**Docker & Docker compose**

**V1 = Docker-compose** (deprecated)

**V2 = docker compose**

**Docker cmd summary -**

|  |  |
| --- | --- |
| 1. **Create Dockerfile for app environment** | **# vi Dockerfile** |
| 1. **Build image** | **# docker build -t *Name*** |
| 1. **Start app container (in background)**   **(map host port to app container port)** | **# docker run -d *Name***  **OR**  **# docker run -dp *IP*:*HostPort*:*ContainerPort* *Name*** |
| 1. **Check app via browser** | [**http://localhost:port**](http://localhost:port) |
| 1. **Check app container status** | **# docker ps** |

**Docker compose cmd summary -**

|  |  |
| --- | --- |
| 1. **Create Dockerfile for app environment** | **# vi Dockerfile** |
| 1. **Create compose.yaml for app services** | **# vi compose.yaml** |
| 1. **Do a dry run** | **# docker compose --dry-run -d** |
| 1. **Start app container (in background)** | **# docker compose up -d [-f /*path*/compose.yaml]** |
| 1. **Check app container logs** | **# docker compose logs -f [*ServiceName*]** |
| 1. **Check app via browser** | [**http://localhost:port**](http://localhost:port) |
| 1. **Check app container status** | **# docker ps** |

**Docker**

You can only start a single container with docker cmd.

1. **Create Dockerfile**

# vi Dockerfile :

[FROM](https://docs.docker.com/reference/dockerfile/#from) node:18-alpine

[WORKDIR](https://docs.docker.com/reference/dockerfile/#workdir) /app

[COPY](https://docs.docker.com/reference/dockerfile/#copy) . .

[RUN](https://docs.docker.com/reference/dockerfile/#run) yarn install --production

[CMD](https://docs.docker.com/reference/dockerfile/#cmd) ["node", "src/index.js"]

[EXPOSE](https://docs.docker.com/reference/dockerfile/#expose) 3000

1. **Build image**

# docker build -t *ImageName* .

1. **Start app container**

# docker run -d *ImageName*  (run detached in background)

Or

# docker run -dp 127.0.0.1:3000:3000 *ImageName* (map local host port to container port)

1. **Check app via browser** 
   1. http://localhost:3000/
2. **Check app container status**

# docker ps

**Docker compose**

[Docker Compose](https://docs.docker.com/compose/) is a tool that helps you define and share multi-container applications. With Compose, you can create a YAML file to define the services and with a single command, you can spin everything up or tear it all down.

1. **Define the** **app's environment** with a Dockerfile (no postfix) so it can be reproduced anywhere. Note - Each *COPY* directive results in a new layer in the final Docker image.

# **vi Dockerfile :**

[FROM](https://docs.docker.com/reference/dockerfile/#from) node:18-alpine

[WORKDIR](https://docs.docker.com/reference/dockerfile/#workdir) /app

[COPY](https://docs.docker.com/reference/dockerfile/#copy) . .

[RUN](https://docs.docker.com/reference/dockerfile/#run) yarn install --production

[CMD](https://docs.docker.com/reference/dockerfile/#cmd) ["node", "src/index.js"]

[EXPOSE](https://docs.docker.com/reference/dockerfile/#expose) 3000

1. **Define the** **app services** in **compose.yaml** so they can be run together in an isolated environment.

# **vi compose.yaml**

services:

web:

build: .

ports:

- "5000:5000"

volumes:

- .:/code

redis:

image: redis

1. **Perform a dry run of app startup**

# docker compose --dry-run -d [*Comnmand*]

1. **Start the app**

# docker compose up -d (run in background, using compose.yaml)

Or

# docker compose -f /*path*/docker-compose.yml

1. **Check logs**

# docker compose logs -f [*ServiceName*] (shows logs for each service interleaved into a single stream)

…

mysql\_1 | 2019-10-03T03:07:16.083639Z 0 [Note] mysqld: ready for connections.

mysql\_1 | Version: '8.0.31' socket: '/var/run/mysqld/mysqld.sock' port: 3306 MySQL Community Server (GPL)

app\_1 | Connected to mysql db at host mysql

app\_1 | Listening on port 3000

1. **Check app via browser**

<http://localhost:80/>

1. **Check container status**

# docker ps

To Check network status

# docker network ls

To delete the container app and network

# docker compose down [--volumes]

**Docker services**

# systemctl start docker

# systemctl enable docker

# systemctl status docker

**Important Instructions used in Dockerfile**

A Dockerfile is a text document that includes all the different steps and instructions on how to build a Docker image. The main elements described in the Dockerfile are the base image, required dependencies, and commands to execute application deployment within a container.

The essential instructions of a Dockerfile are illustrated below −

**FROM**

This instruction sets the base image on which the new image is going to be built upon. It is usually the first instruction in a Dockerfile.

FROM ubuntu:22

**RUN**

This will be an instruction that will be executed for running the commands inside the container while building. It typically can be utilized to install an application, update libraries, or do general setup.

RUN apt-get update && apt-get install -y python3

**COPY**

This instruction copies files and directories from the host machine into the container image.

COPY ./app /app

**ADD**

Like COPY but more advanced in features like it auto-decompresses archives and fetches files from URLs.

ADD https://example.com/file.tar.gz /app

**WORKDIR**

The instruction sets the working directory where the subsequent commands in a Dockerfile will be executed.

WORKDIR /app

**ENV**

The ENV instruction in this command defines environment variables within the container.

ENV FLASK\_APP=main.py

**EXPOSE**

This option defines to Docker that the container listens on the declared network ports at runtime.

EXPOSE 8000

**CMD**

Defines defaults for an executing container. There can only be one CMD instruction in a Dockerfile. If you list more than one CMD, then only the last CMD will take effect.

CMD ["python3", "main.py"]

**ENTRYPOINT**

This is an instruction that enables the configuration of a container to run the container as an executable.

ENTRYPOINT ["python3", "main.py"]

**LABEL**

This command provides meta-information for an image, like details of the maintainer, version, or description.

LABEL maintainer="johndoe@example.com"

**ARG**

This command defines a variable that allows users to be passed to the builder at build time using the "--build-arg" flag on the docker build command.

ARG version=1

**VOLUME**

It simply creates a mount point and assigns the given name to it, indicating that it will hold externally mounted volumes from the native host or other containers.

VOLUME /app/data

USER

This instruction allows the setting of the username (or UID) and optionally the group (or GID) to be used when running that image and for any RUN, CMD, and ENTRYPOINT instructions that follow it in the Dockerfile.

USER johndoe

**Best Practices for Dockerfile**

* Use multi-stage builds to slash your final image size by dropping unwanted build tools and dependencies. This way, you partition the build and runtime environment to attain peak efficiencies.
* Create reusable stages. If you have multiple images with a lot in common, consider creating a reusable stage that includes the shared components, and basing your unique stages on that.
* Use Official Base Images − Build on top of the official Docker Hub images. They tend to be minimal and well-maintained. Usually, they are optimized for security and size, laying a solid foundation for a custom image.
* Rebuild your images often. Docker images are immutable. Building an image is taking a snapshot of that image at that moment. That includes any base images, libraries, or other software you use in your build. To keep your images up-to-date and secure, make sure to rebuild your image often, with updated dependencies.
* Minimize the Number of Layers − As you learned earlier, each instruction in a Dockerfile creates a layer. Whenever possible, combine any commands related to one another in a single RUN instruction. This will help reduce the number of layers created for any build, making builds more cacheable.
* Leverage Build Cache − Ensure that Dockerfile instructions that can change more frequently, such as COPY, are placed towards the end. This would enable building again more rapidly upon making changes at later stages.
* Install Only Necessary Packages − Install only necessary packages and dependencies in your application to reduce the image size and possible vulnerabilities there.
* Use '.dockerignore' − To exclude unnecessary files and directories from the build context, add a '.dockerignore' file. This will speed up builds and prevent sensitive information from being leaked into your image.
* Use Non-Root User − Run containers with a non-root user to enhance security. It is always a good idea to give a specific user and group in Dockerfile another isolation layer.
* Image Scanning − Scan your Docker images often for vulnerabilities. With technologies such as Trivy and Clair, there are several tools that you could use for this kind of scanning. Keep your base images and dependencies up to date at all times to minimize the potential risk.
* Document your Dockerfile − Comment and explain your Dockerfile, you'll thank yourself later. This helps others, even your future self, understand the build process.
* Pin Versions − Pin versions for base images and dependencies, as this ensures reproducibility and avoids any unintended issues by getting updated.
* Sort multi-line arguments. Whenever possible, sort multi-line arguments alphanumerically to make maintenance easier. This helps to avoid duplication of packages and make the list much easier to update.
* Decouple applications. Each container should have only one concern. Decoupling applications into multiple containers makes it easier to scale horizontally and reuse containers.
* Build and test your images in CI. When you check in a change to source control or create a pull request, use [GitHub Actions](https://docs.docker.com/build/ci/github-actions/) or another CI/CD pipeline to automatically build and tag a Docker image and test it.

**Docker Subcommands**

Command Description

docker build (legacy builder) Build an image from a Dockerfile

docker builder Manage builds

docker buildx Docker Buildx

docker checkpoint Manage checkpoints

docker compose Docker Compose

docker config Manage Swarm configs

docker container Manage containers

docker context Manage contexts

docker debug Get a shell into any container or image. An alternative to debugging with `docker exec`.

docker image Manage images

docker init Creates Docker-related starter files for your project

docker inspect Return low-level information on Docker objects

docker login Authenticate to a registry

docker logout Log out from a registry

docker manifest Manage Docker image manifests and manifest lists

docker network Manage networks

docker node Manage Swarm nodes

docker plugin Manage plugins

docker scout Command line tool for Docker Scout

docker search Search Docker Hub for images

docker secret Manage Swarm secrets

docker service Manage Swarm services

docker stack Manage Swarm stacks

docker swarm Manage Swarm

docker system Manage Docker

docker trust Manage trust on Docker images

docker version Show the Docker version information

docker volume Manage volumes

**Docker-compose subcommands**

Command Description

docker compose alpha Experimental commands

docker compose build Build or rebuild services

docker compose config Parse, resolve and render compose file in canonical format

docker compose cp Copy files/folders between a service container and the local filesystem

docker compose create Creates containers for a service

docker compose down Stop and remove containers, networks

docker compose events Receive real time events from containers

docker compose exec Execute a command in a running container

docker compose images List images used by the created containers

docker compose kill Force stop service containers

docker compose logs View output from containers

docker compose ls List running compose projects

docker compose pause Pause services

docker compose port Print the public port for a port binding

docker compose ps List containers

docker compose pull Pull service images

docker compose push Push service images

docker compose restart Restart service containers

docker compose rm Removes stopped service containers

docker compose run Run a one-off command on a service

docker compose start Start services

docker compose stop Stop services

docker compose top Display the running processes

docker compose unpause Unpause services

docker compose up Create and start containers

docker compose version Show the Docker Compose version information

docker compose wait Block until the first service container stops

docker compose watch Watch build context for service and rebuild/refresh containers when files are updated