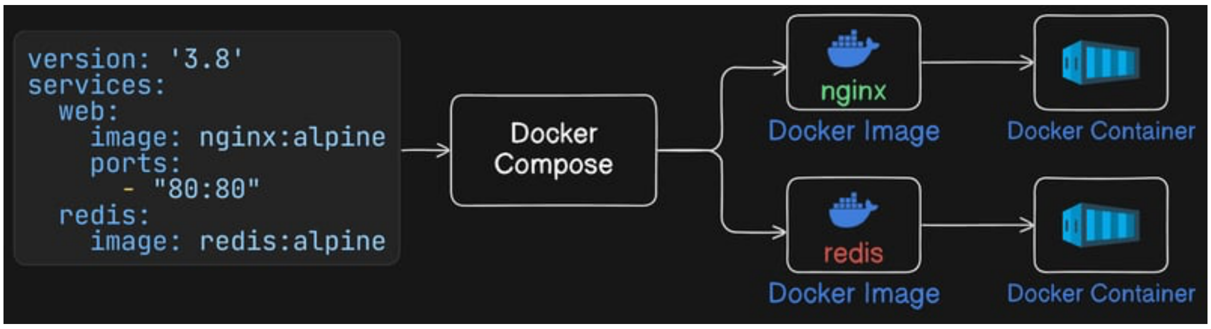
Step 1: Create a Directory
Create a directory for your project.

```
mkdir myapp

cd myapp
```

Step 2: Create Docker Compose File
Create a file named docker-compose.yml and add the following content:

```
version: '3.8'

services:

 web:

  image: nginx:alpine

  ports:

   - "80:80"

 redis:

  image: redis:alpine
```

This file defines two services: a web server (nginx) and a Redis server.



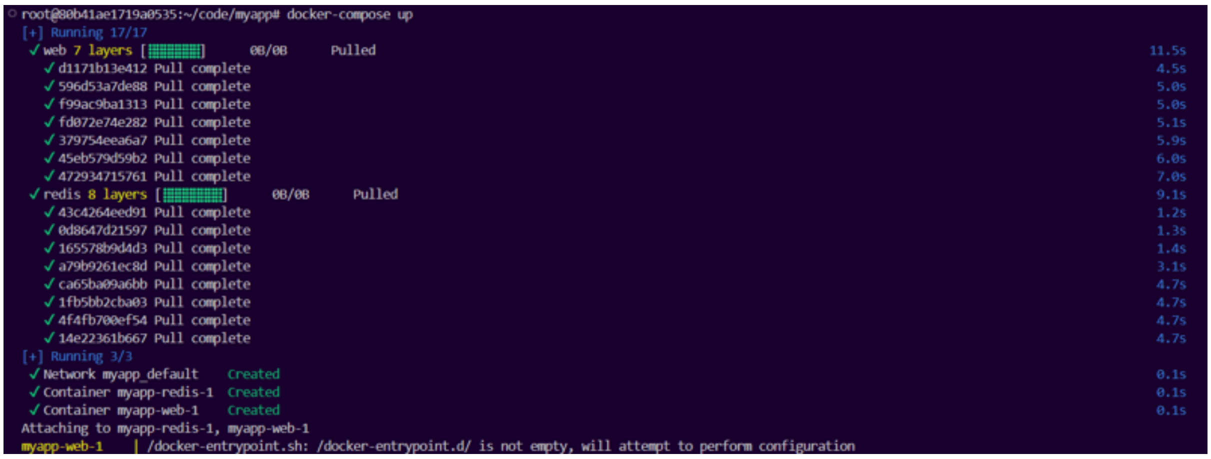
```
version: '3.8'
services:
  web:
    image: nginx:alpine
    ports:
      - "80:80"
  redis:
    image: redis:alpine
```

Step 3: Start the Application
Run the following command to start the services:

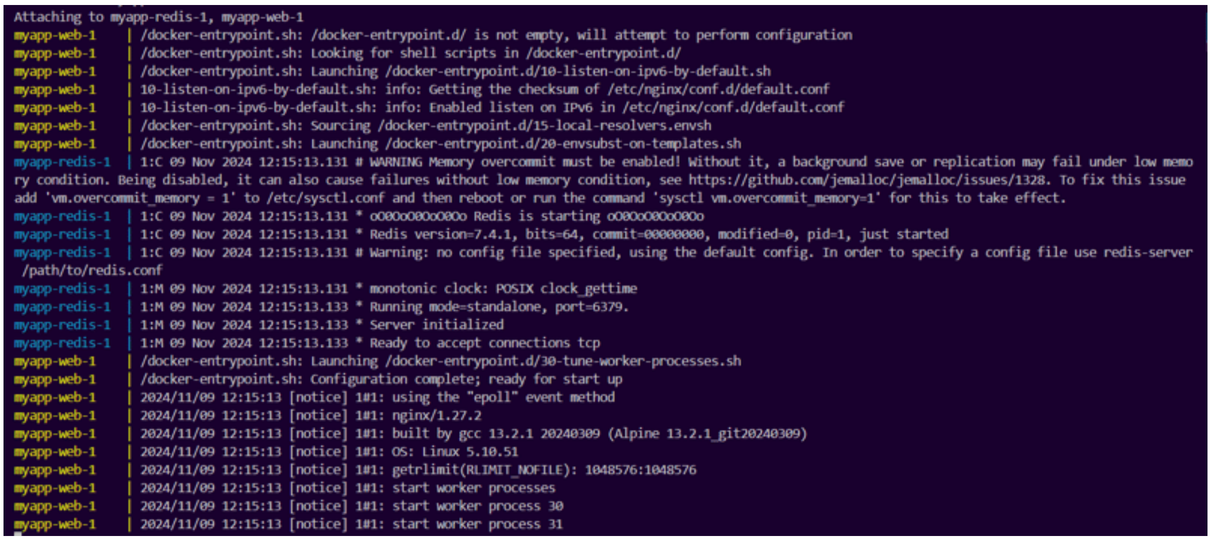
```
docker compose up
```

You should see output indicating the services are starting.

Expected output:



```
○ root@80b41ae1719a0535:~/code/myapp# docker-compose up
 [+] Running 17/17
  ✓ web 7 layers [███████]      0B/0B      Pulled                                                             11.5s
    ✓ d1171b13e412 Pull complete                                                                               4.5s
    ✓ 596d53a7de88 Pull complete                                                                               5.0s
    ✓ f99ac9ba1313 Pull complete                                                                               5.0s
    ✓ fd072e74e282 Pull complete                                                                               5.1s
    ✓ 379754eea6a7 Pull complete                                                                               5.9s
    ✓ 45eb579d59b2 Pull complete                                                                               6.0s
    ✓ 472934715761 Pull complete                                                                               7.0s
  ✓ redis 8 layers [████████]      0B/0B      Pulled                                                           9.1s
    ✓ 43c4264eed91 Pull complete                                                                               1.2s
    ✓ 0d8647d21597 Pull complete                                                                               1.3s
    ✓ 165578b9d4d3 Pull complete                                                                               1.4s
    ✓ a79b9261ec8d Pull complete                                                                               3.1s
    ✓ ca65ba09a6bb Pull complete                                                                               4.7s
    ✓ 1fb5bb2cba03 Pull complete                                                                               4.7s
    ✓ 4f4fb700ef54 Pull complete                                                                               4.7s
    ✓ 14e22361b667 Pull complete                                                                               4.7s
 [+] Running 3/3
  ✓ Network myapp_default    Created                                                                           0.1s
  ✓ Container myapp-redis-1  Created                                                                           0.1s
  ✓ Container myapp-web-1    Created                                                                           0.1s
 Attaching to myapp-redis-1, myapp-web-1
 myapp-web-1    | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
```



```
Attaching to myapp-redis-1, myapp-web-1
myapp-web-1    | /docker-entrypoint.sh: /docker-entrypoint.d/ is not empty, will attempt to perform configuration
myapp-web-1    | /docker-entrypoint.sh: Looking for shell scripts in /docker-entrypoint.d/
myapp-web-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/10-listen-on-ipv6-by-default.sh
myapp-web-1    | 10-listen-on-ipv6-by-default.sh: info: Getting the checksum of /etc/nginx/conf.d/default.conf
myapp-web-1    | 10-listen-on-ipv6-by-default.sh: info: Enabled listen on IPv6 in /etc/nginx/conf.d/default.conf
myapp-web-1    | /docker-entrypoint.sh: Sourcing /docker-entrypoint.d/15-local-resolvers.envsh
myapp-web-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/20-envsubst-on-templates.sh
myapp-redis-1  | 1:C 09 Nov 2024 12:15:13.131 # WARNING Memory overcommit must be enabled! Without it, a background save or replication may fail under low memo
ry condition. Being disabled, it can also cause failures without low memory condition, see https://github.com/jemalloc/jemalloc/issues/1328. To fix this issue
add 'vm.overcommit_memory = 1' to /etc/sysctl.conf and then reboot or run the command 'sysctl vm.overcommit_memory=1' for this to take effect.
myapp-redis-1  | 1:C 09 Nov 2024 12:15:13.131 * oO0OoO0OoO0Oo Redis is starting oO0OoO0OoO0Oo
myapp-redis-1  | 1:C 09 Nov 2024 12:15:13.131 * Redis version=7.4.1, bits=64, commit=00000000, modified=0, pid=1, just started
myapp-redis-1  | 1:C 09 Nov 2024 12:15:13.131 # Warning: no config file specified, using the default config. In order to specify a config file use redis-server
 /path/to/redis.conf
myapp-redis-1  | 1:M 09 Nov 2024 12:15:13.131 * monotonic clock: POSIX clock_gettime
myapp-redis-1  | 1:M 09 Nov 2024 12:15:13.133 * Running mode=standalone, port=6379.
myapp-redis-1  | 1:M 09 Nov 2024 12:15:13.133 * Server initialized
myapp-redis-1  | 1:M 09 Nov 2024 12:15:13.133 * Ready to accept connections tcp
myapp-web-1    | /docker-entrypoint.sh: Launching /docker-entrypoint.d/30-tune-worker-processes.sh
myapp-web-1    | /docker-entrypoint.sh: Configuration complete; ready for start up
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: using the "epoll" event method
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: nginx/1.27.2
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: built by gcc 13.2.1 20240309 (Alpine 13.2.1_git20240309)
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: OS: Linux 5.10.51
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: getrlimit(RLIMIT_NOFILE): 1048576:1048576
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: start worker processes
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: start worker process 30
myapp-web-1    | 2024/11/09 12:15:13 [notice] 1#1: start worker process 31
```

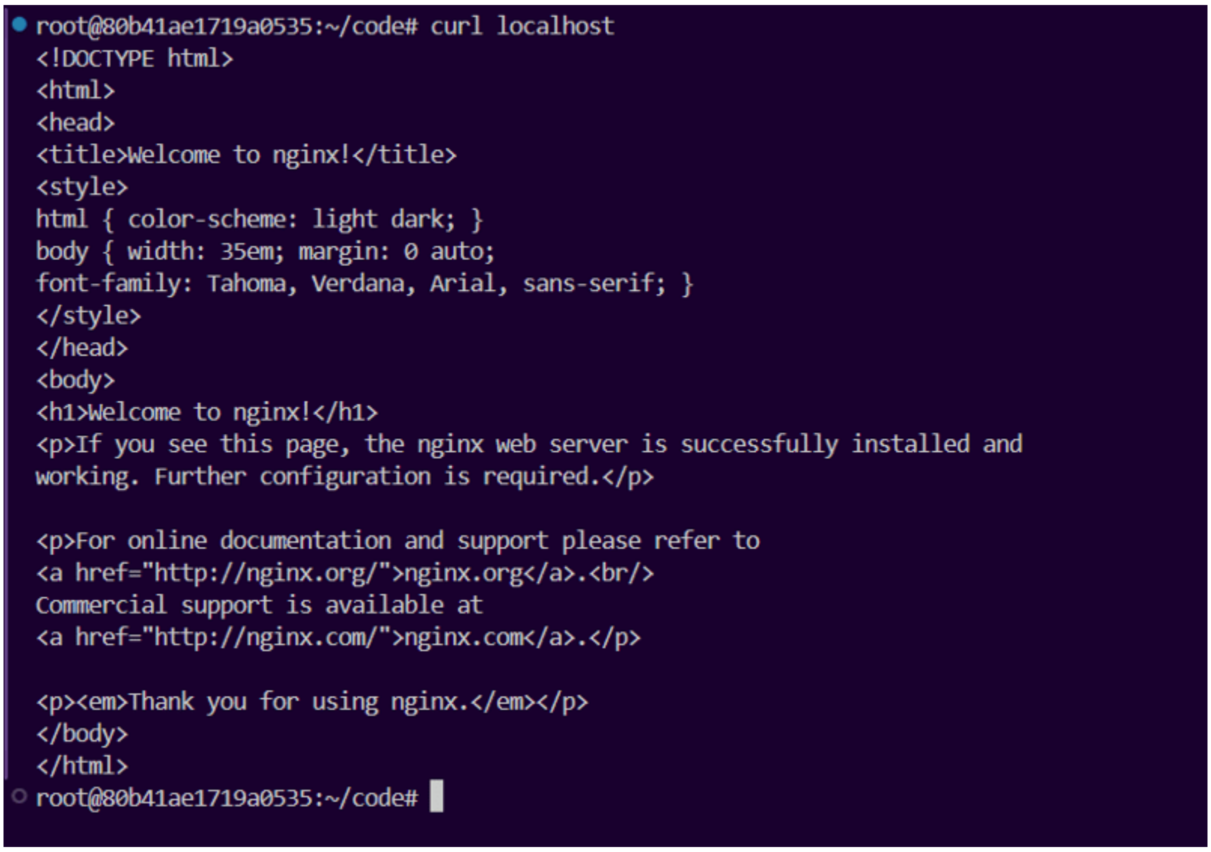
Step 4: Verify the Setup

Check localhost:

```
curl localhost
```

You should see the default Nginx welcome page.

Expected output:



```
root@80b41ae1719a0535:~/code# curl localhost
<!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>
</html>
root@80b41ae1719a0535:~/code#
```

Step 5: View Status
Check the status of the running containers:

```
docker compose ps

#or

docker ps
```

=====####====

Let's discuss Another Example:
Let's investigate docker-compose.yml to understand.

```
version: '3.5'

services:

 api:

  build: .

  volumes:

   - "./app:/src/app"

  ports:

   - "1338:1338"
  depends_on:
   - db
   - cache
  networks:
    - test_nw
  environment:
   - DATABASE_HOST=mongodb://db:27017
   - REDIS_CACHE_HOST=redis://cache:6379
   - PORT=1338
 db:
  image: mongo:latest
  ports:
   - "27017:27017"
  networks:
    - test_nw
 cache:
  image: redis:latest
  ports:
   - "6379:6379"
  networks:
   - test_nw
networks:
 test_nw:
  driver: bridge
```

This is a simple **docker-compse.yml** example to deploy a Nodejs backend with MongoDb and Redis.

The Commands
Of course docker-compose has commands. Whaat! Okay I' will give you the most basic

commands. I'm not super duper software engineer and I can live my life with these commands.

docker compose up

is a command that will look for docker-compose.yml by default and will process the docker-compose.yml, create the environment and run the services.
-d means that terminal is yours, it runs the command detachable mode
-f #non-standard-compose.yml-name# means that you can pass a compose.yml file with different name. Usually projects contains more than one docker-compose files. You can have compose file for production and development or you can seperate applications and tools in a different compose files.

docker compose down

is a command that will look for running compose.yml file and shutdown containers then remove all of them including networks, volumes etc.
docker compose log

is a command that will look for running compose.yml file and displays log which are generated by the containers.
Top Level Definitions

- version :
  Defined by the Compose Specification for backward compatibility. It is only informative, and you'll receive a warning message that it is obsolete if used.
- services :
  A service is an abstract definition of a computing resource within an application which can be scaled or replaced independently from other components. These services run in their containers and can communicate with each other.
- network :
  A layer that allows containers to communicate with each other.
- volume:
  Volumes are persistent data stores implemented by the container engine. Compose offers a neutral way for services to mount volumes, and configuration parameters to allocate them to infrastructure.

Service Definitions

- build:
  Lets you define the Dockerfile path to build when the compose file is being processed.

- volumes :
  Lets you define service-level persisted volumes to mount local files and folders.
- ports:
  Lets you expose container ports. The left-hand of the definition is the localhost address and the right-hand will be the container port. It basically means binding port 1338 of the container to localhost:1338.
- depends_on :
  option in Docker Compose is used to specify the dependencies between different services defined in your docker-compose.yml file.
- networks:
  option in Docker Compose is used to specify which network will be used by this container.
- environment:
  option in Docker Compose is used to specify environment variables which will be passed to container.