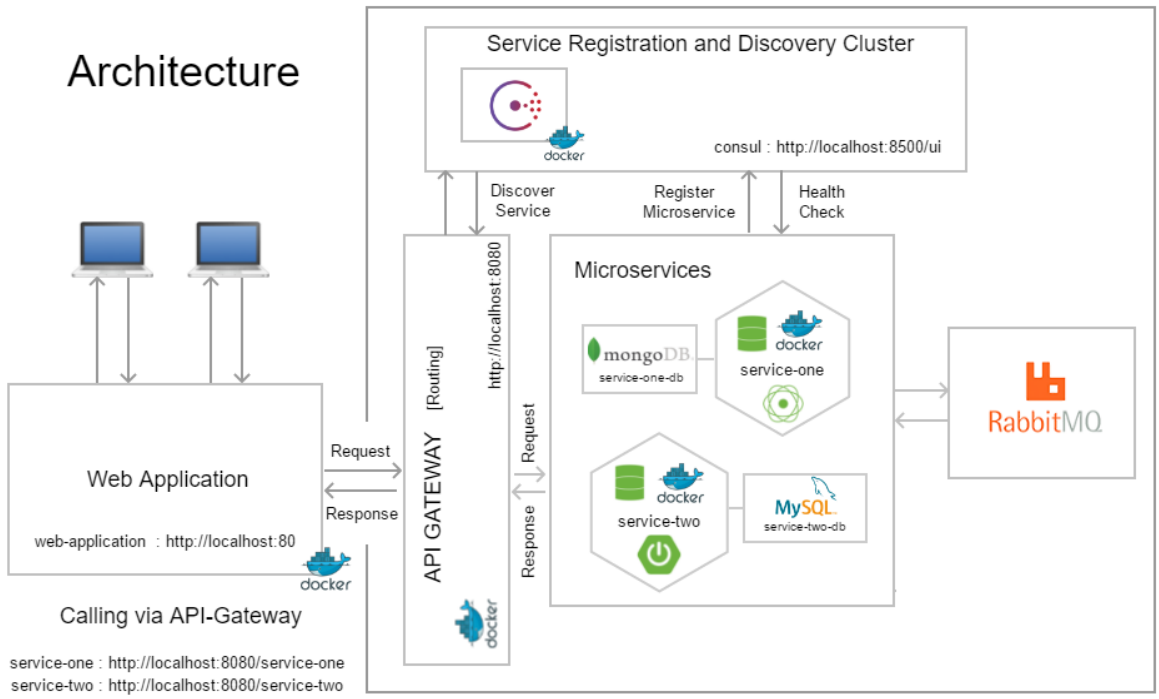
Mastering Docker

Part 1: Overview

Use case

Dockerize an application

Application architecture



Technology

- Spring Boot - Application framework
- Zuul - API Gateway (Load Balancer)
- Consul - Service registration and Discovery
- RabbitMQ - asynchronous microservices messaging.
- Angular, Bootstrap & jQuery - HTML enhanced for web apps!
- Swagger - API documentation

Tools

Ensure your machine installed:

- Git
- JDK8
- Maven
- NodeJS

- Docker Desktop

Part 2: Create Docker images

Step 1: Clone the repository

Repository:

Stay on checked out folder and execute following command

Step 2: Build web-application image

Create Dockerfile

```
$ cd application
$ mvn clean package
$ cd web-application
$ nano Dockerfile
```

Write this content to the file

```
# Create image based on the official Node 12 image from dockerhub
FROM node:12

# Create a directory where our app will be placed
RUN mkdir -p /usr/src/app

# Change directory so that our commands run inside this new directory
WORKDIR /usr/src/app

# Get all the code needed to run the app
COPY . /usr/src/app

# Install dependecies
RUN npm install

# Expose the port the app runs in
EXPOSE 4200

# Serve the app
CMD ["npm", "start"]
```

Build the image

```
$ docker build -t build_web-application .
```

Step 3: Build service-two image

Create Dockerfile

```
$ cd ../service-two
$ nano Dockerfile
```

Write this content to the file

```
FROM openjdk:8-jre

# environment
EXPOSE 8084

# executable ADD @project.artifactId@-@project.version@.jar app.jar
ADD target/service-two.jar app.jar

# run app as user 'booter'
RUN /bin/sh -c 'touch /app.jar'
ENTRYPOINT ["java", "-Xmx256m", "-Xss32m", "-
Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Build the image

```
$ docker build -t build_service-two:latest .
```

Step 4: Build service-one image

Create Dockerfile

```
$ cd ../service-one
$ nano Dockerfile
```

Write this content to the file

```
FROM openjdk:8-jdk

# environment
EXPOSE 8082

# executable ADD @project.artifactId@-@project.version@.jar app.jar
ADD target/service-one.jar app.jar

# run app as user 'booter'
RUN /bin/sh -c 'touch /app.jar'
ENTRYPOINT ["java", "-Xmx256m", "-Xss32m", "-
Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Build the image

```
$ docker build -t build_service-one:latest .
```

Step 5: Build api-gateway image

Create Dockerfile

```
$ cd ../api-gateway
$ nano Dockerfile
```

Write this content to the file

```
FROM openjdk:8-jre-alpine

# environment
EXPOSE 8080

# executable ADD @project.artifactId@-@project.version@.jar app.jar
ADD target/api-gateway.jar app.jar

# run app as user 'booter'
RUN /bin/sh -c 'touch /app.jar'
ENTRYPOINT ["java", "-Xmx256m", "-Xss32m", "-
Djava.security.egd=file:/dev/./urandom","-jar","/app.jar"]
```

Build the image

```
$ docker build -t build_api-gateway:latest .
```

Part 3: Create Docker networks and volumes

Step 1: Create Docker networks

```
$ docker network create -d bridge build_backend
$ docker network create -d bridge build_frontend
```

Step 2: Create Docker volumes

```
$ docker volume create build_mongodata
```

Part 4: Run Docker containers

Step 1: Run services database

```
$ docker run -d --tty --init --privileged \
             -p 3310:3306 \
             --expose 3310 \
             --network build_backend \
             --hostname servicetwodb \
             --name service-two-db \
             --env MYSQL_ROOT_PASSWORD=root123 \
             --env MYSQL_DATABASE=service-two \
             --env MYSQL_USER=service-two \
             --env MYSQL_PASSWORD=service-two \
             mysql/mysql-server:5.7
```

```
$ docker run -d --tty --init --privileged \
             -p 27017:27017 \
             --expose 27017 \
             --network build_backend \
             --hostname serviceonedb \
             --name service-one-db \
             --env MONGODB_USER="service-one" \
             --env MONGODB_PASS="service-one" \
             --env MONGO_DATA_DIR=/data/db \
             --env MONGO_LOG_DIR=/dev/null \
             --volume build_mongodata:/data/db \
             mongo:3.7 mongod --smallfiles
```

Step 2: Run web-application

```
$ docker run -d --tty --init --privileged \
             -p 80:4200 \
             --network build_frontend \
             --name web-application \
             --volume dist:/usr/share/nginx/html \
             build_web-application
```

Step 3: Run service registration and discovery

```
$ docker run -d --tty --init --privileged \
             -p 8500:8500 \
             -p 8600:8600 \
             --expose 8500 \
             --expose 8600 \
             --network build_backend \
             --hostname consul \
             --name consul \
             --volume data:/consul/data \
             consul:1.7.3 agent -server -client 0.0.0.0 -ui -bootstrap-expect=1
-data-dir=consul/data -datacenter=blr
```

Step 4: Run messaging queue

```
$ docker run -d --tty --init --privileged \
             -p 5672:5672 \
             -p 15672:15672 \
             --expose 15672 \
             --network build_backend \
             --hostname rabbitmq \
             --name rabbit-mq \
             --env CLUSTERED=true \
             --env RAM_NODE=true \
             --env CLUSTER_WITH=rabbit \
             --env RABBITMQ_DEFAULT_USER=docker \
             --env RABBITMQ_DEFAULT_PASS=docker \
             rabbitmq:3.8-management-alpine
```

Step 5: Run service API gateway

```
$ docker run -d --tty --init --privileged \
             -p 8080:8080 \
             --expose 8080 \
             --network build_backend \
             --hostname api-gateway \
             --name api-gateway \
             --env SPRING_PROFILES_ACTIVE=docker \
             build_api-gateway
```

Step 6: Run services

Run service-two

```
$ docker run -d --tty --init --privileged \
             -p 8084:8084 \
             --network build_backend \
             --hostname service-two \
             --name service-two \
             --env SPRING_PROFILES_ACTIVE=docker \
             build_service-two
```

Test service-two by

```
$ curl -X GET http://localhost:8084/ | json_pp
```

Run service-one

```
$ docker run -d --tty --init --privileged \
             -p 8082:8082 \
             --network build_backend \
             --hostname service-one \
             --name service-one \
             --env SPRING_PROFILES_ACTIVE=docker \
             build_service-one
```

Test service-one by

```
$ curl -X GET http://localhost:8082/ | json_pp
```

Step final: Check you result

Access to: http://localhost

Part 5: Optimize Docker images

Optimize 1: Don't install unnecessary dependencies

Using a --no-install-recommends when apt-get installing packages. This will result in a smaller image size.

```
RUN apt-get update && apt-get install -y --no-install-recommends
```

Optimize 2: Remove apt library cache

Use of apt-get update should be paired with rm -rf /var/lib/apt/lists/* in the same layer.

```
RUN apt-get update && rm -rf /var/lib/apt/lists/*
```

Optimize 3: Use .dockerignore file

Use docker ignore file on the root of context directory to ignore stuff we don't want to keep when building docker image.

```
.dockerignore
.gitignore
*.DS_Store
target/
*.iml
*.md
src/main/docker/
```

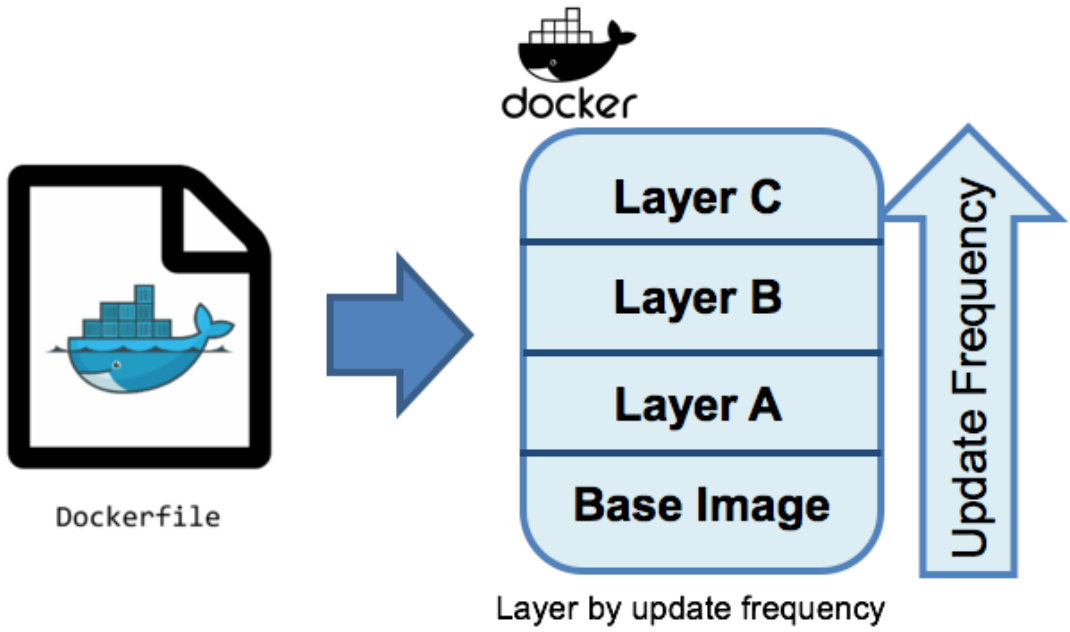
Optimize 4: Choose correct base image

- Don't use latest tag because it's change frequently.
- Prefer minimal version (use image with *-alpine* suffix).
- Use official image. Example:

```
FROM node:12-alpine ## instead of "FROM node:12"
FROM postgres:9.6-alpine ## instead of "FROM postgres:9.6"
..
```

Optimize 5: Order matters for caching

Analyse the source code, move some thing doesn't change frequently to top layers.




```
# Copy only pom file for fetching dependencies
COPY pom.xml /usr/src/app
# Fetch dependencies first
RUN mvn -e -B dependency:resolve

COPY . /usr/src/app

RUN mvn clean package
..
```

instead of

```
COPY . /usr/src/app

RUN mvn clean package
..
```

Optimize 6: Use multi-stage if needed

If you are using one container for multi purpose, please consider to break it smaller into multi-stage.

BAD

```
## Following section used for packaging
FROM maven:3.6.1-jdk-8-alpine
LABEL author="Phong Pham"
LABEL maintainer="pnqphong95@gmail.com"

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app


COPY pom.xml /usr/src/app

RUN mvn -e -B dependency:resolve

COPY . /usr/src/app

RUN mvn clean package

## Following section used for running application

EXPOSE 8082
ENTRYPOINT ["java","-jar","target/my-service.jar"]
```

GOOD

```
## Following section used for packaging, create stage packaging
FROM maven:3.6.1-jdk-8-alpine AS packaging
LABEL author="Phong Pham"
LABEL maintainer="pnqphong95@gmail.com"

RUN mkdir -p /usr/src/app
WORKDIR /usr/src/app


COPY pom.xml /usr/src/app

RUN mvn -e -B dependency:resolve

COPY . /usr/src/app

RUN mvn clean package

## Following section used for running application
FROM openjdk:8-jre-alpine

COPY --from=packaging /usr/src/app/target/my-service.jar /

EXPOSE 8082
ENTRYPOINT ["java","-jar","/my-service.jar"]
```

Part Final: Migrate to Docker Compose (Optional)

Continue migrate to Docker Compose

https://www.composerize.com/

https://8gwifi.org/dc1.jsp