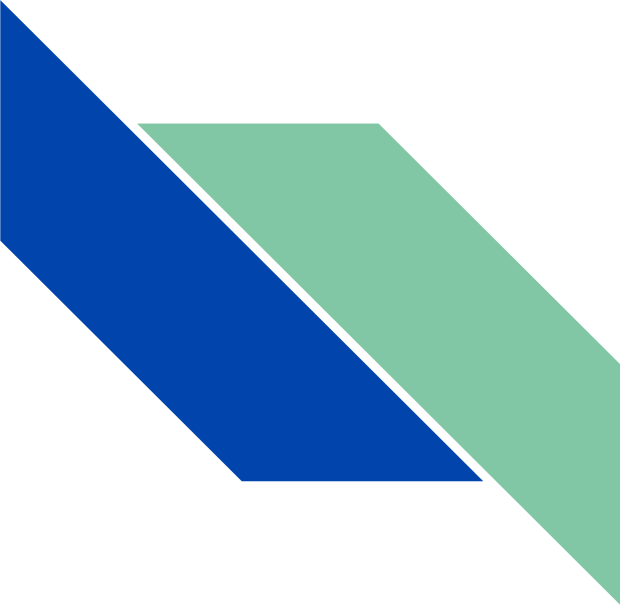


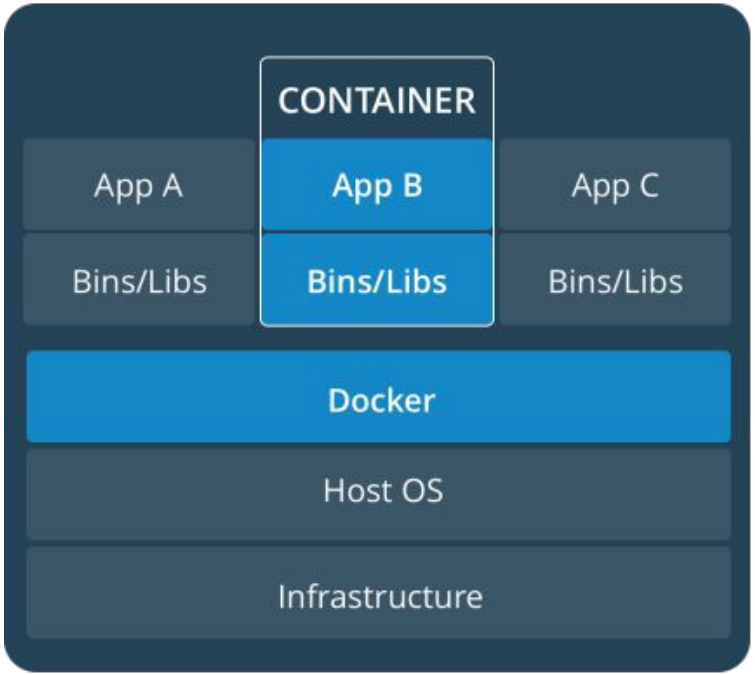
An Introduction to Docker & 12 Factor App Implementation Using Docker

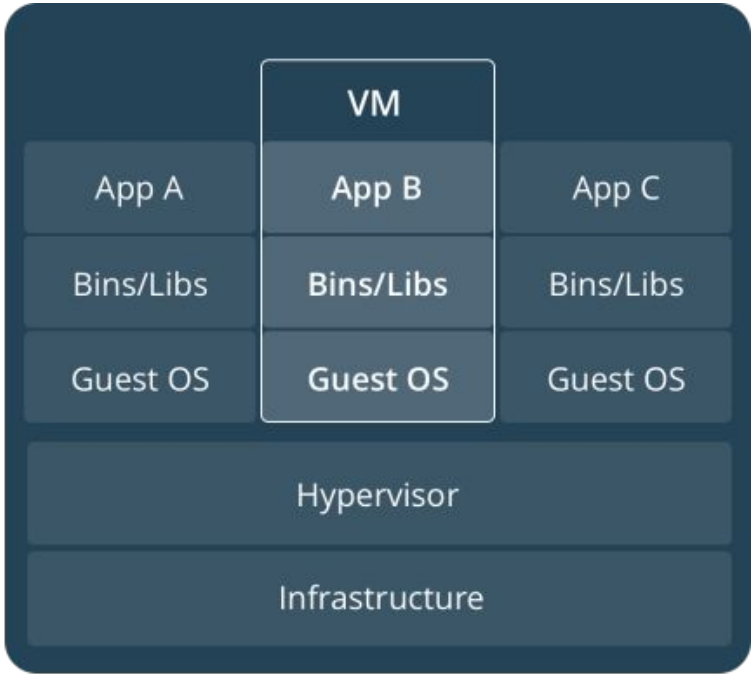
Cemal Ünal

- Software Engineer @ Havelsan Inc.
  - Cloud Native Application Development / Delivery / Monitoring

- cemalunal@yahoo.com
- https://www.linkedin.com/in/cemalunal/

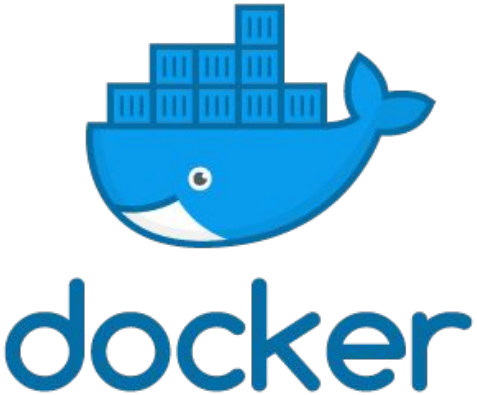
Containers vs VMs



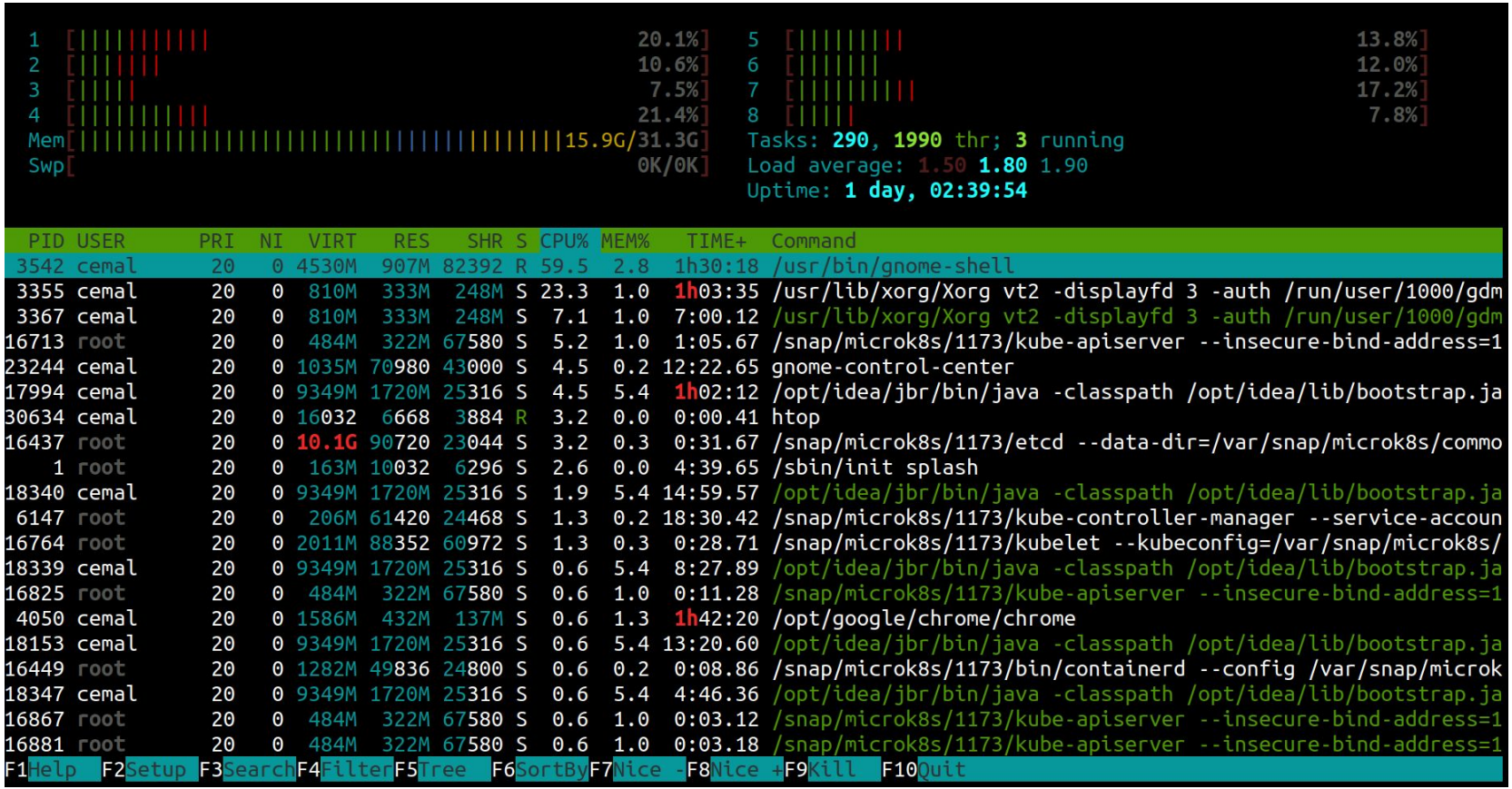


What is Docker ?

- An open platform
- A tool designed to make it easier to create, deploy, and run applications by using containers.



htop Inside the OS



htop Inside a Container

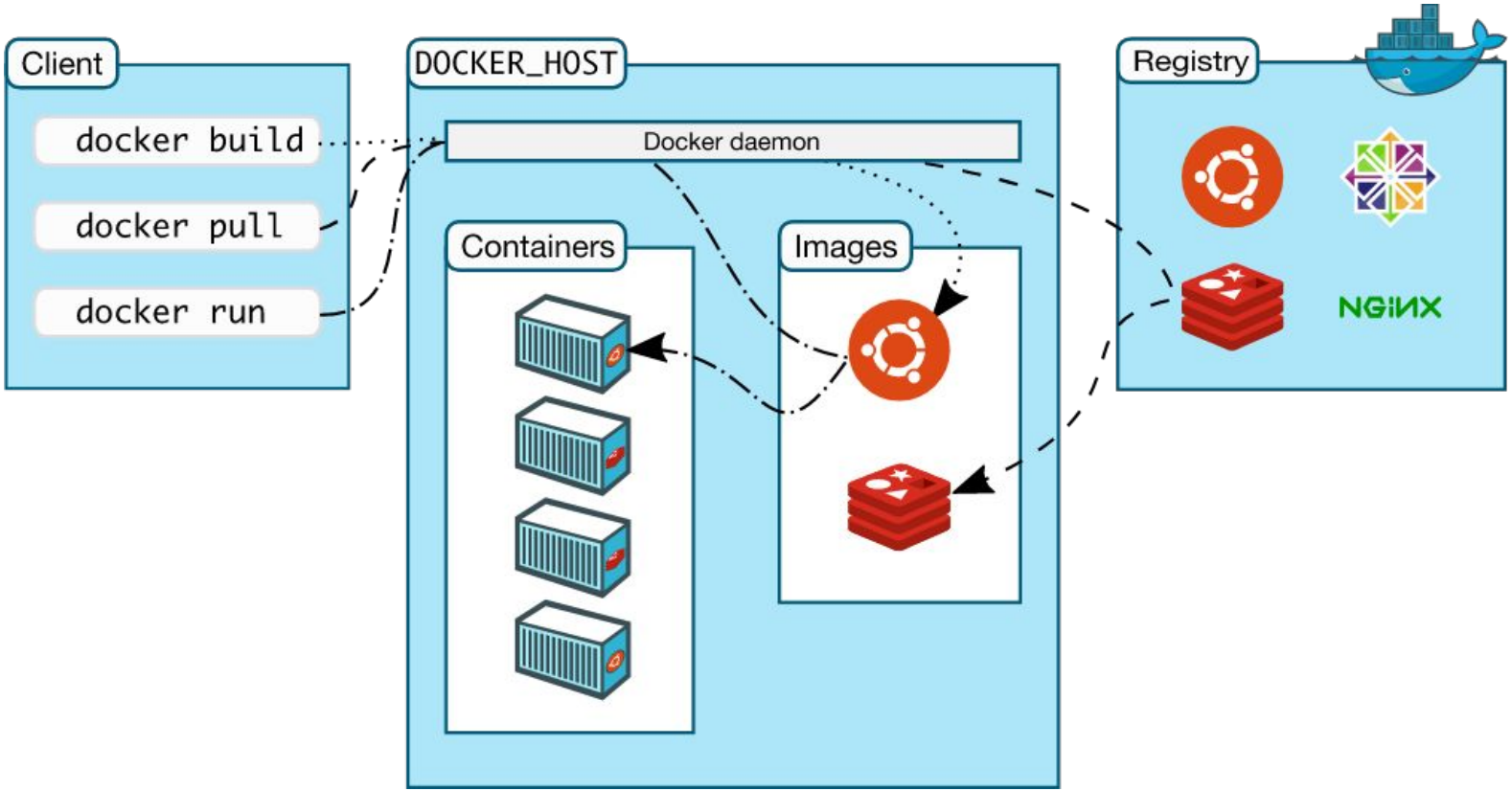
Docker Container Example

```
➜  ~ 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 Client - Host Communication



Docker Image Creation (Docker commit)

```
➜  ~ docker run -it  --name greeting ubuntu:18.04 bash
root@76ba099af323:/# echo hello > hello.txt
root@76ba099af323:/# cat hello.txt
hello
root@76ba099af323:/# exit
exit
➜  ~ docker commit greeting ubuntu:18.04-greeting
sha256:492cc01917eb38c2832084cafc8a0eb07a996c175a49d1e1a70b2b50aab89ad3
➜  ~ docker run -it  --name greeting-new ubuntu:18.04-greeting bash
root@70b58835b46b:/# cat hello.txt
hello
root@70b58835b46b:/#
```

Docker Image Creation (Declarative Approach - Dockerfile)

Dockerfile

```
1  FROM ubuntu:18.04
2
3  RUN echo hello > hello.txt
4
5
```

```
➜  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

```
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  VOLUME /tmp
18
19  # copy over the built artifact from the maven image
20  COPY --from=maven /app/target/*.jar /app/target/
21
22  # set the startup command to run binary
23  CMD java ${JAVA_OPTS} -jar  /app/target/*.jar
24
```

TODO: Multi from explanation

Docker Benefits

- Build once, run anywhere (Platform independent)
- Isolated and Disposable Applications
- Rapid Deployment of the applications
- Scale up & down fast

Docker Use Cases

- Deployment of multiple microservices
- Running different versions of the same application at the same time
- Easily switch between different versions of a deployment

Container Management Solutions

- Docker Compose (todo: add example yaml file)
- Docker Swarm
- Kubernetes
- Amazon Container Service
- Google Container Engine
- Azure App Services

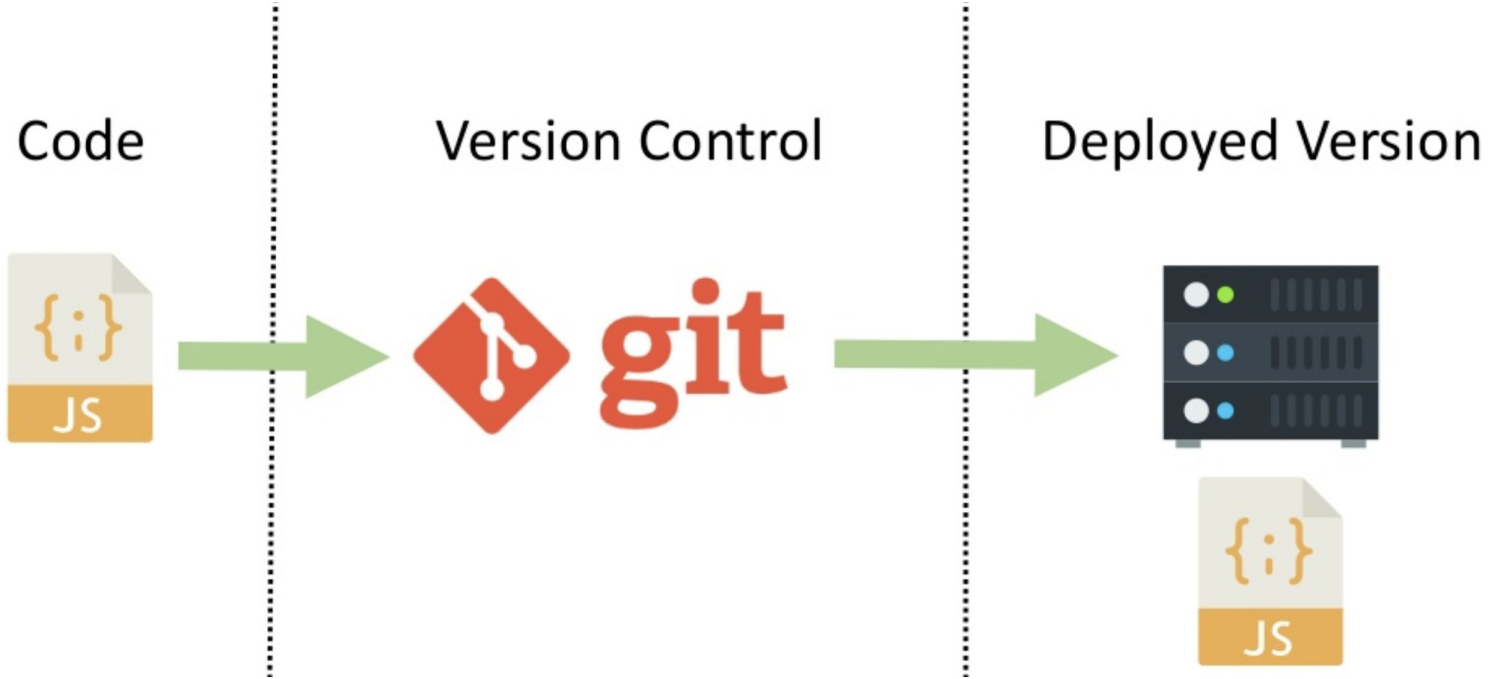
Some of the Keywords

- **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

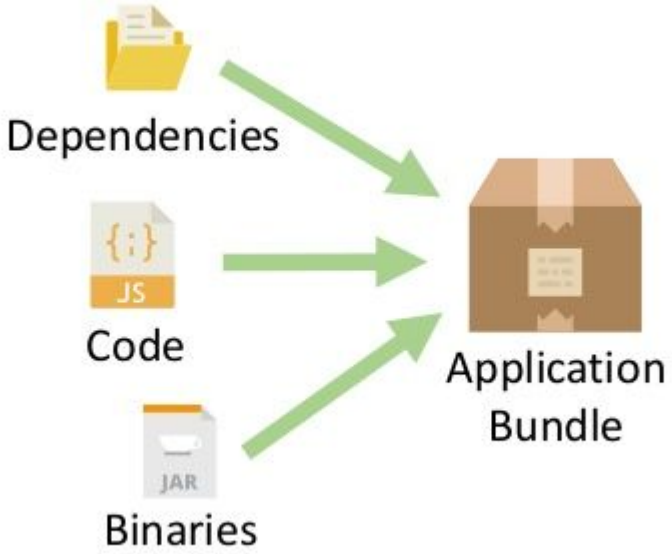
Sample CRUD App and 12-factor App

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

1- Codebase



2- 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

- 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}

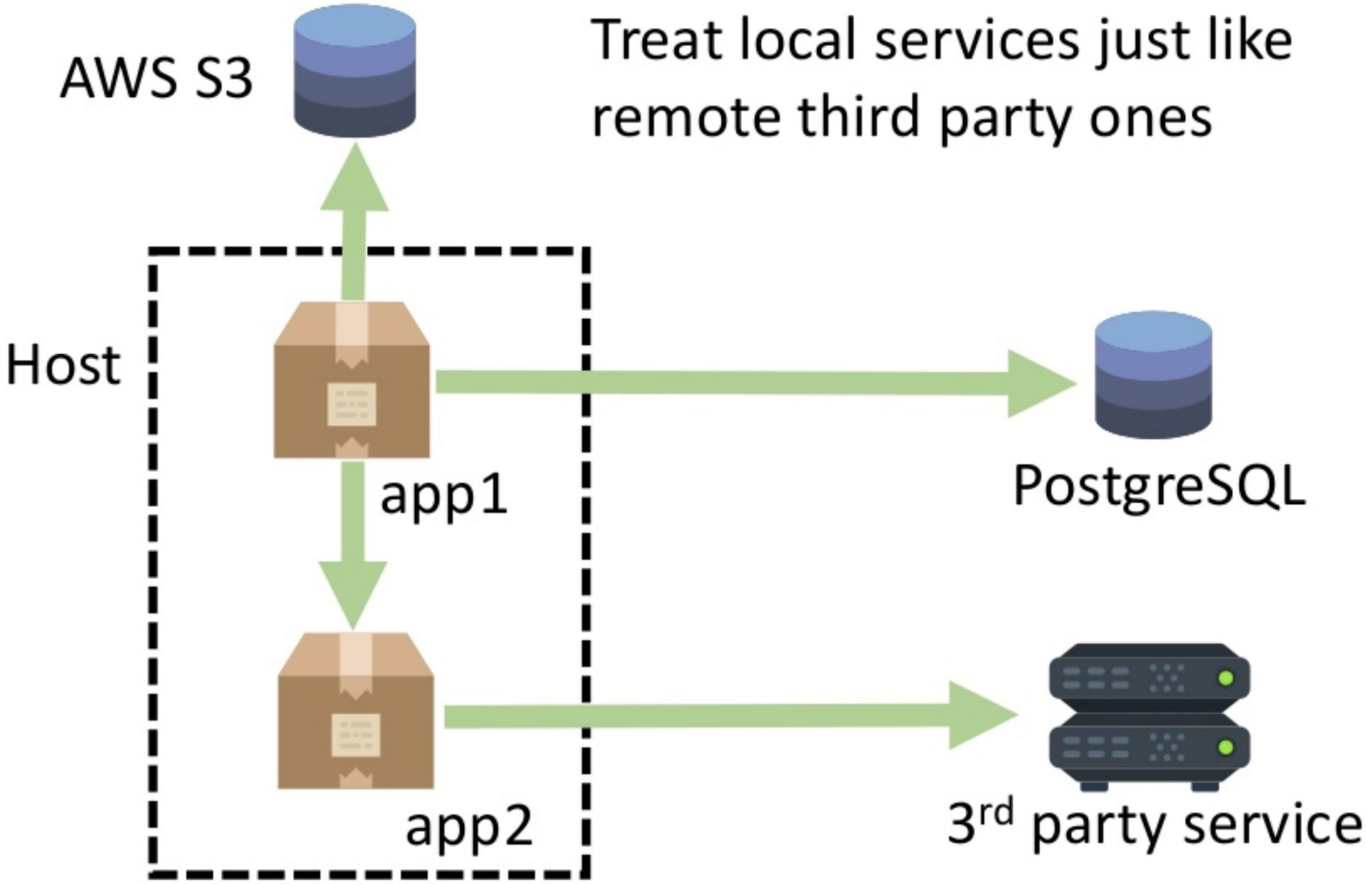
Backend container gets config from the environment

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



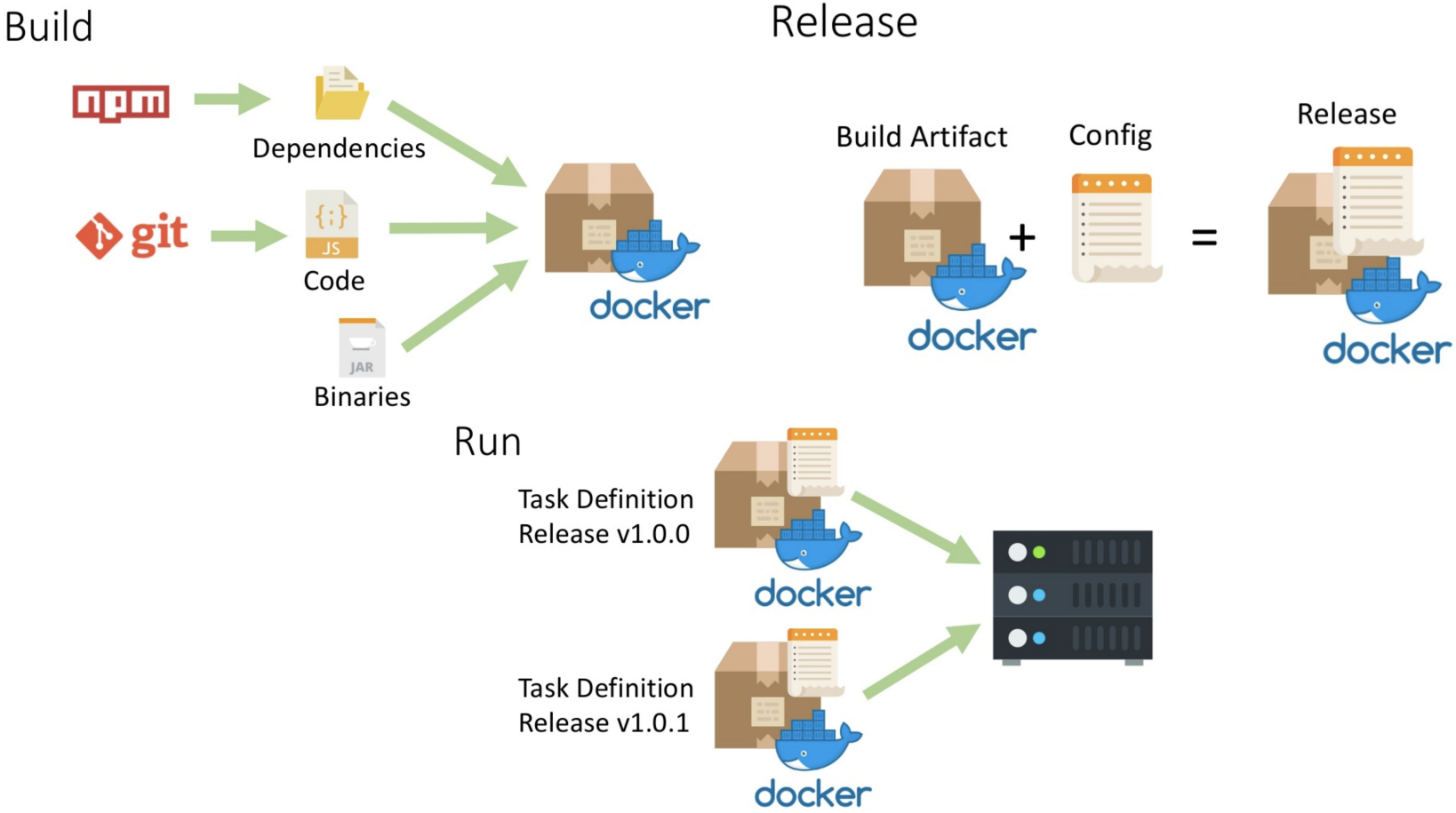
MongoDB connection for Backend

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

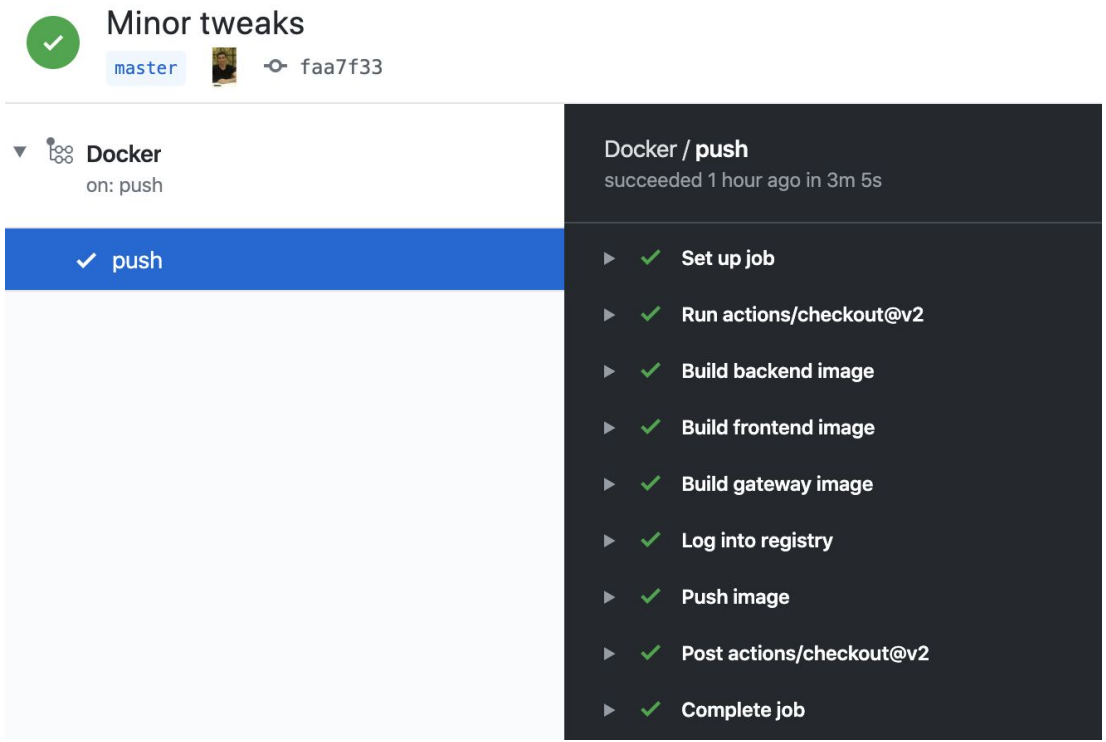
- We can easily switch between local and production MongoDB databases. Or we can even use Azure Cosmos DB by just changing the connection string. Examples:

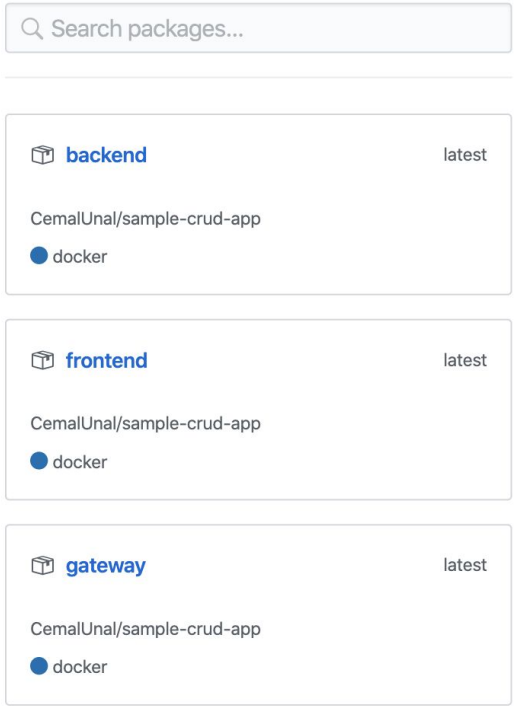
    - mongodb://localhost:27017/sample-app
    - mongodb://mongodb:27017/sample-app
    - mongodb://user:pass@test.documents.azure.com:10255/dbname?ssl=true

5- Build, release, run



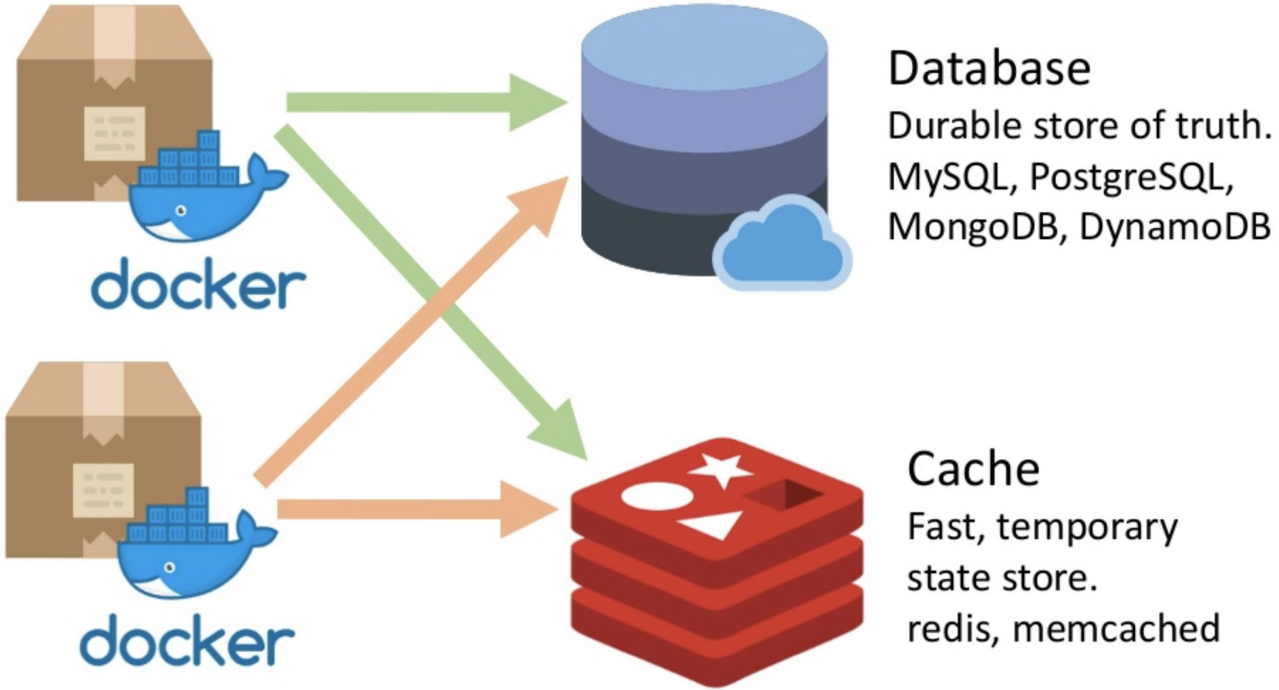
Build - Release





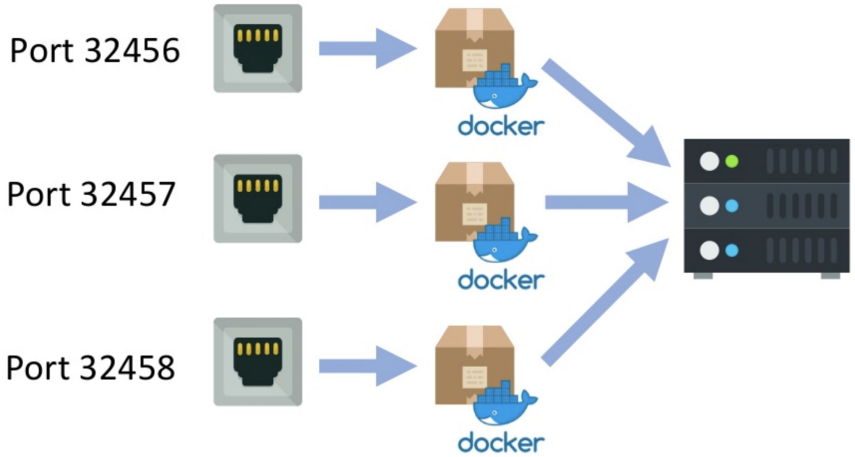
Run TODO:

6- Processes



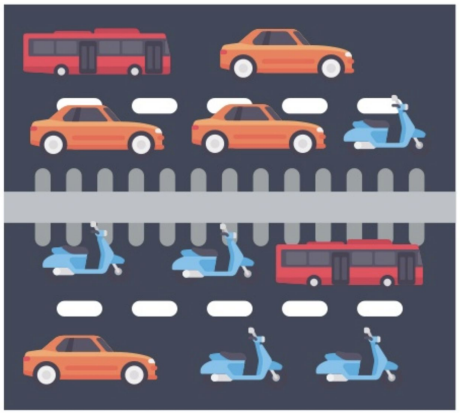
- The application delegates stateful persistence to MongoDB.
- It is easily scalable since it is stateless.

7- Port binding

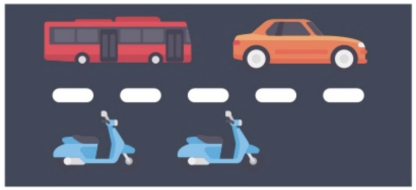


- Backend and Gateway
  - Spring Boot is used along with embedded Jetty server.
  - server.port=${SERVER_PORT} in application-deployment.properties
- Frontend
  - serve npm package is used to serve the static frontend
  - serve -l $SERVER_PORT -s build in startup.sh

8- Concurrency



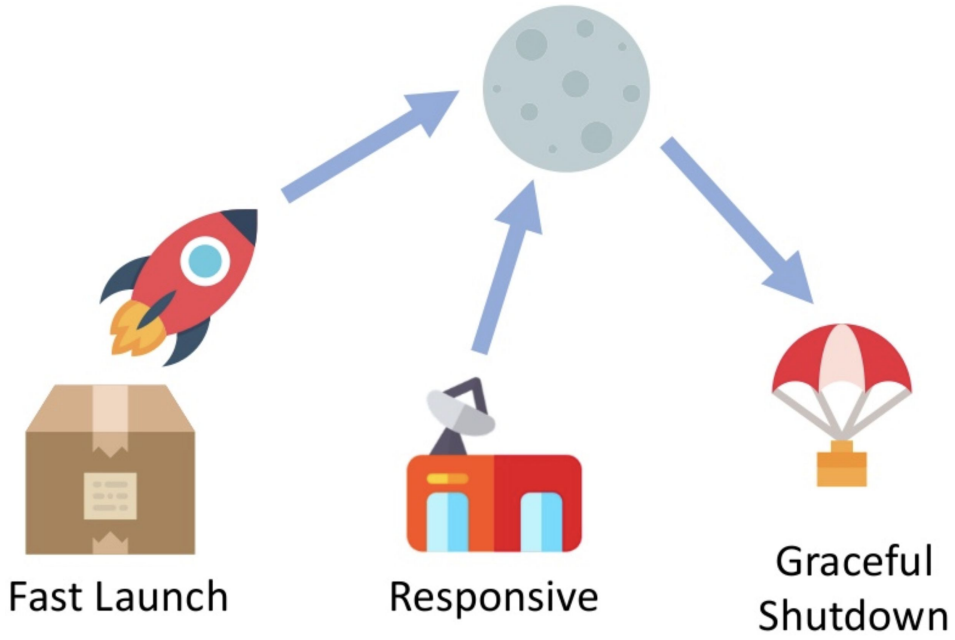
Large Host = More Concurrent Processes



Small Host = Fewer Concurrent Processes

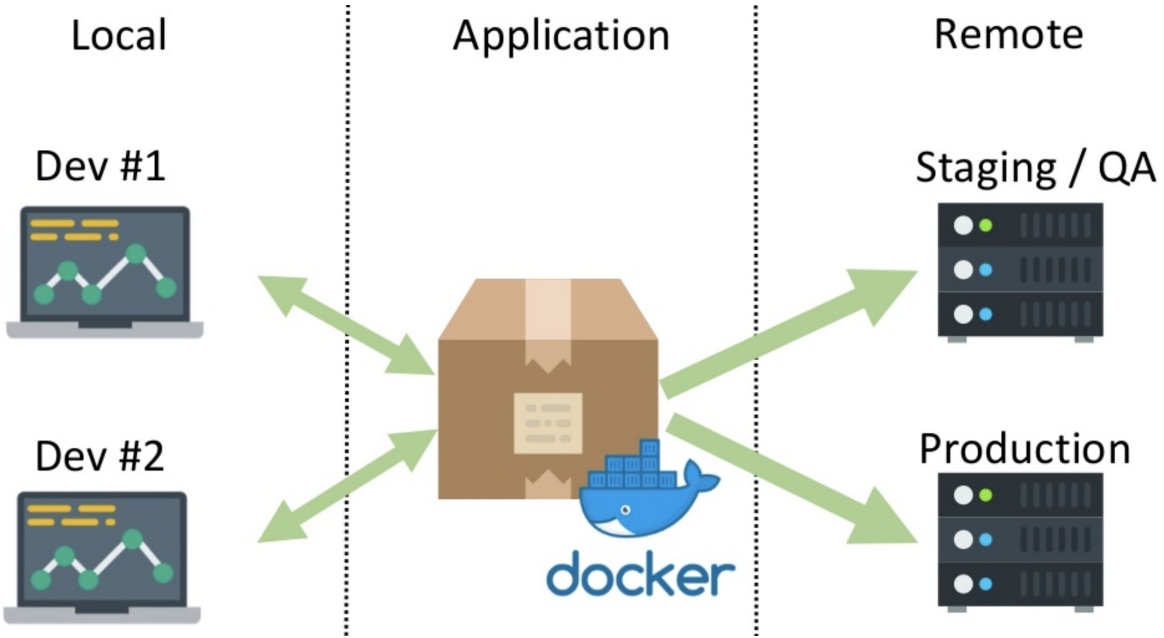
- All components of the application is dockerized
- Launching multiple instances is really simple.

9- Disposability



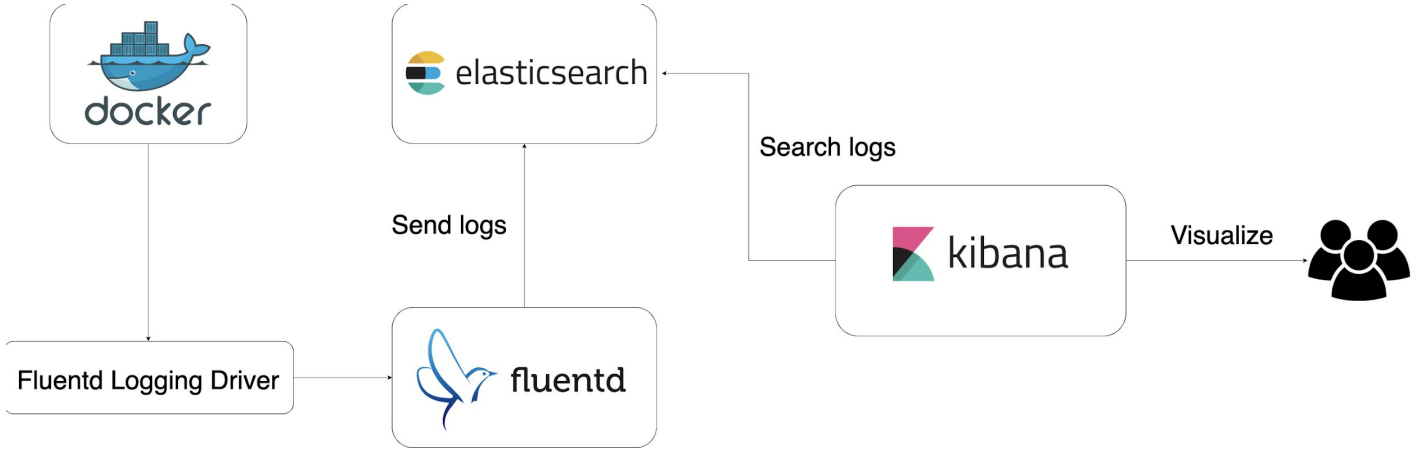
- All components of the sample application are disposable and can be started and stopped quickly
- They shut down gracefully when they receive SIGTERM
- TODO: startup / shutdown logs ?

10- Dev / prod parity



- **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




```
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
```

12- Admin processes

THANKS!