Getting started with Docker Compose

- Now for the advanced stuff. Docker Compose is a Docker tool used to define and run multi-container applications. With Compose, you use a YAML file to configure your application's services and create all the app's services from that configuration. Think of docker-compose as an automated multi-container workflow. Compose is an excellent tool for development, testing, CI workflows, and staging environments. According to the Docker documentation, the most popular features of Docker Compose are:
    - Multiple isolated environments on a single host
    - Preserve volume data when containers are created
    - Only recreate containers that have changed
    - Variables and moving a composition between environments
    - Orchestrate multiple containers that work together

How to use and install Docker Compose

- Compose uses the Docker Engine, so you'll need to have the Docker Engine installed on your device. You can run Compose on Windows, Mac, and 64-bit Linux. Installing Docker Compose is actually quite easy. On desktop systems, such as Docker Desktop for Mac and Windows, Docker Compose is already included. No additional steps are needed. On Linux systems, you'll need to:
    - Install the Docker Engine
    - Run the following command to download Docker Compose

```
sudo curl -L "https://github.com/docker/compose/releases/download/1.26.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

Apply permissions to the binary, like so:

```
sudo chmod +x /usr/local/bin/docker-compose
```

Test the installation to check it worked properly

```
$ docker-compose --version
```

- Once you have Docker Compose downloaded and running properly, however you choose to install it, you can start using it with your Dockerfiles. This process requires three basic steps:
    - Define your app's environment using a Dockerfile. This way, it can be reproduced.
    - Define the services for your app in a docker-compose.yml file. This way, they can run in an isolated environment.

    - Run docker-compose to start your app.

- You can easily add Docker Compose to a pre-existing project. If you already have some Dockerfiles, add Docker Compose files by opening the Command Palette. Use the Docker: Docker Compose Files to the Workspace command, and, when promoted, choose the Dockerfiles you want to include.

- You can also add Docker Compose files to your workspace when you add a Dockerfile. Similarly, open the Command Palette and use the Docker: Add Docker Files to Workspace command. You'll then be asked if you want to add any Docker Compose files. In both cases, Compose extension will add the `docker-compose.yml` file to your workspace.

Docker Compose file structure

Now that we know how to download Docker Compose, we need to understand how Compose files work. It's actually simpler than it seems. In short, Docker Compose files work by applying mutiple commands that are declared within a single docker-compose.yml configuration file. The basic structure of a Docker Compose YAML file looks like this:

```
version: 'X'

services:
  web:
    build: .
    ports:
     - "5000:5000"
    volumes:
     - .:/code
  redis:
    image: redis
```

Now, let's look at real-world example of a Docker Compose file and break it down step-by-step to understand all of this better. Note that all the clauses and keywords in this example are commonly used keywords and industry standard. With just these, you can start a development workflow. There are some more advanced keywords that you can use in production, but for now, let's just get started with the necessary clauses.

```
version: '3'
services:
  web:
    # Path to dockerfile.
    # '.' represents the current directory in which
    # docker-compose.yml is present.
```

```
  build: .

  # Mapping of container port to host

  ports:
    - "5000:5000"
  # Mount volume
  volumes:
    - "/usercode/:/code"

  # Link database container to app container
  # for reachability.
  links:
    - "database:backenddb"

database:

  # image to fetch from docker hub
  image: mysql/mysql-server:5.7

  # Environment variables for startup script
  # container will use these variables
  # to start the container with these define variables.
  environment:
    - "MYSQL_ROOT_PASSWORD=root"
    - "MYSQL_USER=testuser"
    - "MYSQL_PASSWORD=admin123"
    - "MYSQL_DATABASE=backend"
  # Mount init.sql file to automatically run
  # and create tables for us.
  # everything in docker-entrypoint-initdb.d folder
  # is executed as soon as container is up nd running.
  volumes:
    - "/usercode/db/init.sql:/docker-entrypoint-initdb.d/init.sql"
```

- version ‘3’: This denotes that we are using version 3 of Docker Compose, and Docker will provide the appropriate features. At the time of writing this article, version 3.7 is latest version of Compose.
- services: This section defines all the different containers we will create. In our example, we have two services, web and database.
- web: This is the name of our Flask app service. Docker Compose will create containers with the name we provide.
- build: This specifies the location of our Dockerfile, and . represents the directory where the docker-compose.yml file is located.
- ports: This is used to map the container’s ports to the host machine.

- volumes: This is just like the -v option for mounting disks in Docker. In this example, we attach our code files directory to the containers' ./code directory. This way, we won't have to rebuild the images if changes are made.
- links: This will link one service to another. For the bridge network, we must specify which container should be accessible to which container using links.
- image: If we don't have a Dockerfile and want to run a service using a pre-built image, we specify the image location using the image clause. Compose will fork a container from that image.
- environment: The clause allows us to set up an environment variable in the container. This is the same as the -e argument in Docker when running a container.

Congrats! Now you know a bit about Docker Compose and the necessary parts you'll need to get started with your workflow.

docker-compose commands

docker-compose

Every command starts with this. Anything you want to do in Compose, you have to do with commands starting with docker-compose. docker-comopse --help will give you a list of commands provided in the installed version of docker-compose.

```
$ docker-compose --help
Define and run multi-container applications with Docker.

Usage:
  docker-compose [-f <arg>...] [options] [COMMAND] [ARGS...]
  docker-compose -h|--help

Options:
  -f, --file FILE             Specify an alternate compose file
                              (default: docker-compose.yml)
  -p, --project-name NAME     Specify an alternate project name
                              (default: directory name)
  --verbose                   Show more output
  --log-level LEVEL           Set log level (DEBUG, INFO, WARNING, ERROR,
CRITICAL)
  --no-ansi                   Do not print ANSI control characters
  -v, --version               Print version and exit
  -H, --host HOST             Daemon socket to connect to

  --tls                       Use TLS; implied by --tlsverify
  --tlscacert CA_PATH         Trust certs signed only by this CA
  --tlscert CLIENT_CERT_PATH  Path to TLS certificate file
  --tlskey TLS_KEY_PATH       Path to TLS key file
  --tlsverify                 Use TLS and verify the remote
```

```
  --skip-hostname-check       Don't check the daemon's hostname against the
                              name specified in the client certificate
  --project-directory PATH    Specify an alternate working directory
                              (default: the path of the Compose file)
  --compatibility             If set, Compose will attempt to convert keys
                              in v3 files to their non-Swarm equivalent
  --env-file PATH             Specify an alternate environment file

Commands:
  build              Build or rebuild services
  config             Validate and view the Compose file
  create             Create services
  down               Stop and remove containers, networks, images, and volumes
  events             Receive real time events from containers
  exec               Execute a command in a running container
  help               Get help on a command
  images             List images
  kill               Kill containers
  logs               View output from containers
  pause              Pause services
  port               Print the public port for a port binding
  ps                 List containers
  pull               Pull service images
  push               Push service images
  restart            Restart services
  rm                 Remove stopped containers
  run                Run a one-off command
  scale              Set number of containers for a service
  start              Start services
  stop               Stop services
  top                Display the running processes
  unpause            Unpause services
  up                 Create and start containers
  version            Show the Docker-Compose version information
```

docker-compose --help will provide you additional information about arguments and implementation details of the command.

docker-compose build

This command builds images of the mentioned services in the docker-compose.yml file for which a Dockerfile is provided.

Carefully read the statement above. The job of the 'build' command is to get the images ready to create containers. If a service is using the prebuilt image, it will skip that service.

```
$ docker-compose build
database uses an image, skipping
Building web
```

```
Step 1/11 : FROM python:3.9-rc-buster
 ---> 2e0edf7d3a8a
Step 2/11 : RUN apt-get update && apt-get install -y docker.io
```

docker-compose images

This command lists images built using the current docker-compose file.

```
docker-compose images
          Container                  Repository        Tag       Image Id
Size
------------------------------------------------------------------------------
-----
7001788f31a9_docker_database_1   mysql/mysql-server   5.7      2a6c84ecfcb2
333.9 MB
docker_database_1                mysql/mysql-server   5.7      2a6c84ecfcb2
333.9 MB
docker_web_1                     <none>               <none>   d986d824dae4   953
MB
```

docker-compose run

Similar to docker run command, this one creates containers from images built for the services mentioned in the compose file. It runs a specific service provided as an argument to the command

```
docker-compose run web
Starting 7001788f31a9_docker_database_1 ... done
 * Serving Flask app "app.py" (lazy loading)
 * Environment: development
 * Debug mode: on
 * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 116-917-688
```

If you look at the output closely, you'll notice that the database service also started without being mentioned in the command. That's because the web service is dependent on the database service. So, it will start all the dependent services and then, the mentioned service.

docker-compose up

This does the job of the docker-compose build and docker-compose run commands. It initially builds the images if they are not located locally and then starts the containers. If images are already built, it will fork the container directly. We can force it to rebuild the image by adding a --build argument.

```
$ docker-compose up
Creating docker_database_1 ... done
Creating docker_web_1      ... done
Attaching to docker_database_1, docker_web_1
database_1  | [Entrypoint] MySQL Docker Image 5.7.29-1.1.15
database_1  | [Entrypoint] Initializing database
web_1       |  * Serving Flask app "app.py" (lazy loading)
web_1       |  * Environment: development
web_1       |  * Debug mode: on
web_1       |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1       |  * Restarting with stat
web_1       |  * Debugger is active!
web_1       |  * Debugger PIN: 855-188-665
database_1  | [Entrypoint] Database initialized
database_1  | Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time
zone. Skipping it.
```

docker-compose stop

This command stops the running containers of the specified services in the docker-compose file.

```
$ docker-compose stop
Stopping docker_web_1      ... done
Stopping docker_database_1 ... done
```

docker-compose rm

This command removes the containers of the services or the containers created using the current docker-compose file. It can be containers created using the docker-compose run command or the docker-compose up command. It will remove all the containers which have services mentioned in the docker-compose file.

```
$ docker-compose rm
Going to remove docker_web_1, docker_database_1
Are you sure? [yN] y
Removing docker_web_1      ... done
```

```
Removing docker_database_1 ... done
```

docker-compose start

This command starts any stopped containers of the services. If all the containers are already up and running, they will just inform that all containers are starting and exit with 0 status.

```
$ docker-compose start
Starting database ... done
Starting web      ... done
```

docker-compose restart

This command restarts all the containers of the services.

```
$ docker-compose restart
Restarting docker_web_1      ... done
Restarting docker_database_1 ... done
```

docker-compose ps

This lists all the containers for services mentioned in the current docker-compose file. The containers can either be running or stopped.

```
$ docker-compose ps
      Name                 Command             State               Ports
-------------------------------------------------------------------------------
docker_database_1   /entrypoint.sh mysqld   Up (healthy)   3306/tcp, 33060/tcp
docker_web_1        flask run               Up             0.0.0.0:5000->5000/tcp

$ docker-compose ps
      Name                 Command          State    Ports
-----------------------------------------------------------
docker_database_1   /entrypoint.sh mysqld   Exit 0
docker_web_1        flask run               Exit 0
```

docker-compose down

This command is similar to docker system prune. However, there is a little difference. It stops all the services and then cleans up the containers, networks and images used and created by the compose file services.

```
$ docker-compose down
Removing docker_web_1      ... done
Removing docker_database_1 ... done
Removing network docker_default
sangam-MacBook-Air:Docker venkateshachintalwar$ docker-compose images
Container   Repository   Tag   Image Id   Size
------------------------------------------------
sangam-MacBook-Air:Docker sangam$ docker-compose ps
Name   Command   State   Ports
------------------------------
```

docker-compose logs

This command is similar to docker logs . The little difference is this prints all the logs created by all the services. We can also use the -f argument to see real-time logs.

```
$ docker-compose logs
Attaching to docker_web_1, docker_database_1
database_1  | [Entrypoint] MySQL Docker Image 5.7.29-1.1.15
database_1  | [Entrypoint] Initializing database
database_1  | [Entrypoint] Database initialized
database_1  | Warning: Unable to load '/usr/share/zoneinfo/iso3166.tab' as time
zone. Skipping it.
database_1  | Warning: Unable to load '/usr/share/zoneinfo/leapseconds' as time
zone. Skipping it.
web_1       |  * Serving Flask app "app.py" (lazy loading)
web_1       |  * Environment: development
web_1       |  * Debug mode: on
web_1       |  * Running on http://0.0.0.0:5000/ (Press CTRL+C to quit)
web_1       |  * Restarting with stat
web_1       |  * Debugger is active!
web_1       |  * Debugger PIN: 290-944-777
```