Introduction to Docker & 12 Factor App Implementation Using Docker

Cemal Ünal

- Software Engineer @ Havelsan Inc.

cemalunal

cemalunal

cunal@havelsan.com.tr

Keywords

- **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

Docker Container Example

```
➜  ~ docker run --name greeting ubuntu:18.04 echo "Hello class of CS443, I'm a container"
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
5c939e3a4d10: Already exists
c63719cdbe7a: Already exists
19a861ea6baf: Already exists
651c9d2d6c4f: Already exists
Digest: sha256:8d31dad0c58f552e890d68bbfb735588b6b820a46e459672d96e585871acc110
Status: Downloaded newer image for ubuntu:18.04
Hello class of CS443, I'm a container
➜  ~
```

Docker Run Command

- --name flag - specify a name for your container
- ubuntu:18.04 - Docker image
- echo - process name to execute
- greeting message - parameter(s)

Docker Image Creation using Dockerfile)

Dockerfile

```
FROM ubuntu:18.04

RUN echo hello > hello.txt
```

```
➜  backend git:(master) ✗ docker build -t ubuntu:18.04-greeting-declarative .
Sending build context to Docker daemon    106kB
Step 1/2 : FROM ubuntu:18.04
 ---> ccc6e87d482b
Step 2/2 : RUN echo hello > hello.txt
 ---> Running in 603f82cc6874
Removing intermediate container 603f82cc6874
 ---> 21ff26d14177
Successfully built 21ff26d14177
Successfully tagged ubuntu:18.04-greeting-declarative
➜  backend git:(master) ✗ docker run -it  --name greeting-declarative ubuntu:18
.04-greeting-declarative bash
root@87cae5072156:/# echo hello.txt
hello.txt
root@87cae5072156:/# cat hello.txt
hello
root@87cae5072156:/#
```

Dockerfile Example of a Java Program

backend > Dockerfile

```
FROM maven:3.6.1-jdk-11-slim as maven

WORKDIR /app
COPY ./pom.xml ./pom.xml

# build all dependencies
RUN mvn dependency:go-offline -B

COPY ./src ./src

RUN mvn clean package

# specify base image runtime
FROM openjdk:11.0-jre-slim

WORKDIR /app
VOLUME /tmp

# copy over the built artifact from the maven image
COPY --from=maven /app/target/*.jar /app/target/

# set the startup command to run binary
CMD java ${JAVA_OPTS} -jar  /app/target/*.jar
```

Docker Volumes

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

Docker Volumes

```
root@dell-precision:~# docker volume create --name mongodb_data
mongodb_data
root@dell-precision:~# docker volume ls | grep mongo
local               mongodb_data
root@dell-precision:~# docker run -p 27017:27017 -d \
>             --name mongodb \
>             -v mongodb_data:/data/db \
>             mongo:4.0.2
```

```
root@dell-precision:~# docker run -p 27017:27017 -d \
>             --name mongodb \
>             -v /home/local-pc/data:/data/db \
>             mongo:4.0.2
```

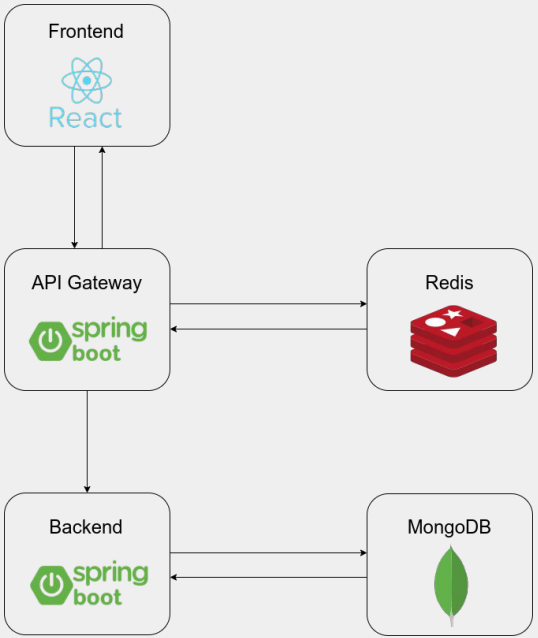
Docker Network

- 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 Network Bridge Example

```
root@dell-precision:/home/cemal# docker network create test
d8bc7c29d8660a890ecdce17ccef67b14a325fb3b32e19e45345e4b5c16f3520
root@dell-precision:/home/cemal# docker network ls | grep test
d8bc7c29d866        test                bridge              local
root@dell-precision:/home/cemal# docker run -d --network=test --name=redis redis
cab5f006089caa119916cf063c576f15130c204882550a56f7dc6df6da0158ea
root@dell-precision:/home/cemal# docker run --network=test --name=alpine alpine ping redis
PING redis (172.21.0.2): 56 data bytes
64 bytes from 172.21.0.2: seq=0 ttl=64 time=0.084 ms
64 bytes from 172.21.0.2: seq=1 ttl=64 time=0.147 ms
64 bytes from 172.21.0.2: seq=2 ttl=64 time=0.129 ms
64 bytes from 172.21.0.2: seq=3 ttl=64 time=0.132 ms
64 bytes from 172.21.0.2: seq=4 ttl=64 time=0.128 ms
^C
--- redis ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max = 0.084/0.124/0.147 ms
root@dell-precision:/home/cemal#
```

Sample CRUD App and 12-factor App

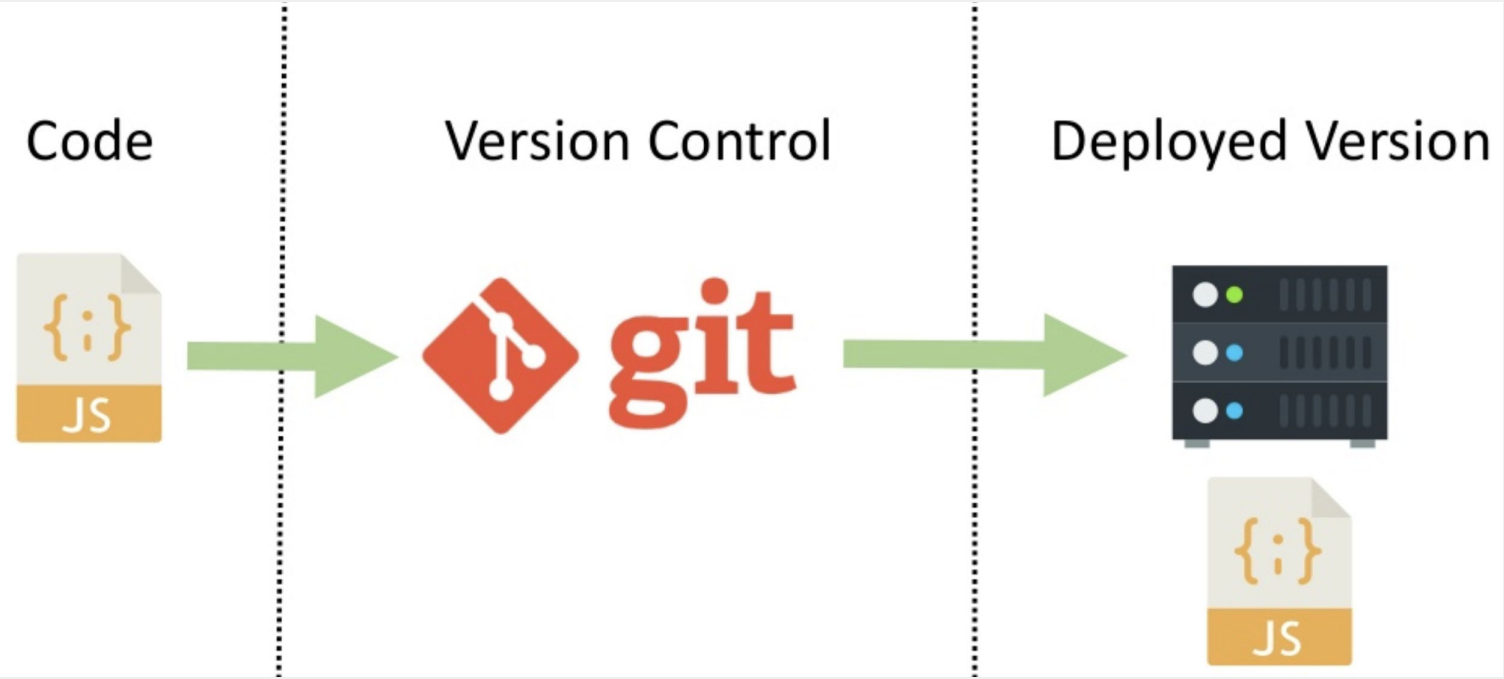


$ git clone https://github.com/cemalunal/sample-crud-app.git

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

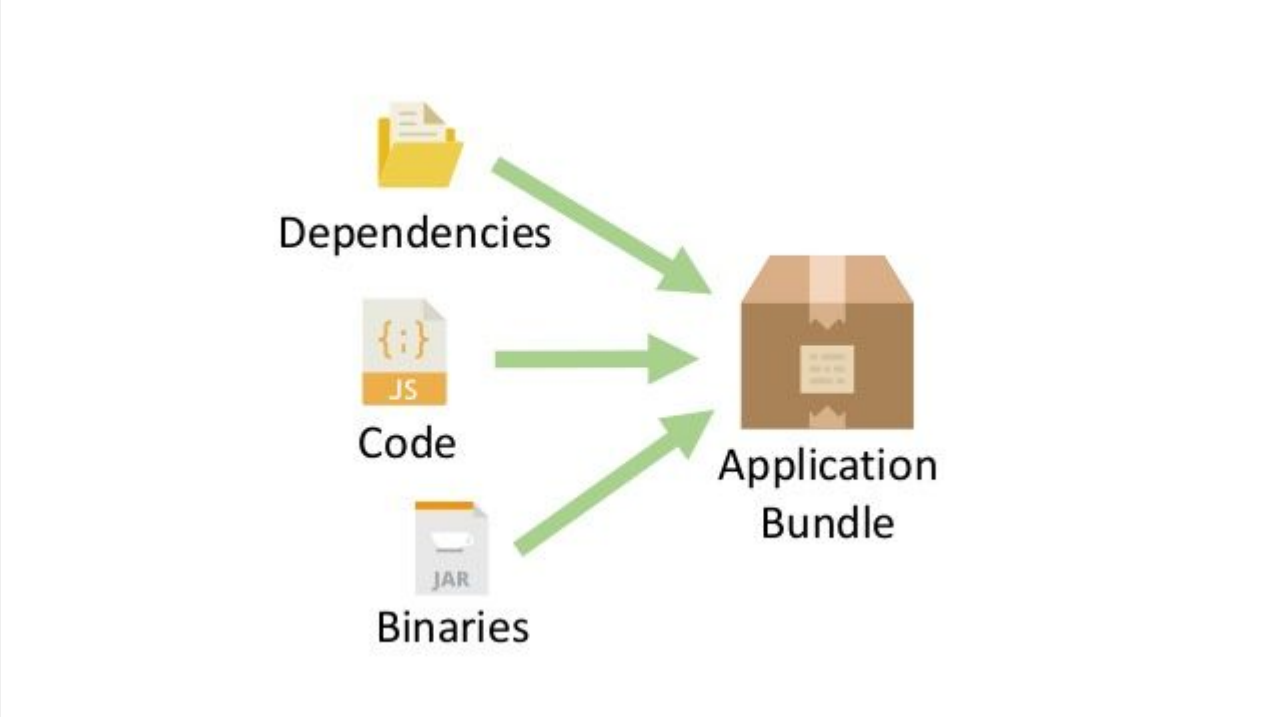
1- Codebase

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

```
<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

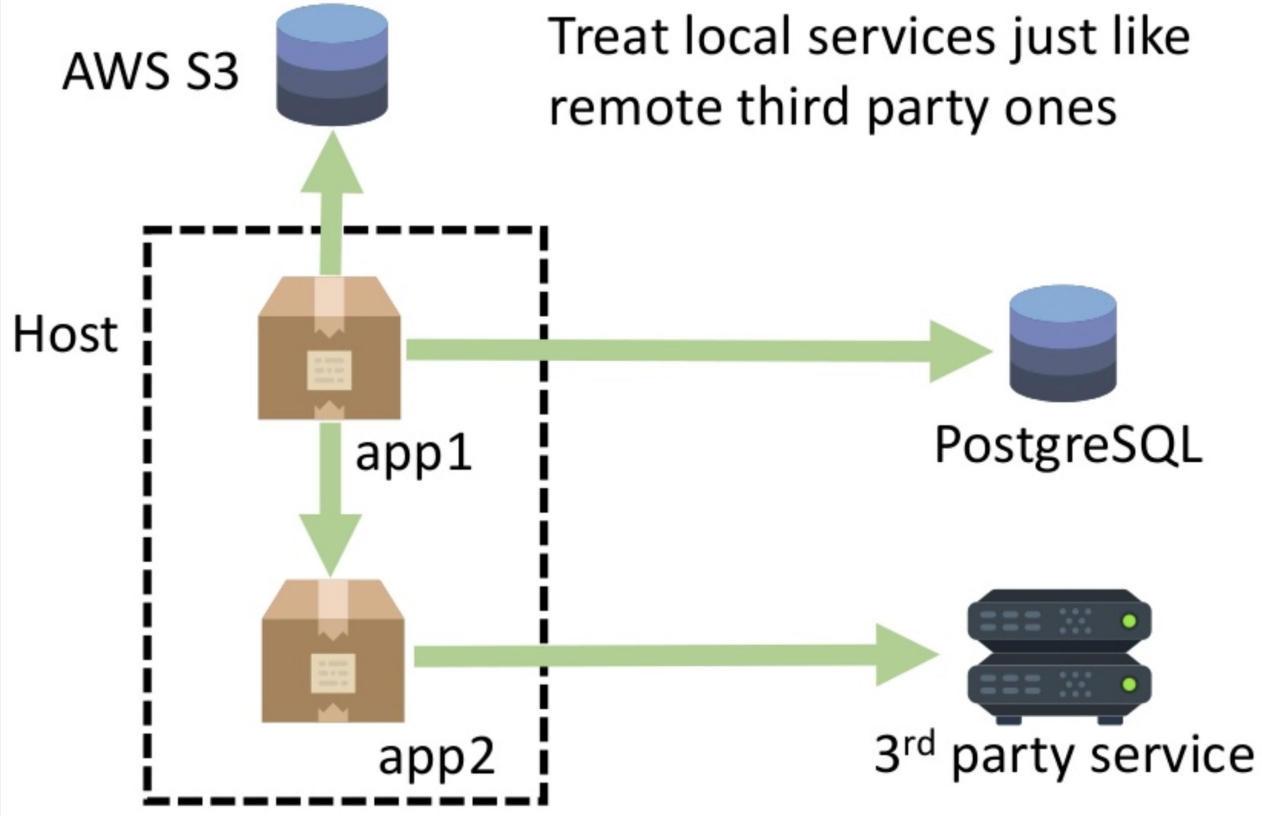
application-deployment.properties

```
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

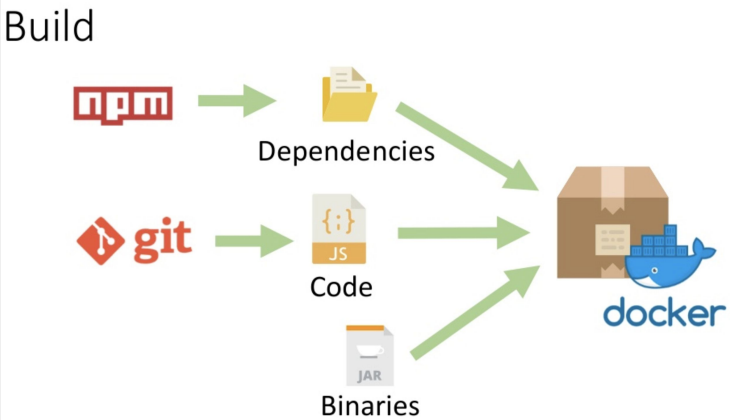
- Think about MongoDB - Connection URI is stored in **MONGODB_URI** environment variable.

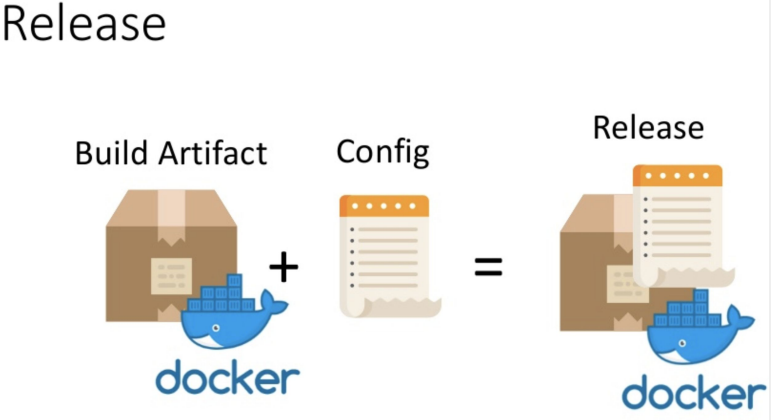
- We can easily switch between local and production MongoDB databases. Or we can even use Azure Cosmos DB by just changing the connection string. Examples:

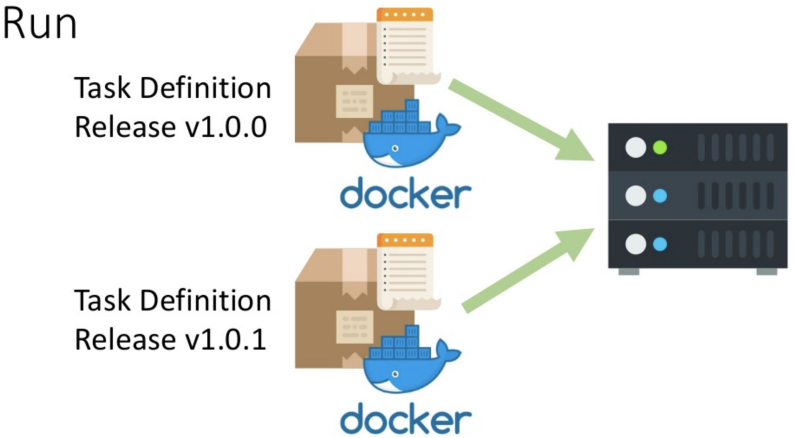
  - mongodb://localhost:27017/sample-app
  - mongodb://mongodb:27017/sample-app
  - mongodb://user:pass@test.documents.azure.com:10255/dbname?ssl=true

5- Build, release, run

Strictly separate build and run stages

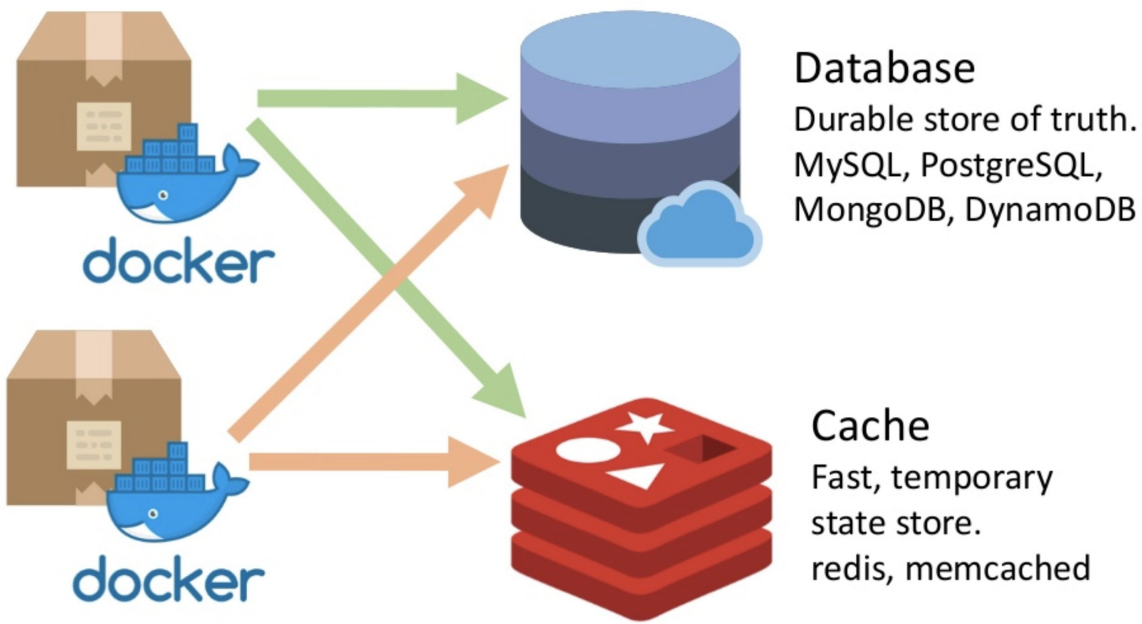






6- Processes

Execute the app as one or more stateless processes



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

Build - Release

Minor tweaks

master faa7f33

Docker
on: push

push

Docker / **push**
succeeded 1 hour ago in 3m 5s

- Set up job
- Run actions/checkout@v2
- Build backend image
- Build frontend image
- Build gateway image
- Log into registry
- Push image
- Post actions/checkout@v2
- Complete job

Search packages...

backend latest
CemalUnal/sample-crud-app
docker

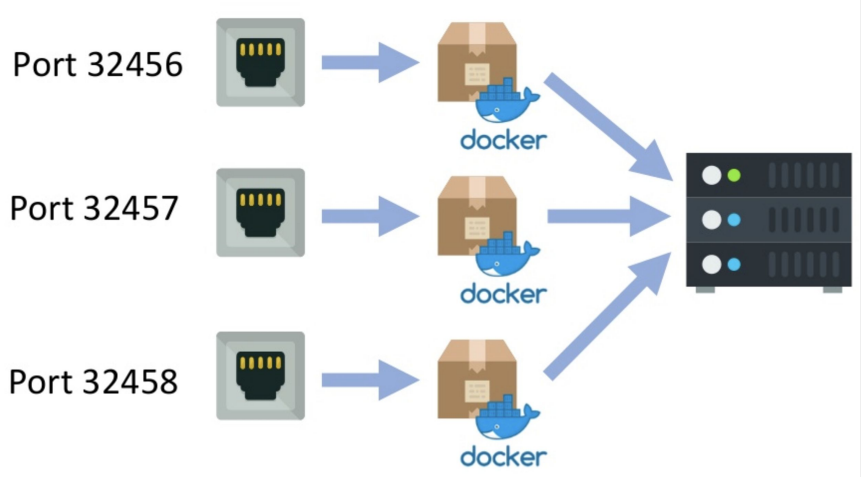
frontend latest
CemalUnal/sample-crud-app
docker

gateway latest
CemalUnal/sample-crud-app
docker

Run:

7- Port Binding

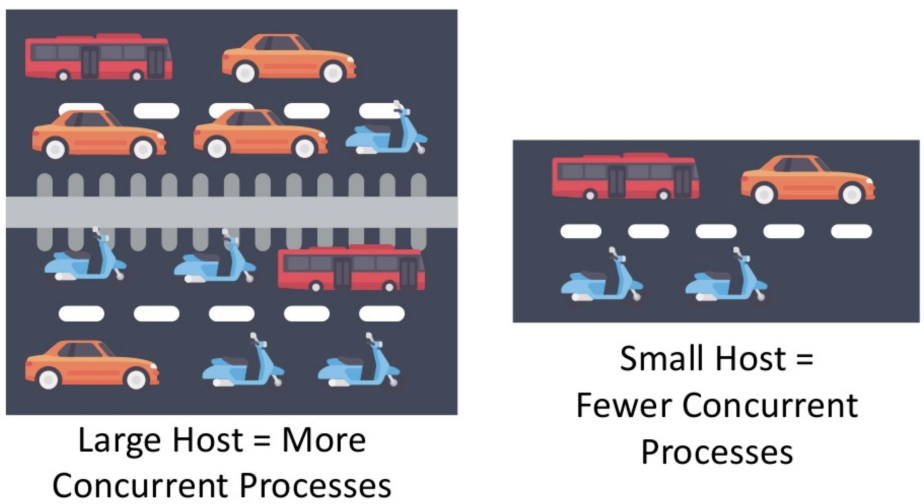
Export services via port binding



- 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

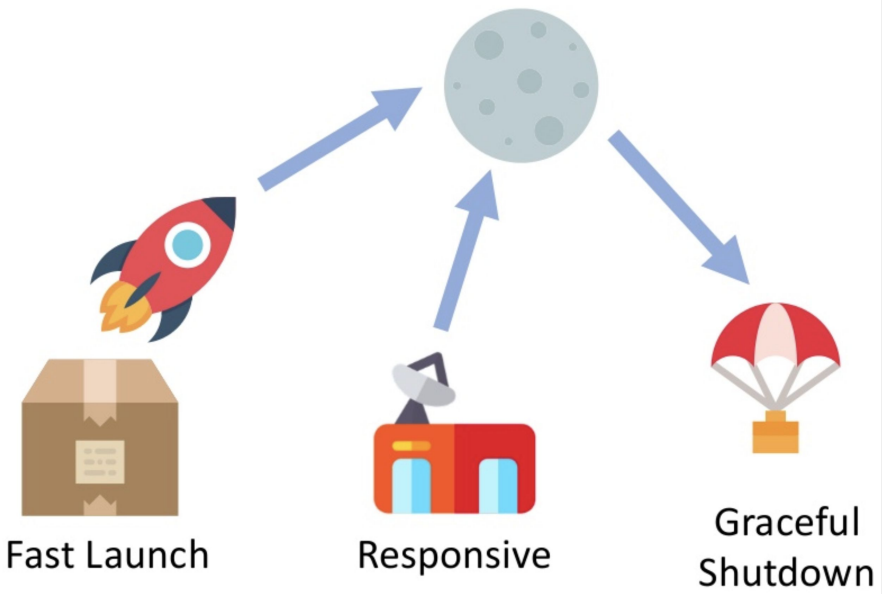
Scale out via the process model



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

9- Disposability

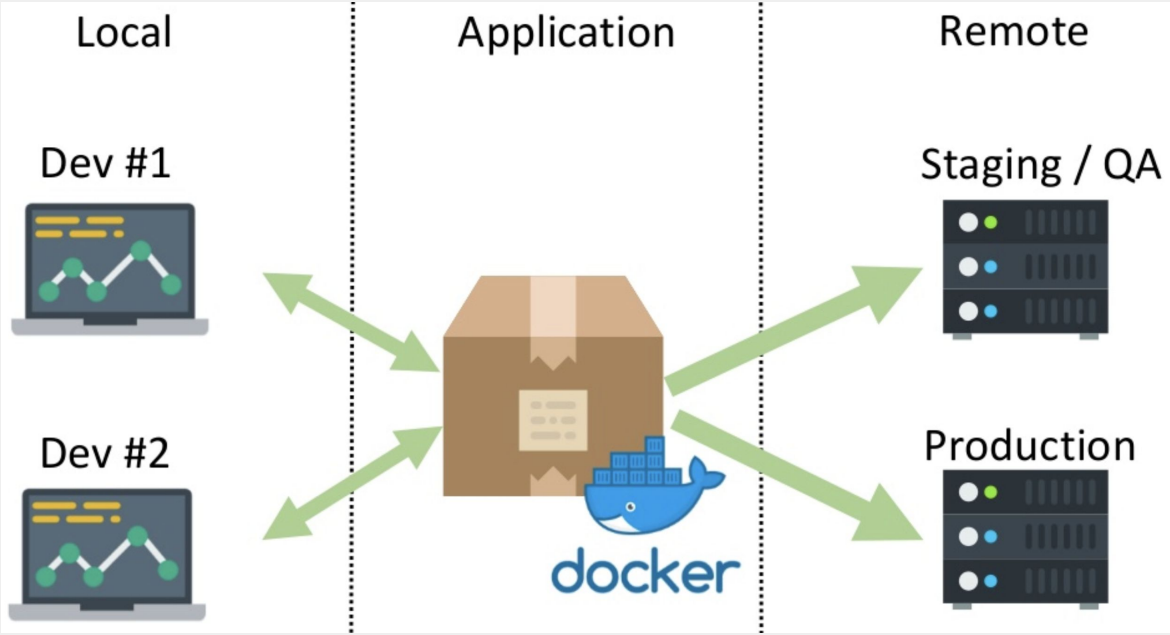
Maximize robustness with fast startup and graceful shutdown



- 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

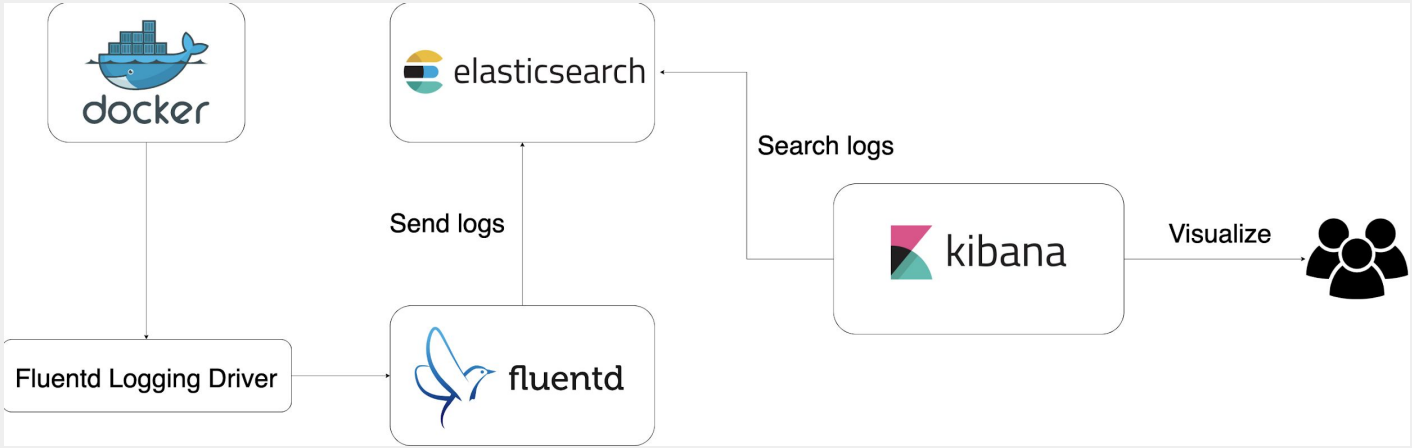
Keep development, staging, and production as similar as possible



- **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!