Compose File Syntax

A docker-compose.yml file is organized into four sections:

| Directive | Use |
| --- | --- |
| version | Specifies the Compose file syntax version. This guide will use Version 3 throughout. |
| services | In Docker a service is the name for a [“Container in production”](https://docs.docker.com/get-started/part3/#introduction). This section defines the containers that will be started as a part of the Docker Compose instance. |
| networks | This section is used to configure networking for your application. You can change the settings of the default network, connect to an external network, or define app-specific networks. |
| volumes | Mounts a linked path on the host machine that can be used by the container. |

Most of this guide will focus on setting up containers using the services section. Here are some of the common directives used to set up and configure containers:

| Directive | Use |
| --- | --- |
| image | Sets the image that will be used to build the container. Using this directive assumes that the specified image already exists either on the host or on [Docker Hub](https://hub.docker.com/). |
| build | This directive can be used instead of image. Specifies the location of the Dockerfile that will be used to build this container. |
| db | In the case of the example Dockercompose file, db is a variable for the container you are about to define. |
| restart | Tells the container to restart if the system restarts. |
| volumes | Mounts a linked path on the host machine that can be used by the container |
| environment | Define environment variables to be passed in to the Docker run command. |
| depends\_on | Sets a service as a dependency for the current block-defined container |
| port | Maps a port from the container to the host in the following manner: host:container |
| links | Link this service to any other services in the Docker Compose file by specifying their names here. |

Many other configuration directives are available. See the [Compose File reference](https://docs.docker.com/compose/compose-file) for details.

**Caution**

The example docker-compose.yml above uses the environment directive to store MySQL user passwords directly in the YAML file to be imported into the container as environment variables. This is not recommended for sensitive information in production environments. Instead, sensitive information can be stored in a separate .env file (which is not checked into version control or made public) and accessed from within docker-compose.yml by using the env\_file directive.

Build an Application from Scratch

Create a docker-compose.yml file one section at a time to illustrate the steps of building a multi-container application.

Define a Simple Service:

1. Create a new docker-compose.yml in a text editor and add the following content:

**docker-compose.yml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | version: '3'  services:  distro:  image: alpine  restart: always  container\_name: Alpine\_Distro  entrypoint: tail -f /dev/**null** |

Each entry in the services section will create a separate container when docker-compose is run. At this point, the section contains a single container based on the official Alpine distribution:

* + The restart directive is used to indicate that the container should always restart (after a crash or system reboot, for example).
  + The container\_name directive is used to override the randomly generated container name and replace it with a name that is easier to remember and work with.
  + Docker containers exit by default if no process is running on them. tail -f is an ongoing process, so it will run indefinitely and prevent the container from stopping. The default entrypoint is overridden to keep the container running.

1. Bring up your container:
2. docker-compose up -d
3. Check the status of your container:
4. docker ps

The output should resemble the following:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

967013c36a27 alpine "tail -f /dev/null" 3 seconds ago Up 2 seconds Alpine\_Distro

1. Bring down the container:
2. docker-compose down

Add Additional Services

From here you can begin to build an ecosystem of containers. You can define how they work together and communicate.

1. Reopen docker-compos.yml and add the database service below:

**docker-compose.yml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | version: '3'  services:  distro:  image: alpine  container\_name: Alpine\_Distro  restart: always  entrypoint: tail -f /dev/**null**  database:  image: postgres:latest  container\_name: postgres\_db  volumes:  - ../dumps:/tmp/  ports:  - "5432:5432" |

There are now two services defined:

* + Distro
  + Database

The Distro service is the same as before. The Database server contains the instructions for a postgres container, and the directives: volumes: - ../dumps:/tmp and ports:-"5432:5432", the first directive maps the containerd /dumps folder to our local /tmp folder. The second directive maps the containers ports to the local host’s ports.

1. Check the running containers:
2. docker ps

This command shows the status of the containers, the port mapping, the names, and the last command running on them. It’s important to note that the postgres container reads “docker-entrypoint…” under commands. The Postgres [Docker Entrypoint](https://github.com/docker-library/postgres/blob/master/docker-entrypoint.sh) script is the last thing that launches when the container starts.

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

ecc37246f6ef postgres:latest "docker-entrypoint..." About a minute ago Up About a minute 0.0.0.0:5432->5432/tcp postgres\_db

35dab3e712d6 alpine "tail -f /dev/null" About a minute ago Up About a minute Alpine\_Distro

1. Bring down both containers:
2. docker-compose down

Add an nginx Service

1. Add an nginx container so that your application will be able to serve websites:

**docker-compose.yml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | version: '3'  services:  distro:  image: alpine  container\_name: Alpine\_Distro  restart: always  entrypoint: tail -f /dev/**null**  database:  image: postgres:latest  container\_name: postgres\_db  volumes:  - ../dumps:/tmp/  ports:  - "5432:5432"  web:  image: nginx:latest  container\_name: nginx  volumes:  - ./mysite.template:/etc/nginx/conf.d/mysite.template  ports:  - "8080:80"  environment:  - NGINX\_HOST=example.com  - NGINX\_port=80  links:  - database:db  - distro |

This docker-compose file contains some new directives: *environment* and *links*. The first directive sets runtime level options within the container. linkscreates a dependency network between the containers. The nginx container depends on the other two to execute. In addition, the corresponding containers will be reachable at a hostname indicated by the alias. In this case, pinging db from the web container will reach the database service. While you do not need the links directive for the containers to talk with each other, links can serve as a failsafe when starting the docker-compose application.

1. Start Docker Compose and check the container status:
2. docker-compose up -d
3. docker ps

The output should be similar to:

CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

55d573674e49 nginx:latest "nginx -g 'daemon ..." 3 minutes ago Up 3 minutes 0.0.0.0:8080->80/tcp nginx

ad9e48b2b82a alpine "tail -f /dev/null" 3 minutes ago Up 3 minutes Alpine\_Distro

736cf2f2239e postgres:latest "docker-entrypoint..." 3 minutes ago Up 3 minutes 0.0.0.0:5432->5432/tcp postgres\_db

1. Test nginx by navigating to your Linode’s public IP address, port 8080 in a browser (for example 192.0.2.0:8080). You should see the default nginx landing page displayed.

Persistent Data Storage[Permalink](https://www.linode.com/docs/applications/containers/how-to-use-docker-compose/#persistent-data-storage)

Storing PostgreSQL data directly inside a container is not recommended. Docker containers are intended to be treated as ephemeral: your application’s containers are built from scratch when running docker-compose up and destroyed when running docker-compose down. In addition, any unexpected crash or restart on your system will cause any data stored in a container to be lost.

For these reasons it is important to set up a persistent volume on the host that the database containers will use to store their data.

1. Add a volumes section to docker-compose.yml and edit the database service to refer to the volume:

**docker-compose.yml**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | version: '3'  services:  distro:  image: alpine  container\_name: Alpine\_Distro  restart: always  entrypoint: tail -f /dev/**null**  database:  image: postgres:latest  container\_name: postgres\_db  volumes:  - data:/var/lib/postgresql  ports:  - "5432:5432"  web:  image: nginx:latest  container\_name: nginx  volumes:  - ./mysite.template:/etc/nginx/conf.d/mysite.template  ports:  - "8080:80"  environment:  - NGINX\_HOST=example.com  - NGINX\_port=80  links:  - database:db  - distro  volumes:  data:  external: **true** |

1. external: true tells Docker Compose to use a pre-existing external data volume. If no volume named data is present, starting the application will cause an error. Create the volume:
2. docker volume create --name=data
3. Start the application as before:
4. docker-compose up -d

Next Steps

Docker Compose is a powerful tool for orchestrating sets of containers that can work together. Things like an app or a development environment can utilize Docker-compose. The result is a modular and configurable environment that can be deployed anywhere.