**DevOps and Cloud**

**March 2025**



# Advanced Containerization Concepts

Homework (M2)

Vasil Atanasov

@VasAtanasov

## Summary

To repeat the steps from the practice, I will use a PostgreSQL Docker image. One key point, which will become evident in the steps involving volumes, is that each PostgreSQL container instance sharing a common data volume with another instance will cause the PostgreSQL server to detect this and create a copy of the volume or fail. If the database is modified by one server, the other server will not see the changes. To see the changes a custom docker image where only PostgreSQL client is installed whit which we will connect to the remote server will be queried.

Postgres Docker Image: <https://github.com/docker-library/postgres>

## Network and Volumes

### Networks

#### Default Network

Containers on the default bridge network cannot resolve each other by container name, unlike user-defined bridge networks.

docker network ls

docker network inspect bridge

docker run -d \

--name postgres1 \

-e POSTGRES\_PASSWORD=Password1 \

-e POSTGRES\_DB=bulgaria \

-p 5432:5432 \

postgres

docker run -d \

--name postgres2 \

-e POSTGRES\_PASSWORD=Password1 \

-e POSTGRES\_DB=bulgaria \

-p 5433:5432 \

postgres

docker network inspect bridge -f '{{range .Containers}}{{.Name}}, {{println .IPv4Address}}{{end}}'

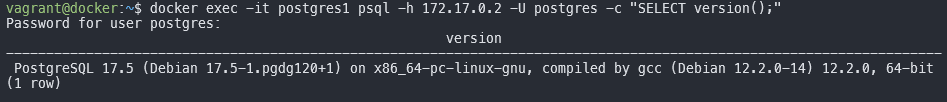
A screenshot of a computer

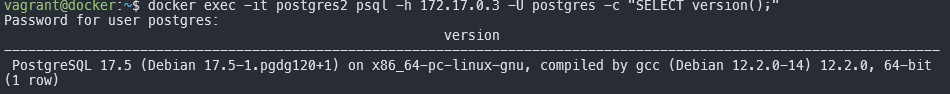
AI-generated content may be incorrect.

We can query for the version of the PostgreSQL server running on postgres1 from postgres2 container using only IP address of the container because default bridge network does not support name resolution.

docker exec -it postgres1 psql -h 172.17.0.2 -U postgres -c "SELECT version();"

docker exec -it postgres2 psql -h 172.17.0.3 -U postgres -c "SELECT version();"





If we try to use the container name as host, we will get an error.



#### Custom Network

Now we will create a custom bridge network **pg-net.**

docker network create -d bridge --subnet 10.0.0.0/24 pg-net

The two PostgreSQL containers are running so we can connect them to the new network.

docker network connect pg-net postgres1

docker network connect pg-net postgres2

Or we can stop them and start them again with the –network option passing the name of the network.

docker run -d \

--name postgres1 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

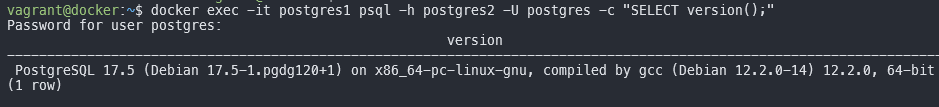
-p 5432:5432 \

postgres

docker network inspect pg-net -f '{{range .Containers}}{{.Name}}, {{println .IPv4Address}}{{end}}'

A computer screen with white text

AI-generated content may be incorrect.

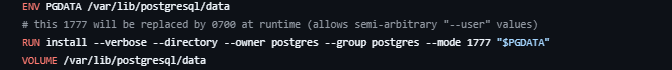


### Volumes

#### On The Fly

Note: Postgres Dockerfile has an instruction to create a volume that is why there is a volume already created on the fly when run.

<https://github.com/docker-library/postgres/blob/38b3c10a487945e08b7f63dee25dc4f7b86a79d1/Dockerfile-debian.template#L200>



That is why when the image is run it will create a volume on the fly.

A screenshot of a computer program

AI-generated content may be incorrect.

We will populate the database and start a second container using the volume from the first container.

docker exec -i postgres1 psql -U postgres -d bulgaria <<-EOSQL

CREATE TABLE cities (

id SERIAL PRIMARY KEY,

city\_name VARCHAR(50),

population INT

);

INSERT INTO cities (city\_name, population) VALUES ('София', 1248452);

INSERT INTO cities (city\_name, population) VALUES ('Пловдив', 343070);

INSERT INTO cities (city\_name, population) VALUES ('Варна', 332686);

INSERT INTO cities (city\_name, population) VALUES ('Бургас', 199571);

INSERT INTO cities (city\_name, population) VALUES ('Русе', 137533);

INSERT INTO cities (city\_name, population) VALUES ('Стара Загора', 124599);

INSERT INTO cities (city\_name, population) VALUES ('Плевен', 93214);

INSERT INTO cities (city\_name, population) VALUES ('Сливен', 83740);

INSERT INTO cities (city\_name, population) VALUES ('Добрич', 79269);

INSERT INTO cities (city\_name, population) VALUES ('Шумен', 72342);

EOSQL

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

We can see still only one volume exists.

If we query the database, we will get:

docker exec -it postgres2 psql \

-h postgres1 \

-U postgres \

-d bulgaria \

-c "SELECT \* FROM cities;"

A screenshot of a computer

AI-generated content may be incorrect.

#### Attach a Prepopulated (Existing) Folder

Due to the nature of Postgres, shared data directory is not allowed, and Postgres makes a copy of the data. If a row is deleted in the first container database, it will not affect the database of the second container. It is recommended to connect to the database container as a client. This way only one server manages the db. That is why we will run the first container, populate it with data and then run the second container. When we query the database in the second container we will see there is data there.

docker run -d \

--name postgres1 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5432:5432 \

-v $(pwd)/postgres-data:/var/lib/postgresql/data \

postgres

docker exec -i postgres1 psql -U postgres -d bulgaria <<-EOSQL

CREATE TABLE cities (

id SERIAL PRIMARY KEY,

city\_name VARCHAR(50),

population INT

);

INSERT INTO cities (city\_name, population) VALUES ('София', 1248452);

INSERT INTO cities (city\_name, population) VALUES ('Пловдив', 343070);

INSERT INTO cities (city\_name, population) VALUES ('Варна', 332686);

INSERT INTO cities (city\_name, population) VALUES ('Бургас', 199571);

INSERT INTO cities (city\_name, population) VALUES ('Русе', 137533);

INSERT INTO cities (city\_name, population) VALUES ('Стара Загора', 124599);

INSERT INTO cities (city\_name, population) VALUES ('Плевен', 93214);

INSERT INTO cities (city\_name, population) VALUES ('Сливен', 83740);

INSERT INTO cities (city\_name, population) VALUES ('Добрич', 79269);

INSERT INTO cities (city\_name, population) VALUES ('Шумен', 72342);

EOSQL

docker run -d \

--name postgres2 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5433:5432 \

-v $(pwd)/postgres-data:/var/lib/postgresql/data \

postgres

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A computer screen shot of a computer code

AI-generated content may be incorrect.

#### Dedicated Volume

docker volume create postgres-data

A screen shot of a computer

AI-generated content may be incorrect.

As before we will start the first PostgreSQL instance populate the data and then start the second instance.

docker run -d \

--name postgres1 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5432:5432 \

-v postgres-data:/var/lib/postgresql/data \

postgres

Population of data omitted for brevity.

docker run -d \

--name postgres2 \

--network pg-net \

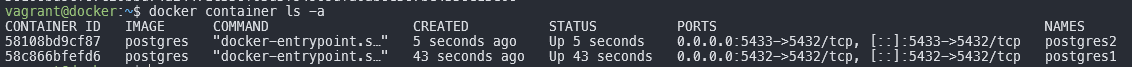
-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5433:5432 \

-v postgres-data:/var/lib/postgresql/data \

postgres



A screenshot of a computer

AI-generated content may be incorrect.

#### Volume Containers

To be useful the database volume we will start the PostgreSQL server, populate it with data and then stop it and use its volume to start other instances.

docker run -d \

--name bulgaria-db-base \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5431:5432 \

-v postgres-data:/var/lib/postgresql/data \

postgres

docker container stop bulgaria-db-base

Population of data omitted for brevity.

docker run -d \

--name postgres1 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5432:5432 \

--volumes-from bulgaria-db-base \

postgres

## Custom Container Images

### Create image from Dockerfile and add content

For this step we will use the files in the *bulgaria-db* folder. Inside there is a Dockerfile which will extend postgres docker image and using custom population script to populate the database with data on run.

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer screen

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

#### Inspecting the image with Dive

Repo: <https://github.com/wagoodman/dive>

dive bulgaria-db

We can see layers information, how the content of the image changes, details about the layer etc.

A screenshot of a computer screen

AI-generated content may be incorrect.

### Publishing an image

docker login

A screenshot of a computer error

AI-generated content may be incorrect.

docker image tag bulgaria-db vasatanasov/bulgaria-db:demo

docker image push vasatanasov/bulgaria-db:demo

A screenshot of a computer

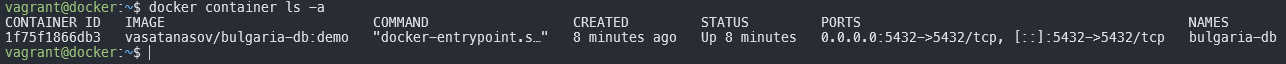
AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

### ENTRYPOINT and CMD

For the demonstration of how ENTRYPOINT and CMD work together I will create a custom Dockerfile with which will prepare an image that will act as executable. The application will query the remote server which will be running our bulgari-db and search for a city by its name and return results on the command line. We will assume that the remote database is up and running.



Files for the custom docker image are in the *city-search* folder.

A screenshot of a computer

AI-generated content may be incorrect.

We will assume that pg-net exist and the bulgaria-db container is connected to it.

docker build -f /vagrant/city-search/Dockerfile \

-t city-search \

/vagrant/city-search

A computer screen shot of a computer screen

AI-generated content may be incorrect.

We can search for the exact name.

docker run -it --rm --network pg-net city-search "Шумен"

A match.

docker run -it --rm --network pg-net city-search "C"

When no argument is passed all results are returned.

docker run -it --rm --network pg-net city-search

A screenshot of a computer program

AI-generated content may be incorrect.

То use it as executable we can create e bash script (*city-search.sh*) and place it in */usr/local/bin* so we can have it as a bash command.

#!/bin/bash  
  
docker run -it --rm --network pg-net city-search "$1"

A screen shot of a computer

AI-generated content may be incorrect.

### Create image from container

We will run our city database and add new city to it.

docker run -d \

--name bulgaria-db \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5432:5432 \

vasatanasov/bulgaria-db:demo

docker exec -it bulgaria-db bash

Insert new city in the create-database.sh script and exit Ctrl-PQ

vi docker-entrypoint-initdb.d/create-database.sh

INSERT INTO cities (city\_name, population) VALUES ('Благоевград', 172446);

Commit the container with the updated database.

docker container commit --author "SoftUni Student Vasil Atanasov" \

bulgaria-db \

bulgaria-db-v2

docker image ls bulgaria-db-v2

Run the container.

docker run -d \

--name bulgaria-db-v2 \

--network pg-net \

-e POSTGRES\_DB=bulgaria \

-e POSTGRES\_PASSWORD=Password1 \

-p 5433:5432 \

bulgaria-db-v2=

A screen shot of a computer

AI-generated content may be incorrect.

A computer screen with white text

AI-generated content may be incorrect.

### Create image with heredoc

We will create e docker image from postgres which will have an empty cities table.

docker image build -t bulgaria-db - << 'EOF'

ARG BASE\_IMAGE\_TAG=latest

FROM postgres:${BASE\_IMAGE\_TAG}

RUN printf '#!/bin/bash\npsql -U postgres <<EOFSQL\nCREATE TABLE IF NOT EXISTS cities (\n id SERIAL PRIMARY KEY,\n city\_name TEXT,\n population INTEGER\n);\nEOFSQL\n' \

> /docker-entrypoint-initdb.d/create-database.sh && chmod +x /docker-entrypoint-initdb.d/create-database.sh

EOF

docker run -d --name bulgaria-db -e POSTGRES\_PASSWORD=Password1 bulgaria-db

docker exec -it bulgaria-db psql -U postgres -d postgres -c 'SELECT \* FROM cities;'

A screenshot of a computer program

AI-generated content may be incorrect.

### Archive and transfer images

A screenshot of a computer

AI-generated content may be incorrect.

### Archive and transfer containers

docker container export -o bulgaria-db.tar vasatanasov/bulgaria-db:demo

Using docker image import, creates a new image from a tarball (usually from docker export) and does not preserve metadata such as the user (USER postgres), environment variables, or entrypoint from the original image. That is why we need specify all on the command line.

This example is not complete because the postgres Dockerfile have more envs.

docker image import \

bulgaria-db.tar \

--change 'CMD ["postgres"]' \

--change 'USER postgres' \

new-bulgaria-db

## Custom Apache Web Server Docker Image

The files of the task are in the *apache* folder.

A screenshot of a computer

AI-generated content may be incorrect.

FROM almalinux:9  
  
RUN dnf -y update && \  
 dnf -y install httpd && \  
 dnf clean all  
  
COPY index.html /var/www/html/index.html  
  
EXPOSE 80  
  
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

A computer screen with white text

AI-generated content may be incorrect.

## Best Practices and Troubleshooting

We will base the following steps on the custom Apache Web Server docker image.

Files are located under the *best-practice-and-troubleshooting* folder.

### Best Practices

#### Provide Details via Labels

FROM almalinux:9

LABEL version="1.0" \

description="A sample web application that displays It Works"

RUN dnf -y update && \

dnf -y install httpd && \

dnf clean all

COPY index.html /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

docker build \

-f /vagrant/best-practice-and-troubleshooting/label.Dockerfile -t web \

/vagrant/best-practice-and-troubleshooting

docker image inspect --format='{{json .Config.Labels}}' labels

A screen shot of a computer

AI-generated content may be incorrect.

#### Use the Right Base Image

FROM almalinux:9.6-minimal

RUN microdnf -y update && \

microdnf -y install httpd && \

microdnf clean all

COPY index.html /var/www/html/index.html

EXPOSE 80

CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

docker build \

-f /vagrant/best-practice-and-troubleshooting/minimal.Dockerfile \

-t web-minimal \

/vagrant/best-practice-and-troubleshooting

A black background with white text

AI-generated content may be incorrect.

We can easily see the difference in size between the two images.

#### Push Readme with the Image

Installing pushrm

install\_pushrm() {

log "Installing Docker Push Readme"

local version

version=$(curl -sL "https://api.github.com/repos/christian-korneck/docker-pushrm/releases/latest" | grep '"tag\_name":' | sed -E 's/.\*"v([^"]+)".\*/\1/')

curl -fLo docker-pushrm "https://github.com/christian-korneck/docker-pushrm/releases/download/v${version}/docker-pushrm\_linux\_amd64"

chmod +x docker-pushrm

mkdir -p "/usr/local/lib/docker/cli-plugins"

mv docker-pushrm "/usr/local/lib/docker/cli-plugins/"

}



docker image tag web-minimal:latest vasatanasov/web-minimal:demo

docker image push vasatanasov/web-minimal:demo

docker pushrm \

--file /vagrant/best-practice-and-troubleshooting/README.md \

vasatanasov/web-minimal:demo

<https://hub.docker.com/repository/docker/vasatanasov/web-minimal/general>

A screenshot of a computer

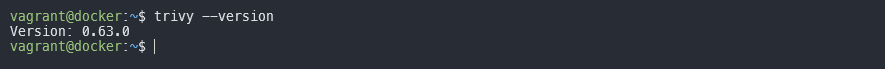
AI-generated content may be incorrect.

#### Scan Images

Installing **Trivy**

install\_trivy() {  
 log "Installing Trivy"

curl -sfL https://raw.githubusercontent.com/aquasecurity/trivy/main/contrib/install.sh | sh -s -- -b "$BIN\_DIR" latest  
}



trivy image vasatanasov/web-minimal:demo

A screen shot of a computer

AI-generated content may be incorrect.

#### Use a Linter

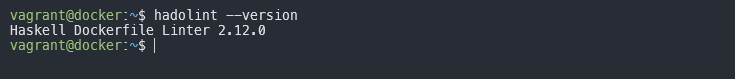
Installing **Hadolint**

install\_hadolint() {  
 log "Installing Hadolint"

local version

version=$(curl -sL "https://api.github.com/repos/hadolint/hadolint/releases/latest" | grep '"tag\_name":' | sed -E 's/.\*"v([^"]+)".\*/\1/')  
 curl -fLo hadolint "https://github.com/hadolint/hadolint/releases/download/v${version}/hadolint-Linux-x86\_64"

chmod +x hadolint mv hadolint "$BIN\_DIR/"  
}



hadolint /vagrant/best-practice-and-troubleshooting/minimal.Dockerfile



The fix:

FROM almalinux:9.6-minimal  
  
RUN microdnf -y update && \  
 microdnf -y install httpd-2.4.62-4.el9 && \  
 microdnf clean all  
  
COPY index.html /var/www/html/index.html  
  
EXPOSE 80  
  
CMD ["/usr/sbin/httpd", "-D", "FOREGROUND"]

hadolint /vagrant/best-practice-and-troubleshooting/linter-fix.Dockerfile

This outputs no errors.

#### Multi-stage Build

This example of Dockerfile builds a custom Keycloak image using a multi-stage approach. It installs extra tools like jq and curl from a minimal UBI 9 image, builds Keycloak with custom providers and config in a builder stage, and combines everything into a clean, secure final image. This setup keeps the final image lightweight and production ready. This Dockerfile I personally created and use in development.

FROM registry.access.redhat.com/ubi9 AS ubi-micro-build

RUN mkdir -p /mnt/rootfs && \

dnf install --installroot=/mnt/rootfs jq tzdata curl --releasever=9 \

--setopt=install\_weak\_deps=false --nodocs -y && \

dnf --installroot=/mnt/rootfs clean all

FROM quay.io/keycloak/keycloak:${BASE\_IMAGE\_TAG} AS builder

ENV KC\_DB=postgres

COPY keycloak/providers/ /opt/keycloak/providers/

COPY conf/keycloak.conf /opt/keycloak/conf/

RUN /opt/keycloak/bin/kc.sh build

FROM quay.io/keycloak/keycloak:${BASE\_IMAGE\_TAG}

COPY --from=builder /opt/keycloak/ /opt/keycloak/

COPY --from=ubi-micro-build /mnt/rootfs/ /

USER root

COPY scripts/ /scripts/

RUN chmod +x /scripts/keycloak/\*.sh

USER 1000

WORKDIR /opt/keycloak

ENV KEYCLOAK\_HOME=/opt/keycloak

ENV PATH=$PATH:$KEYCLOAK\_HOME/bin

### Troubleshooting

docker container logs web-minimal