Docker Compose evolution

This lab will walk through adding additional services and features to a simple Docker Compose file. You will start off with a simple compose file that launches the `Alpine` Linux distribution container, then build on that by adding additional containers, volumes and networks.

Single Service Compose file

This simple Compose file creates and launches an Alpine Linux distro container.

`compose/docker-compose.yml`

```
version: '3'

services:
  distro:
    image: alpine
    restart: always
    container_name: Alpine_Distro
    entrypoint: tail -f /dev/null
```

Now let's create the container. In the `compose` directory run

```
docker-compose up -d
```

After the container starts connect to it and confirm everything looks good

```
docker exec -it Alpine_Distro /bin/ash
```

C

Add Postgres container

Now we'll add an additional container with some options. Review the updated Compose file

`multi-compose/docker-compose.yml`

```
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"
```

We added a `database` container using the `postgres` image and mounted `/tmp` from our Docker host to `dumps` on the container. We also mapped port `5432` from host to Docker container.

To start this run

```
docker-compose -f multi-compose/docker-compose.yml up -d
```

After `docker-compose` completes confirm all containers are running

```
docker-compose -f multi-compose/docker-compose.yml ps
```

Should see something like

```
    Name                   Command              State           Ports
-------------------------------------------------------------------------------
Alpine_Distro   tail -f /dev/null                Up
postgres_db     docker-entrypoint.sh postgres    Up      0.0.0.0:5432->5432/tcp
```

Cleanup

```
docker-compose -f multi-compose/docker-compose.yml stop
docker-compose -f multi-compose/compose/docker-compose.yml rm -f
```

Add Nginx

Let's go ahead and add a web container as well.

Review the updated Compose file

`fullstack-compose/docker-compose.yml`

```
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. `links` creates 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. Start Docker Compose and check the container status:

```
docker-compose -f fullstack-compose/docker-compose.yml up -d
```

To confirm containers are running

```
docker-compose -f fullstack-compose/docker-compose.yml ps
```

This should show something like the following

```
    Name                   Command              State           Ports
-------------------------------------------------------------------------------
Alpine_Distro   tail -f /dev/null                Up
nginx           nginx -g daemon off;             Up      0.0.0.0:8080->80/tcp
postgres_db     docker-entrypoint.sh postgres    Up      0.0.0.0:5432->5432/tcp
```

Now confirm the web server is reachable by visiting `http://localhost:8080` in a browser, or curling it.

```
curl http://localhost:8080
```

You should see the default Nginx web page.

Cleanup

```
docker-compose -f fullstack-compose/docker-compose.yml stop
docker-compose -f fullstack-compose/compose/docker-compose.yml rm -f
```

Add volume

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.

Review the updated compose file paying attention to the new `volumes` section.
`volumes-compose/docker-compose.yml`

```
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
```

`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:

```
docker volume create --name=data
```

Let's go ahead and start it all up.

```
docker-compose -f volume-compose/docker-compose.yml up -d
```

Now confirm the web server is reachable by visiting `http://localhost:8080` in a browser, or curling it.

```
curl http://localhost:8080
```

These simple compose files show the foundation of Compose syntax which can be used to build more advanced configurations.

Cleanup

```
docker-compose -f fullstack-compose/docker-compose.yml stop
```

```
docker-compose -f fullstack-compose/compose/docker-compose.yml rm -f
```

Lab Complete