Introduction to Docker & 12 Factor App Implementation Using Docker

Cemal Ünal

- Software Engineer @ Havelsan Inc.

 cemalunal

 cemalunal

 cunal@havelsan.com.tr

Glossary

- **Image:** A package that contains the application along with the dependencies that required to run this application.
- **Container:** Running instance of the image
- **Tag:** Convey useful information about a specific image version/variant
- **Registry:** Storage and distribution system for named images

Dockerfile Example

Dockerfile

```
1   FROM ubuntu:18.04
2
3   RUN apt update && apt-get -y install nginx
4
5   COPY index.html /var/www/html
6
7   CMD ["nginx", "-g", "daemon off;"]
8
```

Docker Build Command

- Allows building an image using a Dockerfile

```
➜  example-docker-commands git:(master) docker build -t ubuntu-based-nginx:v1 .
Sending build context to Docker daemon  13.82kB
Step 1/4 : FROM ubuntu:18.04
 ---> 72300a873c2c
Step 2/4 : RUN apt update && apt-get -y install nginx
 ---> Using cache
 ---> 2838e5a9cbb5
Step 3/4 : COPY index.html /var/www/html
 ---> Using cache
 ---> ba886a6ad0f0
Step 4/4 : CMD ["nginx", "-g", "daemon off;"]
 ---> Using cache
 ---> b074a47c249c
Successfully built b074a47c249c
Successfully tagged ubuntu-based-nginx:v1
➜  example-docker-commands git:(master)
```

Dockerfile Example of a Java Program

Dockerfile

backend > Dockerfile

```
1   FROM maven:3.6.1-jdk-11-slim as maven
2
3   WORKDIR /app
4   COPY ./pom.xml ./pom.xml
5
6   # build all dependencies
7   RUN mvn dependency:go-offline -B
8
9   COPY ./src ./src
10
11  RUN mvn clean package
12
13  # specify base image runtime
14  FROM openjdk:11.0-jre-slim
15
16  WORKDIR /app
17
18  # copy over the built artifact from the maven image
19  COPY --from=maven /app/target/*.jar /app/target/
20
21  # set the startup command to run binary
22  CMD java ${JAVA_OPTS} -jar  /app/target/*.jar
```

Docker Run

- Allows us to create a running instance (container) of an image

```
➜  example-docker-commands git:(master) docker run ubuntu:18.04 echo 'Hello world!'
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
423ae2b273f4: Pull complete
de83a2304fa1: Pull complete
f9a83bce3af0: Pull complete
b6b53be908de: Pull complete
Digest: sha256:04d48df82c938587820d7b6006f5071dbbffceb7ca01d2814f81857c631d44df
Status: Downloaded newer image for ubuntu:18.04
Hello world!
➜  example-docker-commands git:(master)
```

Docker Volumes

- By default all files created inside a container do not persist when that container no longer exists

- Two options available:
  - Volumes
    - Managed by Docker
    - docker volume create test-volume
    - docker run -v test-volume:/data .....
  - Bind mounts
    - A file or directory on the host machine is mounted into a container
    - The file or directory is referenced by its full path on the host machine
    - docker run -v /full/path/here:/data .....

Docker Volumes Options Use Cases

- Volume
  - Allows storage for container’s data on a remote host or a cloud provider
  - Backup operations are simple
  - When the Docker host is not guaranteed to have a given directory or file structure

- Bind Mount
  - Sharing configuration files from the host machine to containers.
  - When the Docker host is guaranteed to be consistent with the bind mounts the containers require.

Docker Network

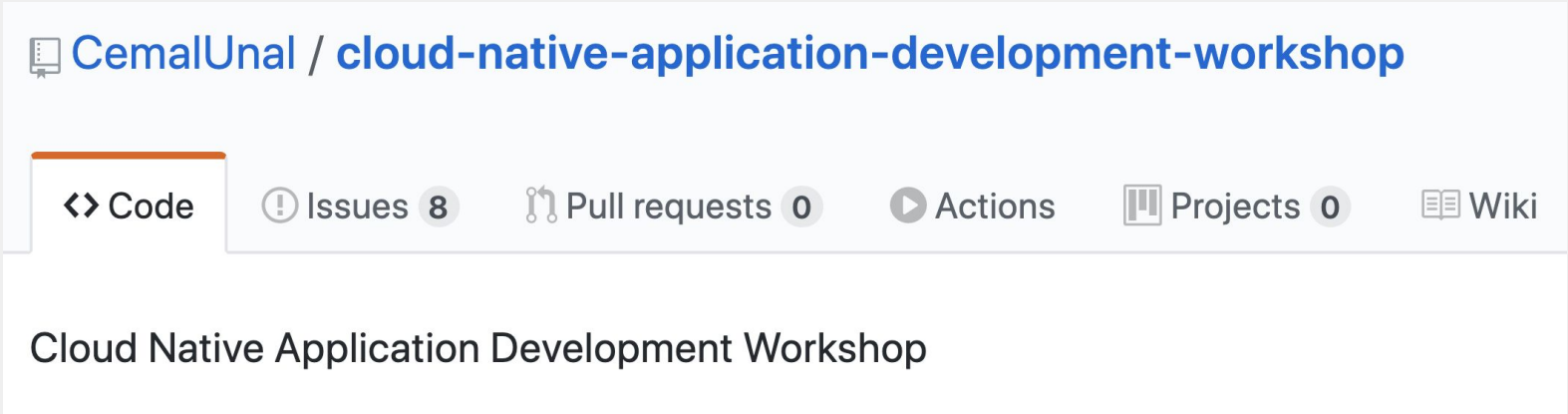
- Provide complete isolation for containers

- Most common docker network drivers:
  - Bridge
    - Default driver
    - Usually used when your applications run in standalone containers that need to communicate

  - Host
    - Remove network isolation between the container and the host
    - Use the host's networking directly

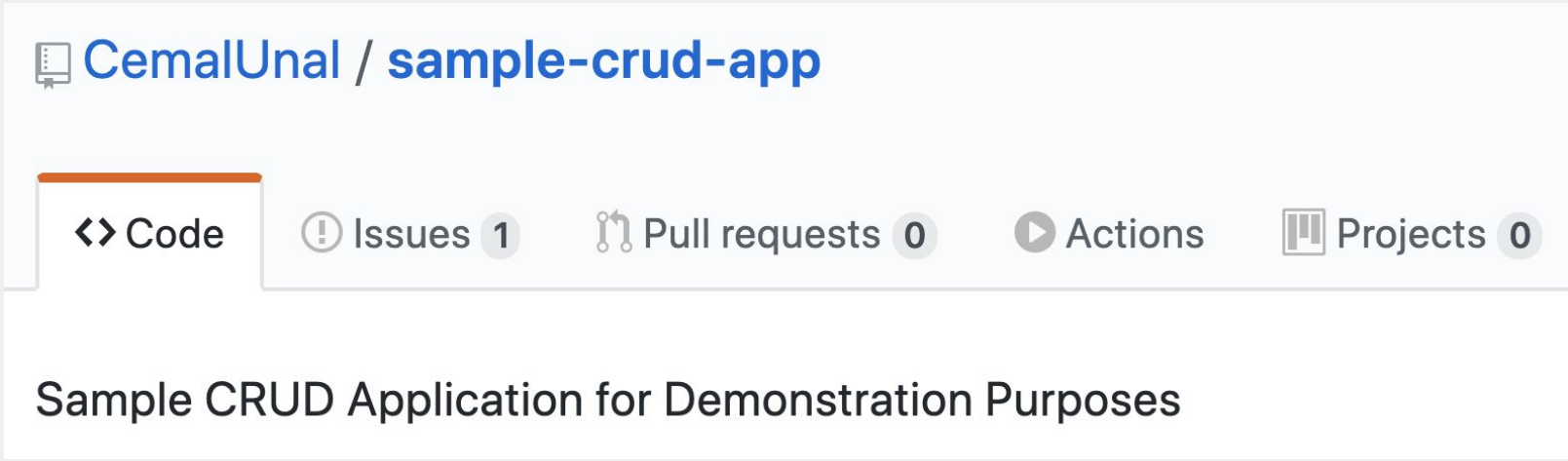
  - Overlay
    - Connect multiple Docker daemons together and enable swarm services to communicate with each other

Docker Demo



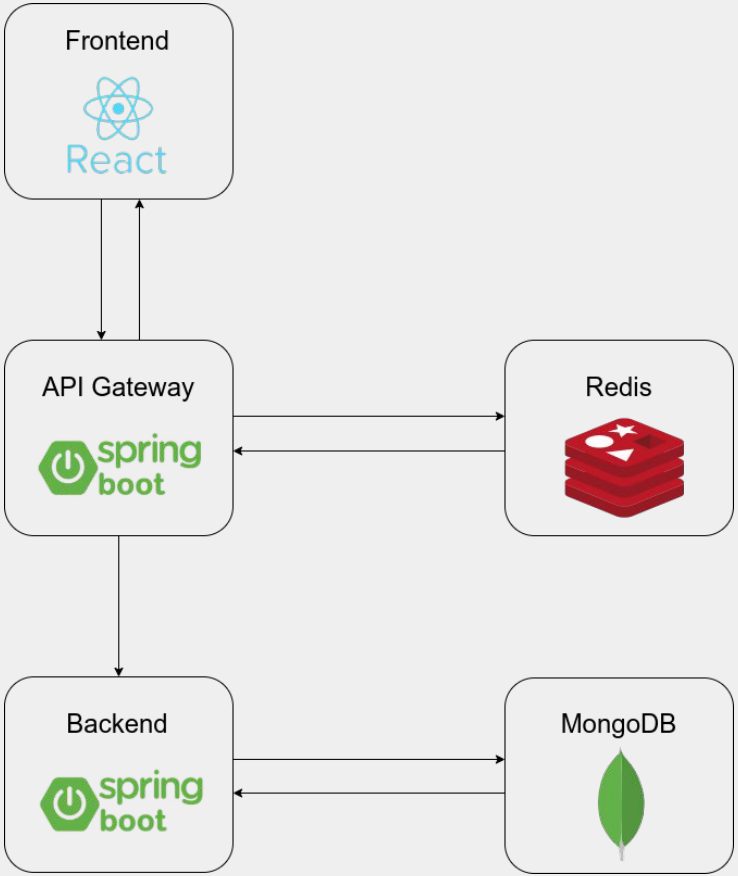
$ git clone
https://github.com/cemalunal/cloud-native-application-development-workshop.git

12-factor App Implementation



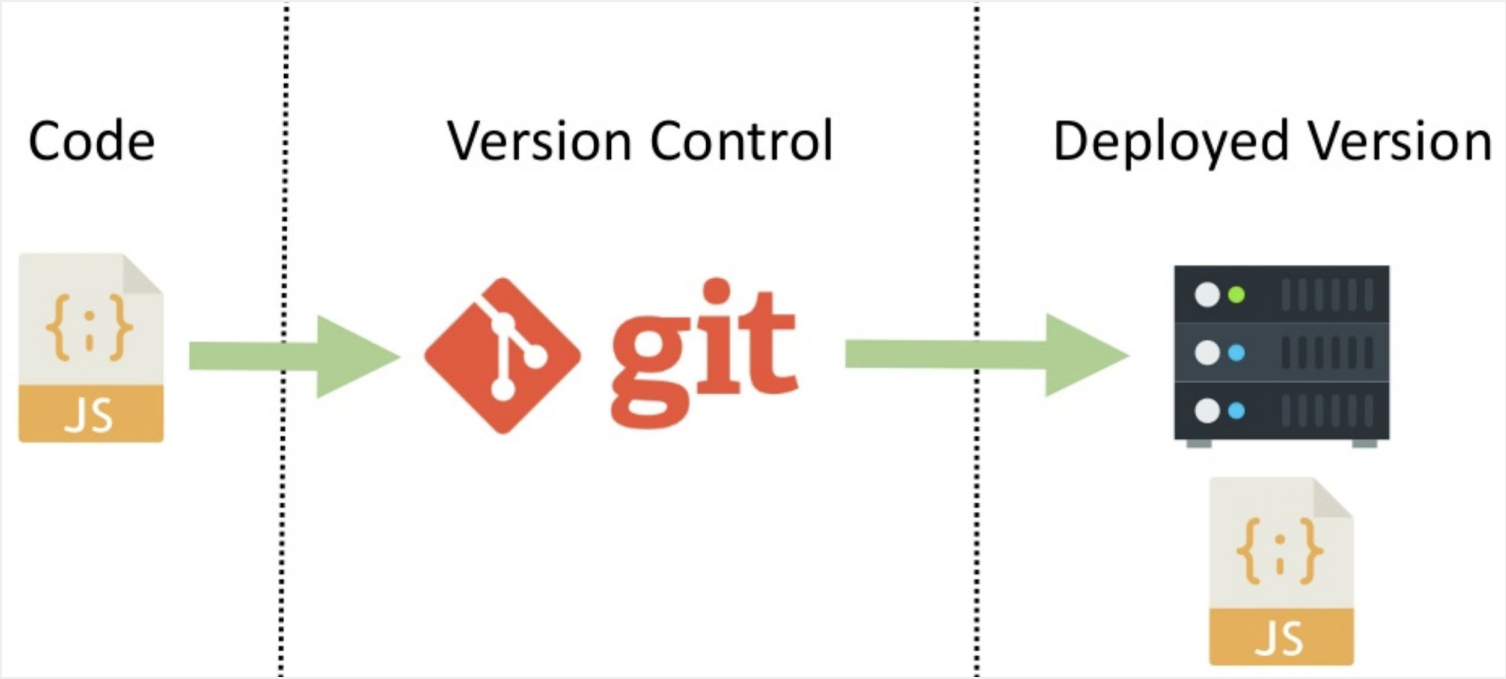
$ git clone https://github.com/cemalunal/sample-crud-app.git

Architecture Diagram



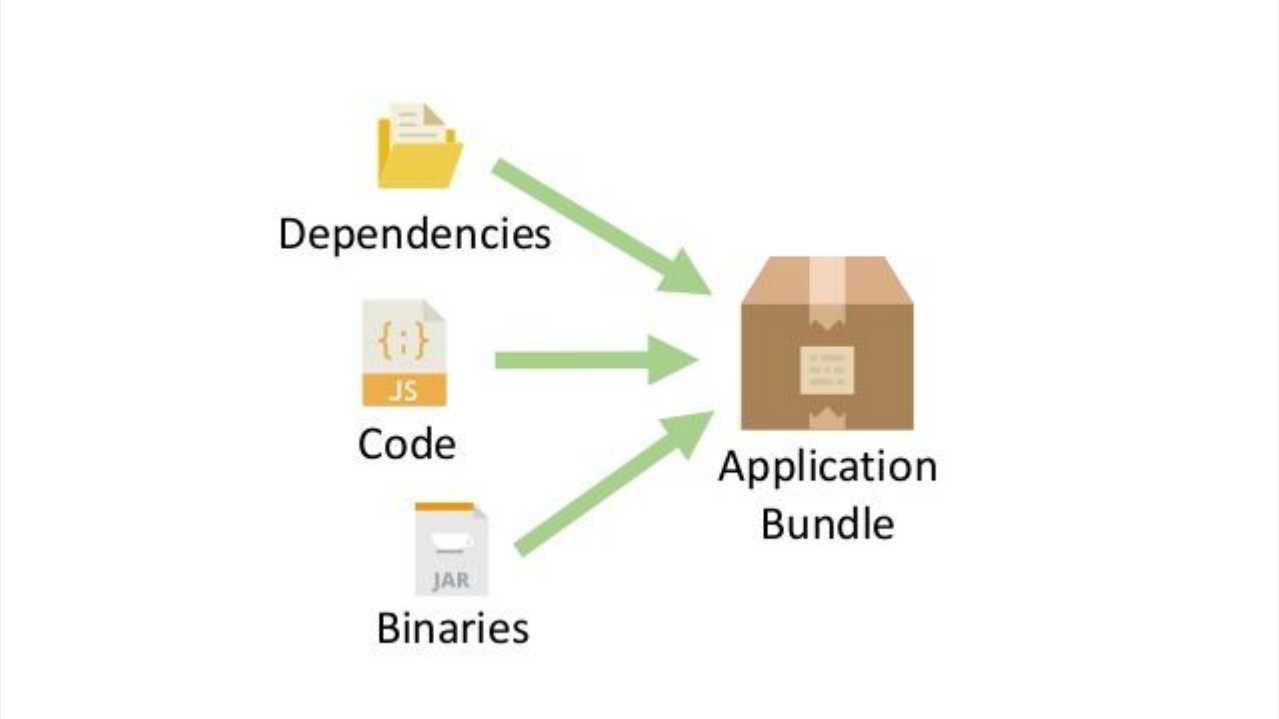
1- Codebase

One codebase tracked in revision control, many deploys



2- Dependencies

Explicitly declare and isolate dependencies



Dependency Declaration - Node.js

package.json

```
{
  "name": "simple-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@material-ui/core": "^3.0.0",
    "isomorphic-fetch": "^2.2.1",
    "react": "^16.4.2",
    "react-dom": "^16.4.2",
    "react-router-dom": "^4.3.1",
    "react-scripts": "1.1.5",
    "serve": "^10.1.2"
  },
```

$ npm install

Dependency Declaration - Java w/ Maven

pom.xml

```xml
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-jetty</artifactId>
  <version>2.2.4.RELEASE</version>
</dependency>

<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-data-mongodb</artifactId>
  <version>2.2.4.RELEASE</version>
</dependency>

<dependency>
  <groupId>io.springfox</groupId>
  <artifactId>springfox-swagger2</artifactId>
  <version>2.7.0</version>
</dependency>
```

$ mvn install

3- Config

Store config in the environment

- Frontend
  - URL of the backend service is stored in environment variables and accessed via **window.env**
    - fetch(`${window.env.REACT_APP_BACKEND_URI}/customers`)

- Backend
  - MongoDB connection URI is stored is stored in environment variables and accessed via application-deployment.properties file
    - spring.data.mongodb.uri=${MONGODB_URI}

Backend container gets config from the environment

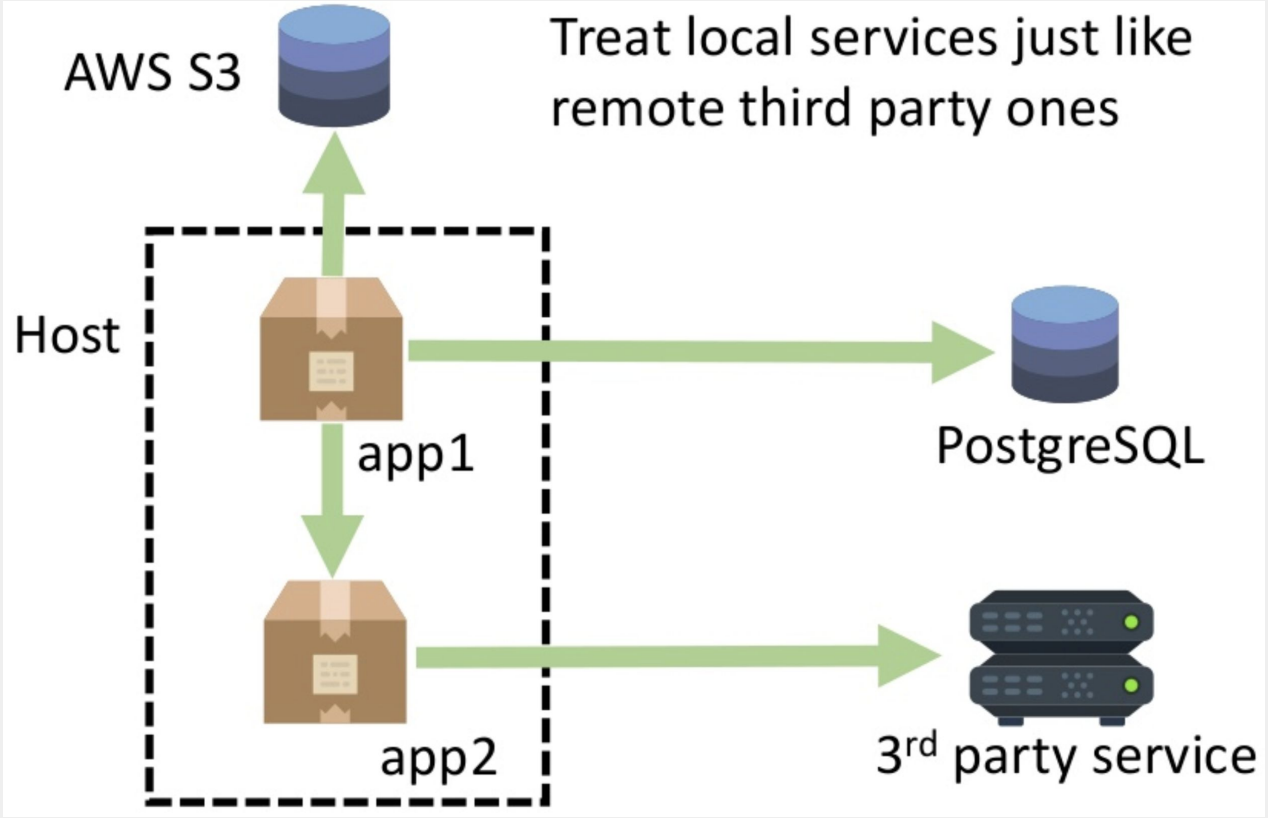
application-deployment.properties

```
server.port=${SERVER_PORT}
spring.data.mongodb.uri=${MONGODB_URI}
```

```
docker run -d --network=crud-app \
           --name backend \
           -e MONGODB_URI="mongodb://mongodb:27017/sample-app" \
           -e JAVA_OPTS="-Dspring.profiles.active=deployment -Dserver.port=80 -Xms125m -Xmx250m" \
           --restart=on-failure \
           cunal/demo-backend:v0.0.1
```

4- Backing Services

Treat backing services as attached resources



MongoDB connection for Backend

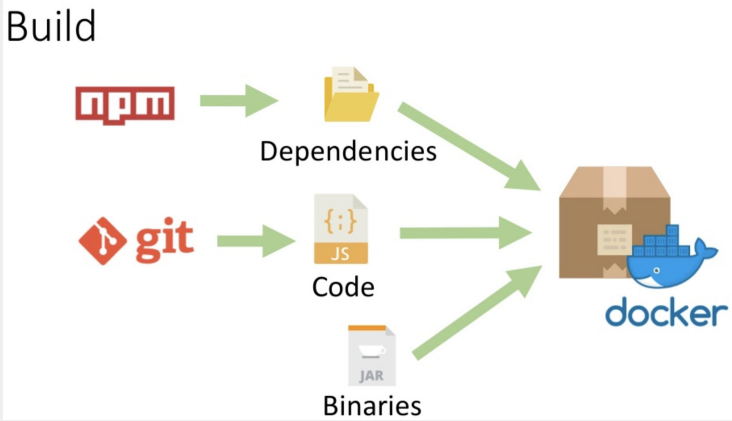
- Think about MongoDB - Connection URI is stored in **MONGODB_URI** environment variable.

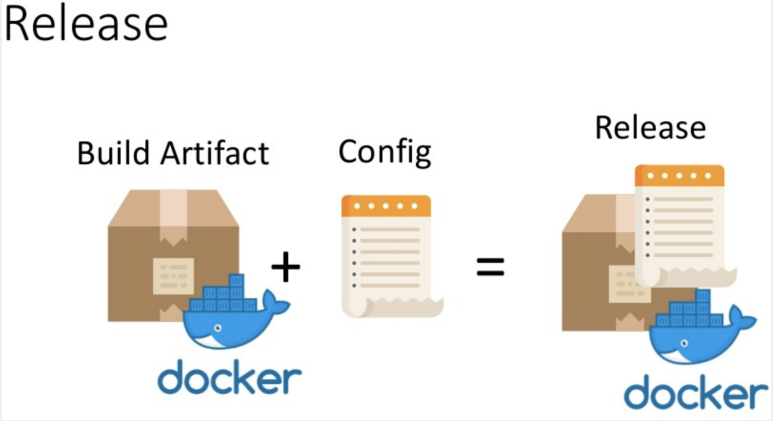
- We can easily switch between local and production MongoDB databases. Or we can even use Azure Cosmos DB by just changing the connection string. Examples:

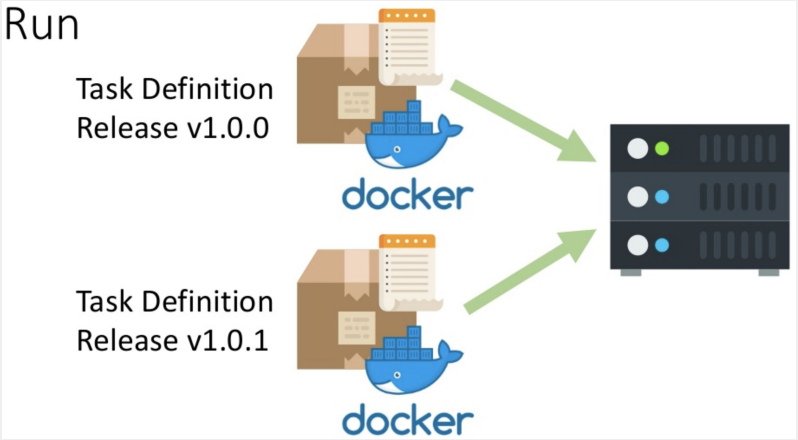
    - mongodb://localhost:27017/sample-app
    - mongodb://mongodb:27017/sample-app
    - mongodb://user:pass@test.documents.azure.com:10255/dbname?ssl=true

5- Build, release, run

Strictly separate build and run stages

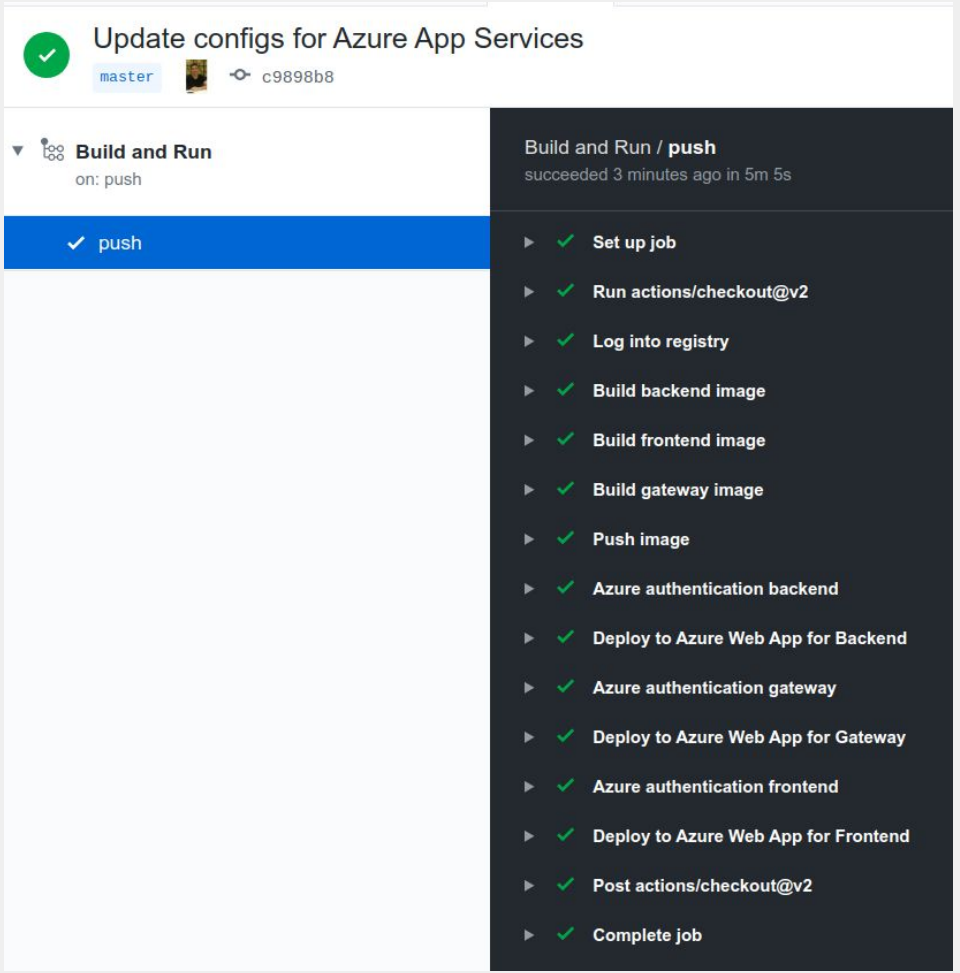






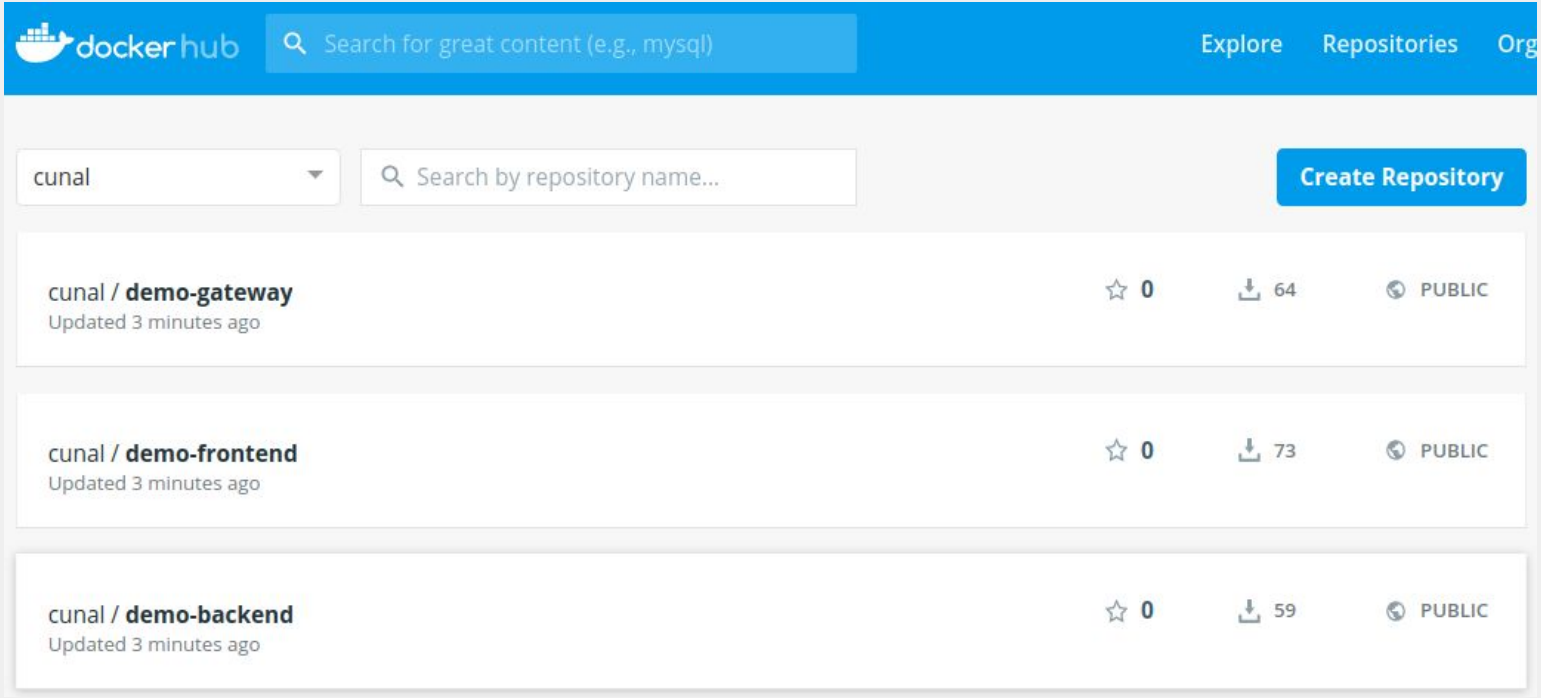
Build - Release

- There are 3 Dockerfiles for each service (frontend, backend and gateway)
- GitHub actions is triggered at each commit on master branch



Build - Release

- Each build artifact is stored in Docker Hub



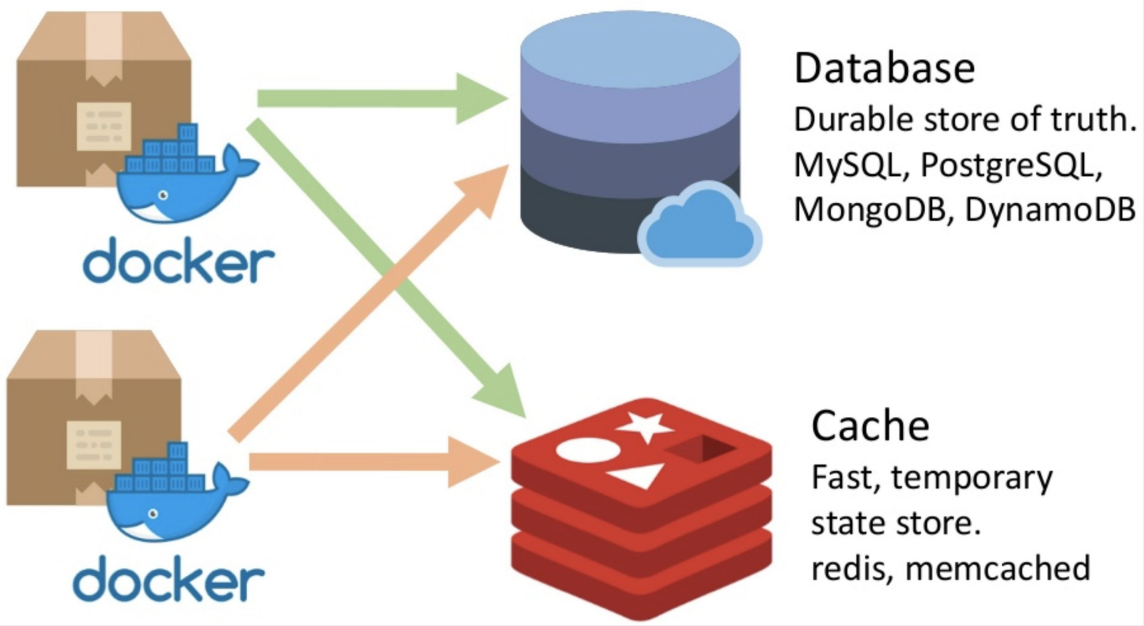
Run

- Azure App Services
  - Allows to run / scale pre built docker image without managing any servers.
- Azure Cosmos DB
  - MongoDB API for backend
- Azure Cache for Redis
  - Gateway Rate limiting

https://azure-demo-frontend.azurewebsites.net/

6- Processes

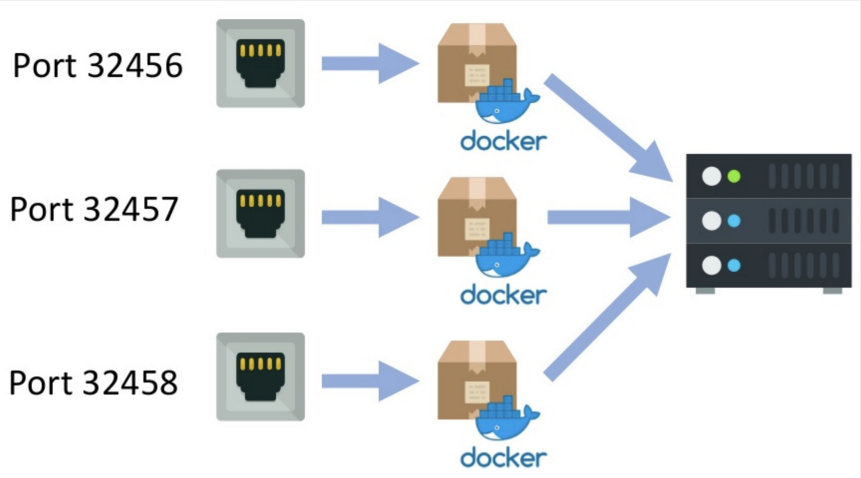
Execute the app as one or more stateless processes



- The application delegates stateful persistence to MongoDB.
- It is easily scalable since it is stateless.

7- Port Binding

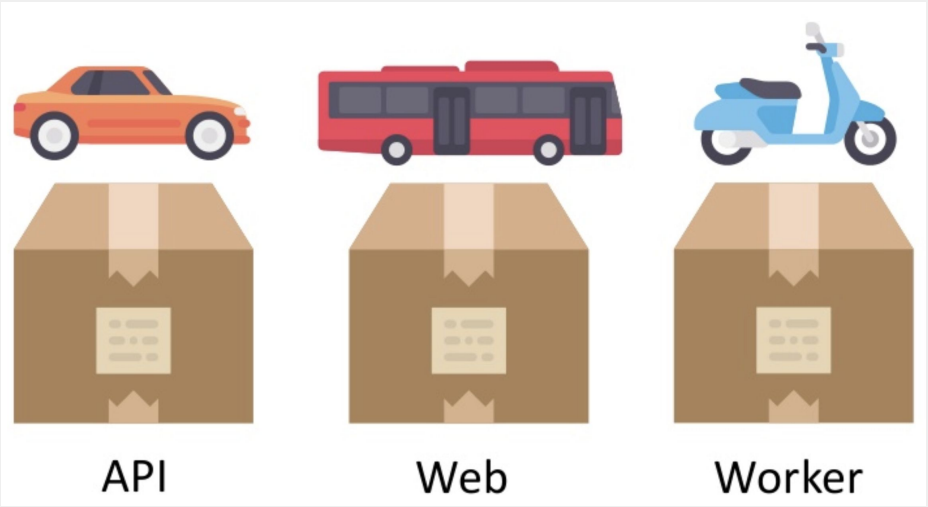
Export services via port binding



- Backend and Gateway
  - Spring Boot is used along with embedded Jetty server.
  - server.port=${SERVER_PORT} in application-deployment.properties
- Frontend
  - serve npm package is used to serve the static frontend
  - serve -l $SERVER_PORT -s build in startup.sh

8- Concurrency

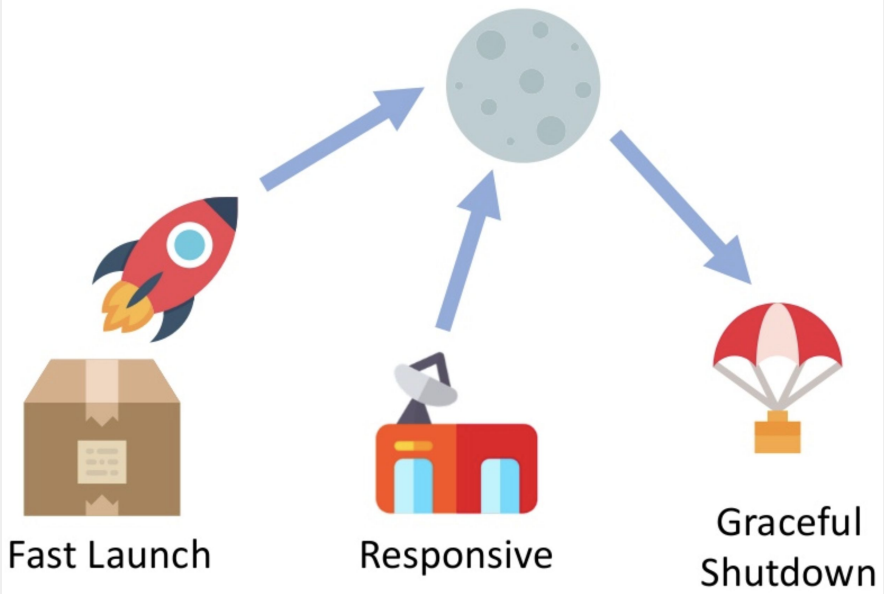
Scale out via the process model



- All components of the application is dockerized
- Launching multiple instances is simple.

9- Disposability

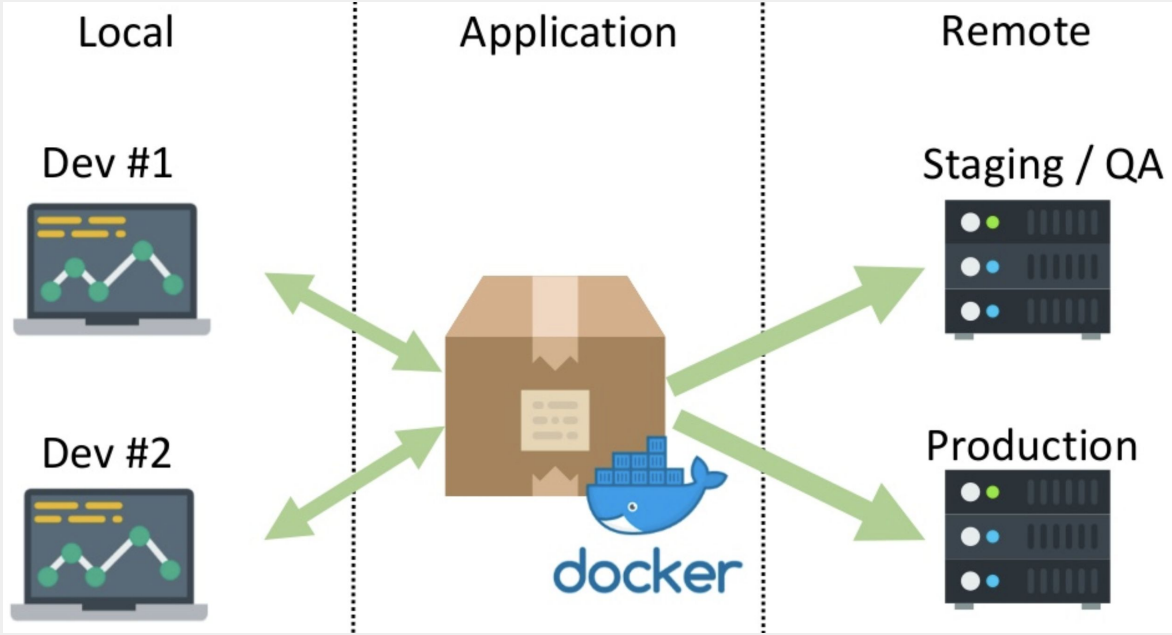
Maximize robustness with fast startup and graceful shutdown



- All components of the sample application are disposable and can be started and stopped quickly
- They shut down gracefully when they receive SIGTERM

10- Dev / prod parity

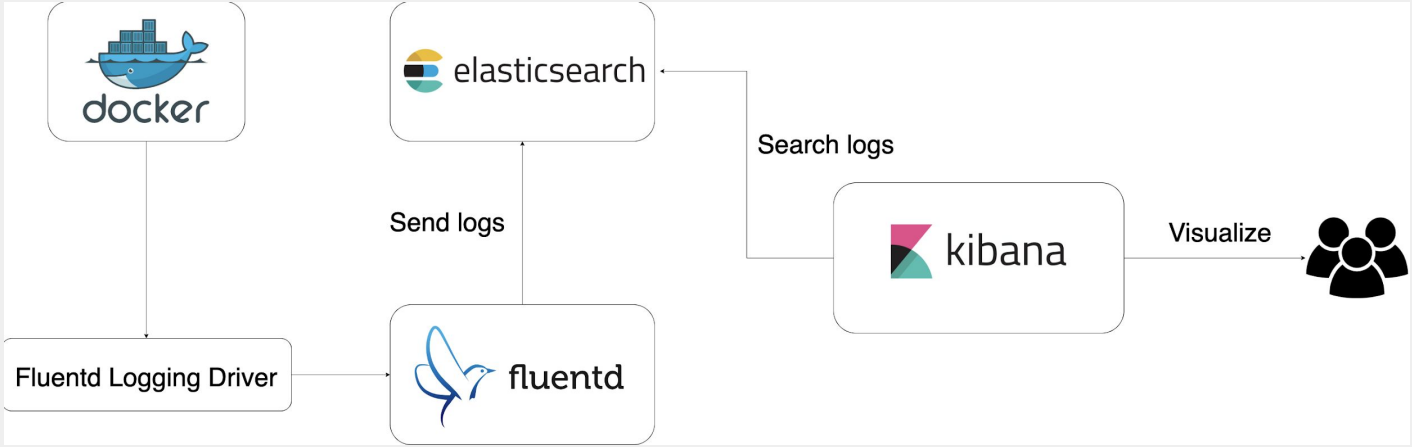
Keep development, staging, and production as similar as possible



- **Docker** is used to run app components and the third party services.
- **Docker** and **Docker Compose** allow developers to run local environments which closely approximate production environments.

11- Logs

Treat logs as event streams




```
docker run -p 27017:27017 -d --network=demo-network \
           --name mongodb \
           -v mongodb_data:/data/db \
           --restart=on-failure \
           --log-driver=fluentd --log-opt fluentd-address=localhost:24224 \
           mongo:4.0.2
```

THANKS!