



Introduction to Docker & 12 Factor App Implementation Using Docker

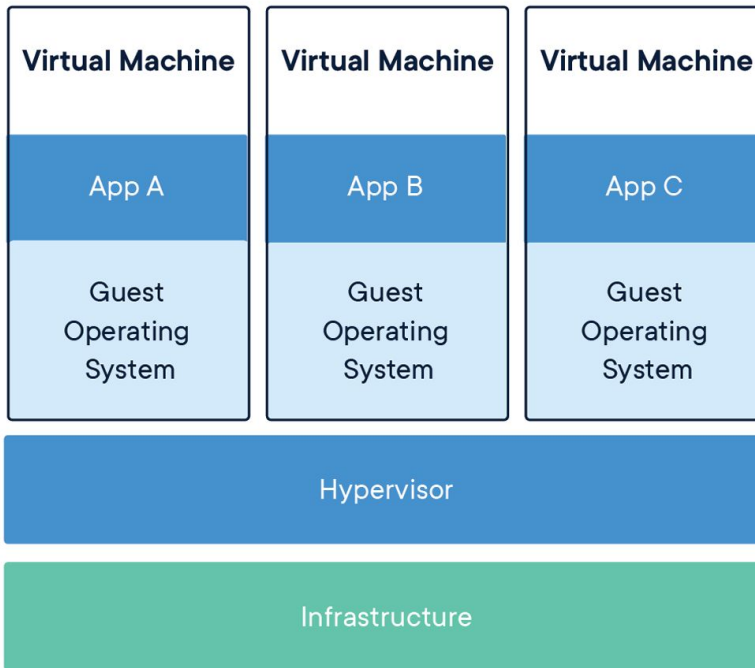
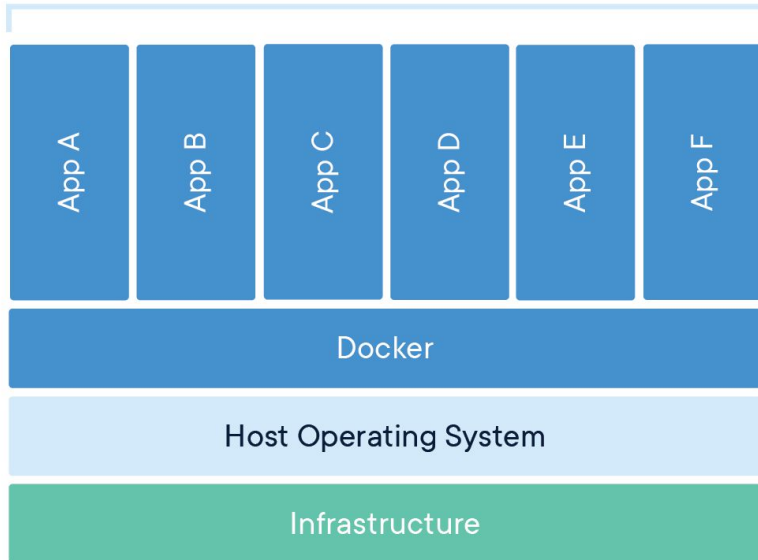


Cemal Ünal

- Software Engineer @ Havelsan Inc.
 - Big Data Analytics & AI Team
- cemalunal@yahoo.com
- <https://www.linkedin.com/in/cemalunal/>

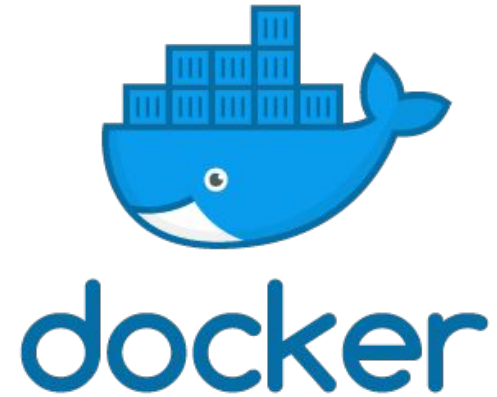
Containers vs Virtual Machines

Containerized Applications

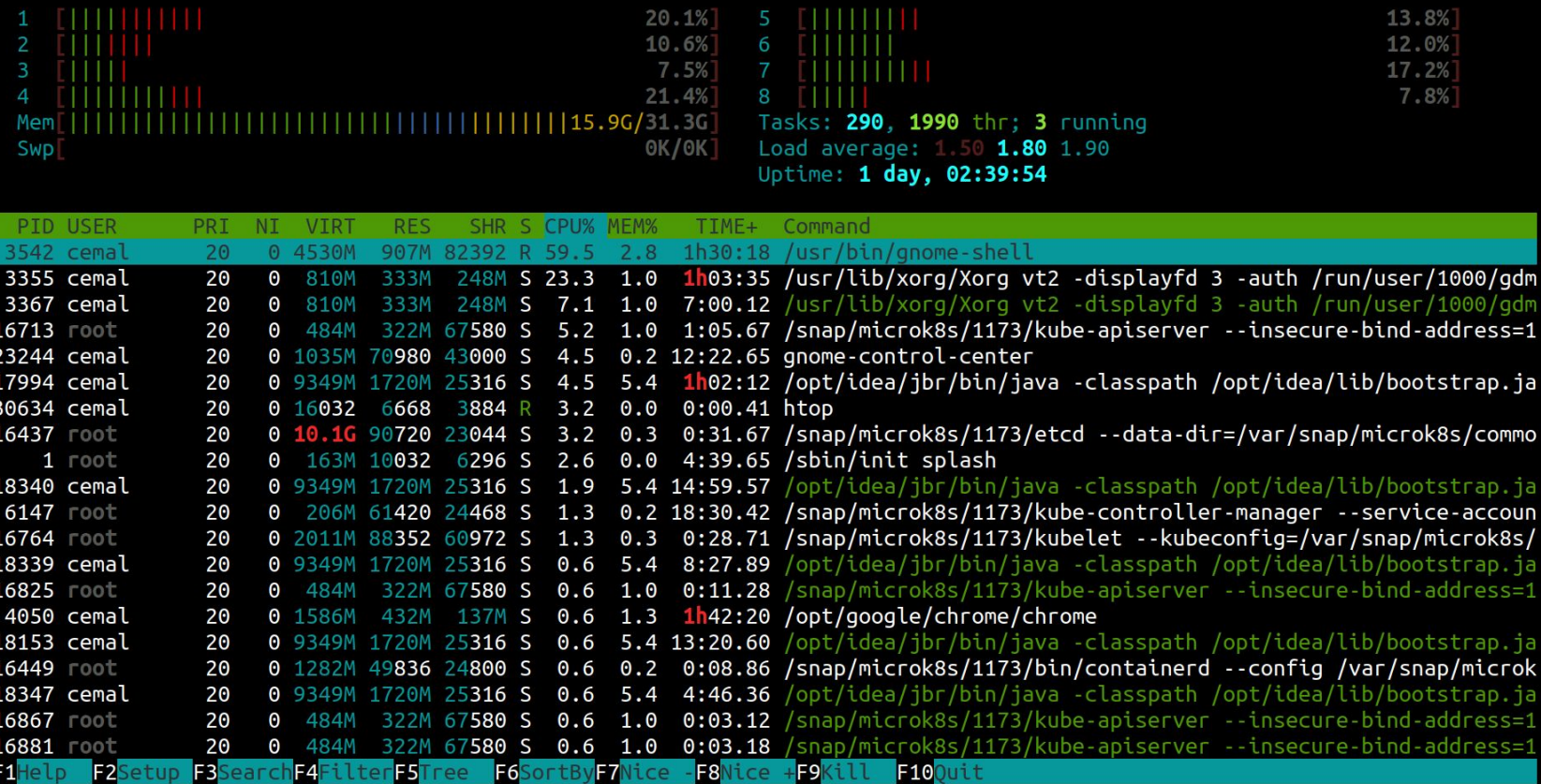


What is Docker ?

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



htop Inside the OS

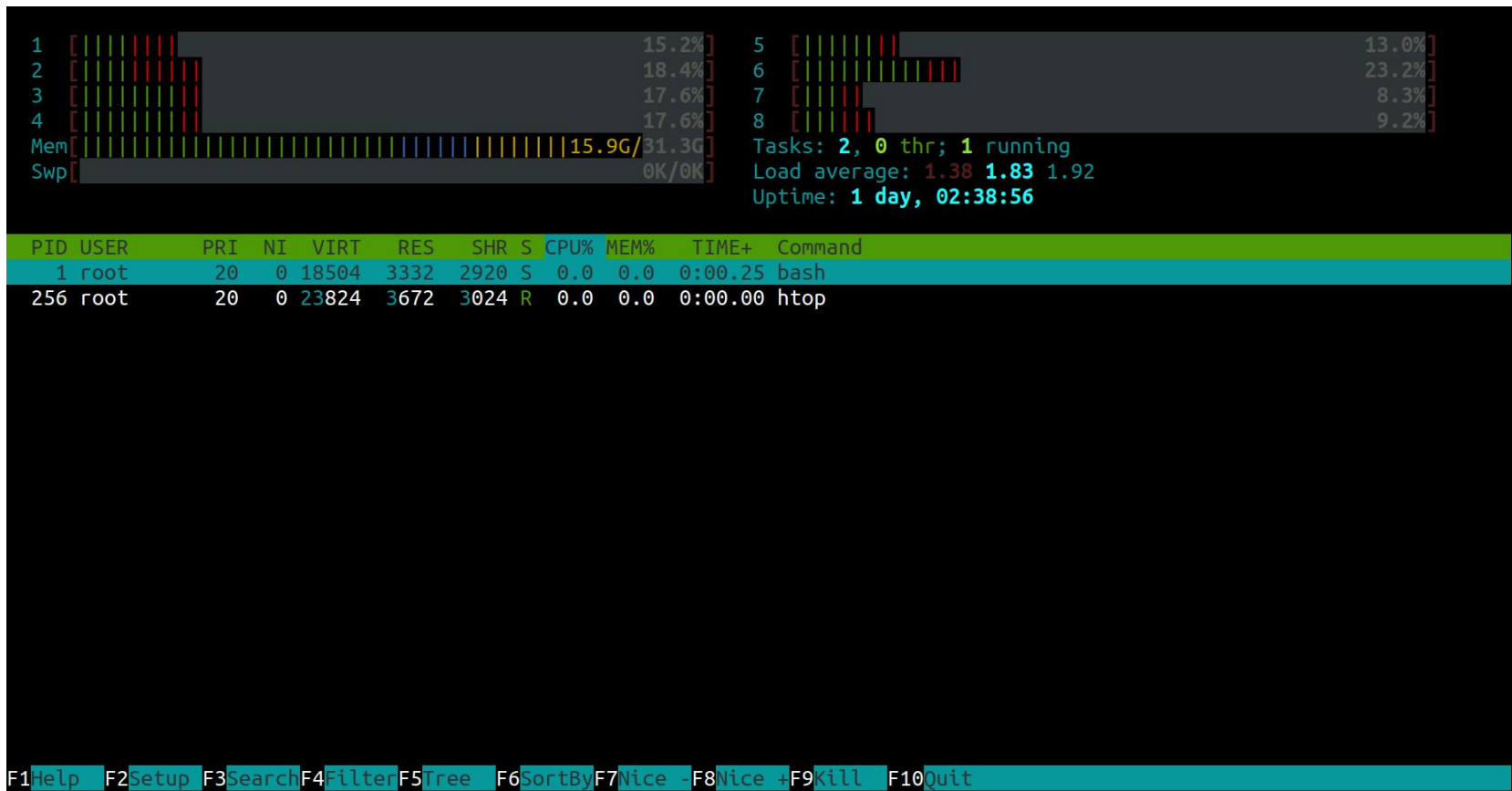


The screenshot shows the htop interface. At the top, system statistics are displayed: 1 CPU at 20.1%, 2 at 10.6%, 3 at 7.5%, 4 at 21.4%, 5 at 13.8%, 6 at 12.0%, 7 at 17.2%, and 8 at 7.8%. Memory usage is 15.9G/31.3G and swap is 0K/0K. System tasks are 290, with 1990 threads and 3 running. Load averages are 1.50, 1.80, and 1.90. Uptime is 1 day, 02:39:54.


PID	USER	PRI	NI	VIRT	RES	SHR	S	CPU%	MEM%	TIME+	Command
3542	cemal	20	0	4530M	907M	82392	R	59.5	2.8	1h30:18	/usr/bin/gnome-shell
3355	cemal	20	0	810M	333M	248M	S	23.3	1.0	1h03:35	/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm
3367	cemal	20	0	810M	333M	248M	S	7.1	1.0	7:00:12	/usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm
16713	root	20	0	484M	322M	67580	S	5.2	1.0	1:05:67	/snap/microk8s/1173/kube-apiserver --insecure-bind-address=1
23244	cemal	20	0	1035M	70980	43000	S	4.5	0.2	12:22:65	gnome-control-center
17994	cemal	20	0	9349M	1720M	25316	S	4.5	5.4	1h02:12	/opt/idea/jbr/bin/java -classpath /opt/idea/lib/bootstrap.jar
30634	cemal	20	0	16032	6668	3884	R	3.2	0.0	0:00:41	htop
16437	root	20	0	10.1G	90720	23044	S	3.2	0.3	0:31:67	/snap/microk8s/1173/etcd --data-dir=/var/snap/microk8s/commo
1	root	20	0	163M	10032	6296	S	2.6	0.0	4:39:65	/sbin/init splash
18340	cemal	20	0	9349M	1720M	25316	S	1.9	5.4	14:59:57	/opt/idea/jbr/bin/java -classpath /opt/idea/lib/bootstrap.jar
6147	root	20	0	206M	61420	24468	S	1.3	0.2	18:30:42	/snap/microk8s/1173/kube-controller-manager --service-accoun
16764	root	20	0	2011M	88352	60972	S	1.3	0.3	0:28:71	/snap/microk8s/1173/kubelet --kubeconfig=/var/snap/microk8s/
18339	cemal	20	0	9349M	1720M	25316	S	0.6	5.4	8:27:89	/opt/idea/jbr/bin/java -classpath /opt/idea/lib/bootstrap.jar
16825	root	20	0	484M	322M	67580	S	0.6	1.0	0:11:28	/snap/microk8s/1173/kube-apiserver --insecure-bind-address=1
4050	cemal	20	0	1586M	432M	137M	S	0.6	1.3	1h42:20	/opt/google/chrome/chrome
18153	cemal	20	0	9349M	1720M	25316	S	0.6	5.4	13:20:60	/opt/idea/jbr/bin/java -classpath /opt/idea/lib/bootstrap.jar
16449	root	20	0	1282M	49836	24800	S	0.6	0.2	0:08:86	/snap/microk8s/1173/bin/containerd --config /var/snap/microk
18347	cemal	20	0	9349M	1720M	25316	S	0.6	5.4	4:46:36	/opt/idea/jbr/bin/java -classpath /opt/idea/lib/bootstrap.jar
16867	root	20	0	484M	322M	67580	S	0.6	1.0	0:03:12	/snap/microk8s/1173/kube-apiserver --insecure-bind-address=1
16881	root	20	0	484M	322M	67580	S	0.6	1.0	0:03:18	/snap/microk8s/1173/kube-apiserver --insecure-bind-address=1

F1Help F2Setup F3Search F4Filter F5Tree F6SortBy F7Nice F8Nice +F9Kill F10Quit

htop Inside a Container



Docker Container Example



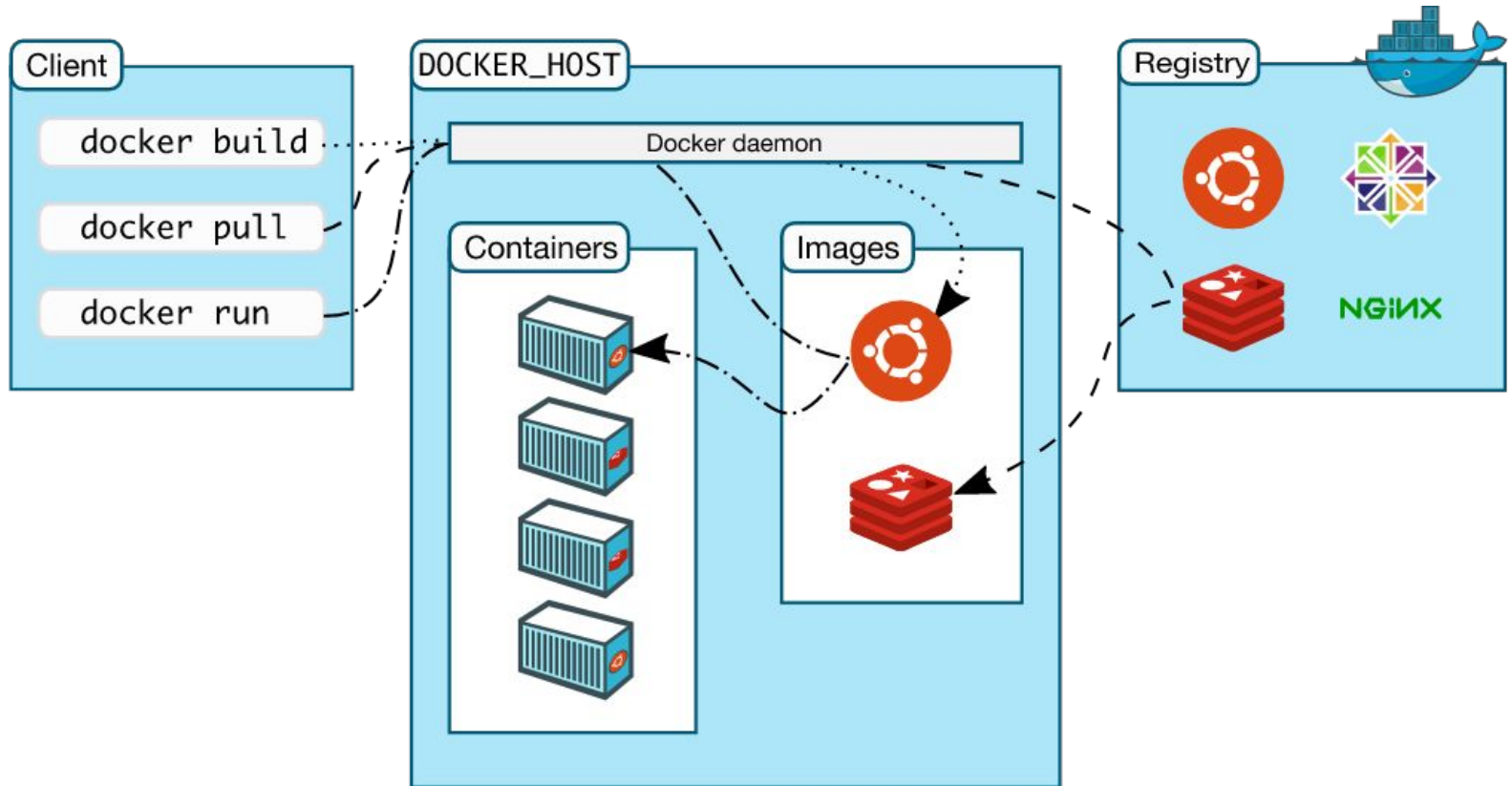
```
→ ~ docker run --name greeting ubuntu:18.04 echo "Hello class of CS443, I'm a container"
Unable to find image 'ubuntu:18.04' locally
18.04: Pulling from library/ubuntu
5c939e3a4d10: Already exists
c63719cdbe7a: Already exists
19a861ea6baf: Already exists
651c9d2d6c4f: Already exists
Digest: sha256:8d31dad0c58f552e890d68bbfb735588b6b820a46e459672d96e585871acc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

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

```
→ backend git:(master) x 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) x 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
```



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
- 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

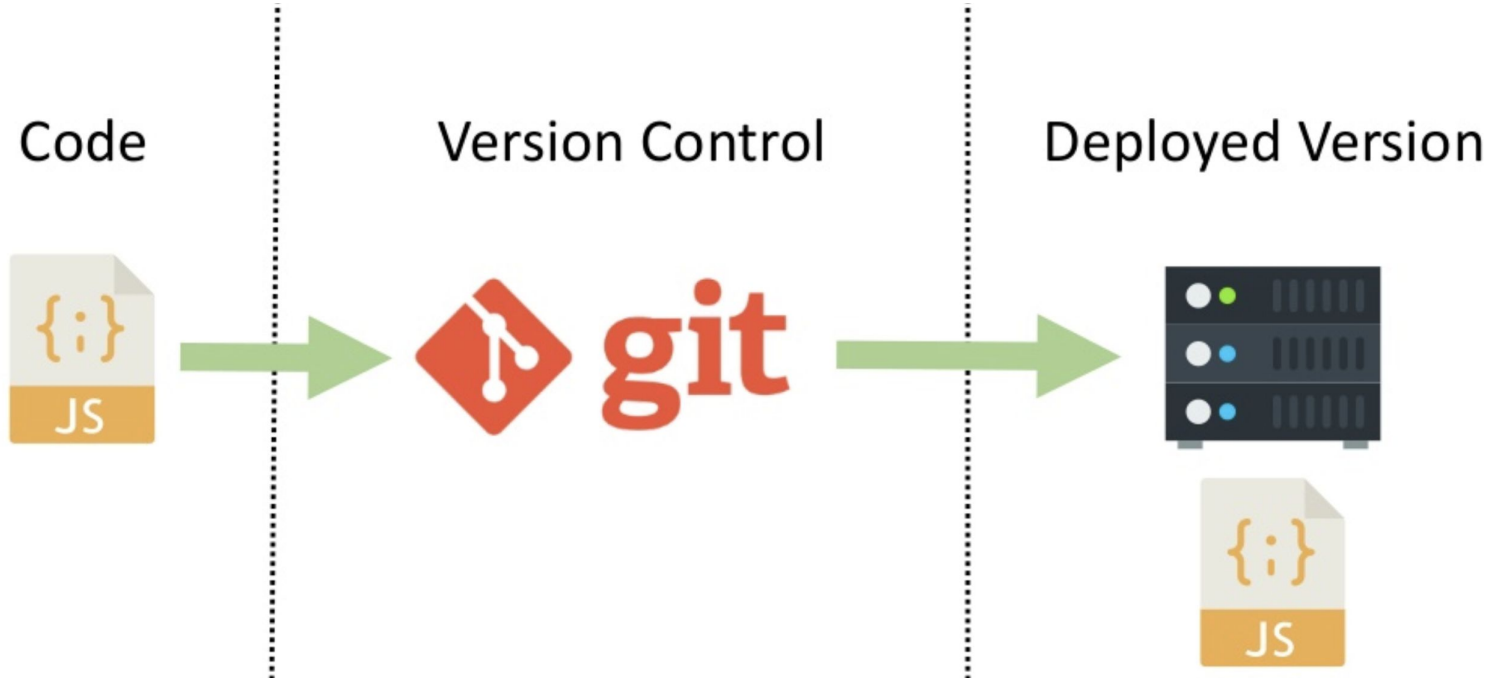


Sample CRUD App and 12-factor App

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

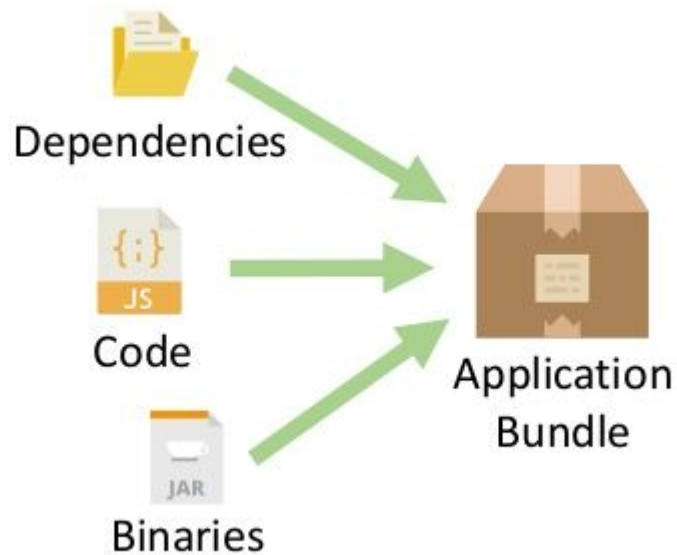
1- Codebase

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(`${window.env.REACT_APP_BACKEND_URI}/customers`)`
- Backend
 - MongoDB connection URI is stored in environment variables and accessed via application-deployment.properties file
 - `spring.data.mongodb.uri=${MONGODB_URI}`

Backend container gets config from the environment

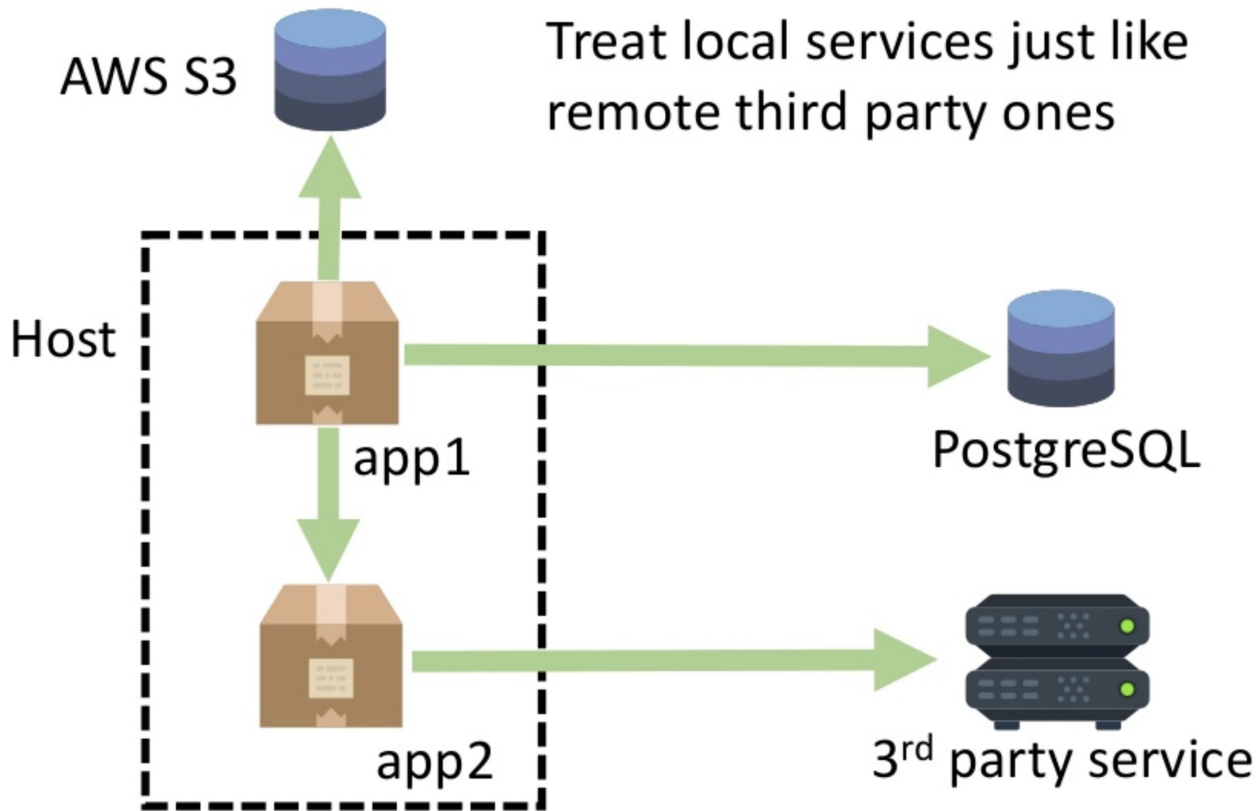


```
application-deployment.properties ✕  
1  server.port=${SERVER_PORT}  
2  spring.data.mongodb.uri=${MONGODB_URI}
```

```
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



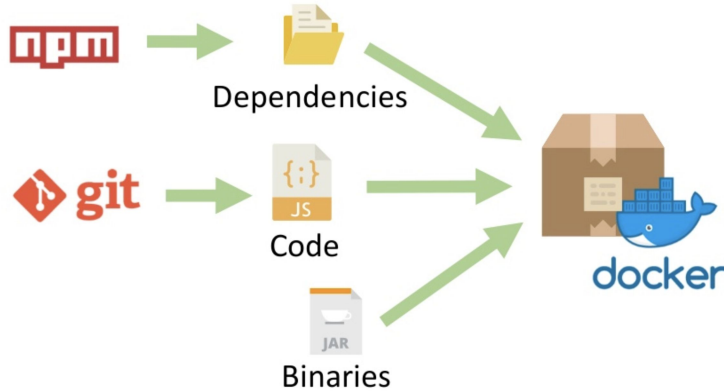


MongoDB connection for Backend

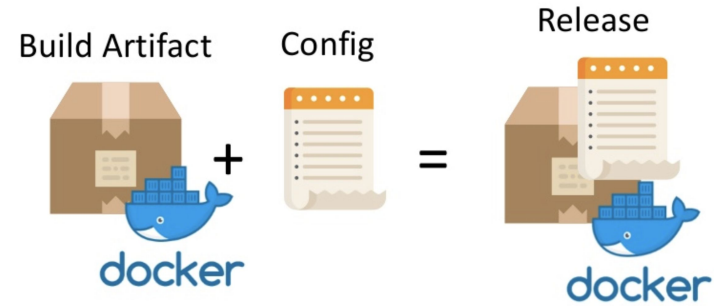
- Think about MongoDB - Connection URI is stored in **MONGODB_URI** environment variable.
- We can easily switch between local and production MongoDB databases. Or we can even use Azure Cosmos DB by just changing the connection string. Examples:
 - `mongodb://localhost:27017/sample-app`
 - `mongodb://mongodb:27017/sample-app`
 - `mongodb://user:pass@test.documents.azure.com:10255/dbname?ssl=true`

5- Build, release, run *Strictly separate build and run stages*

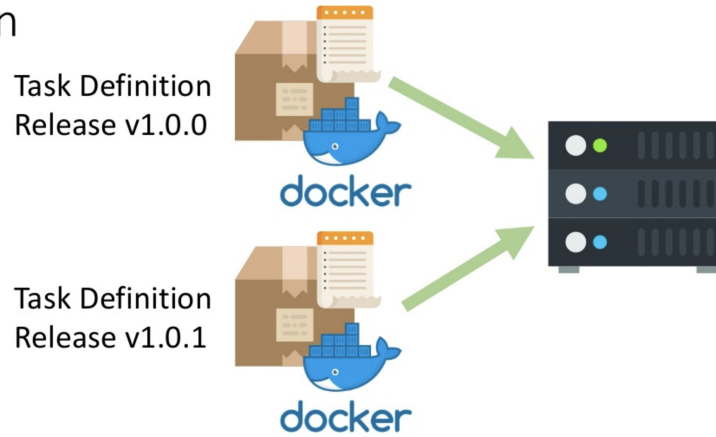
Build



Release



Run



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

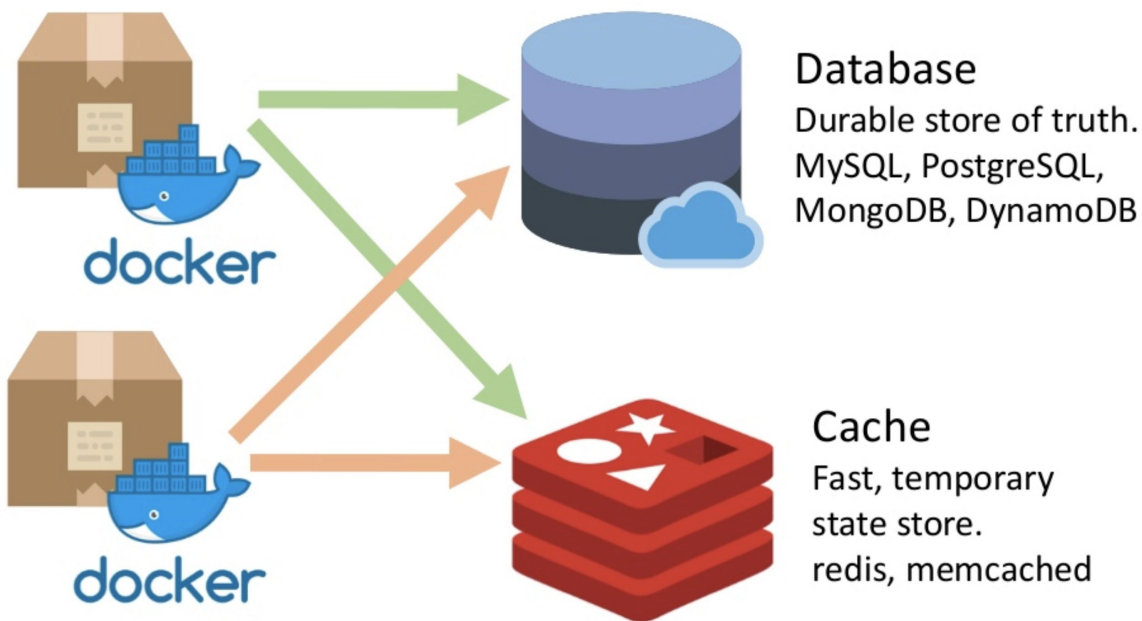
docker



Run TODO:

6- Processes

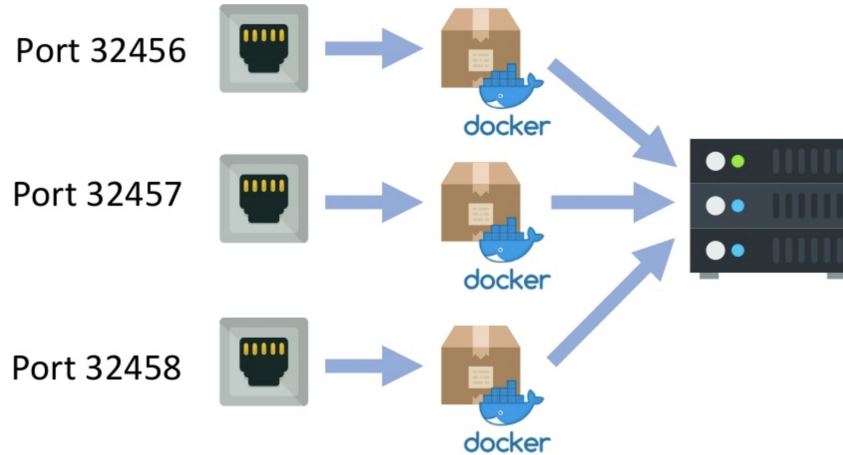
Execute the app as one or more stateless processes



- 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



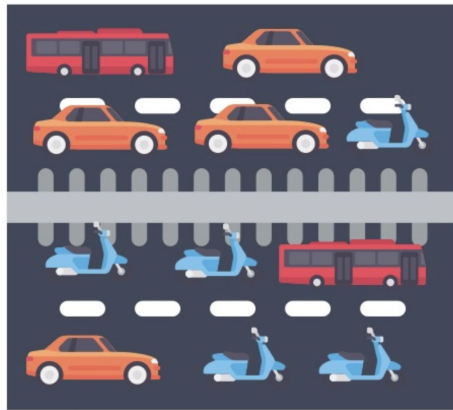
API



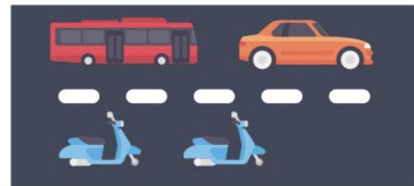
Web



Worker



Large Host = More
Concurrent Processes

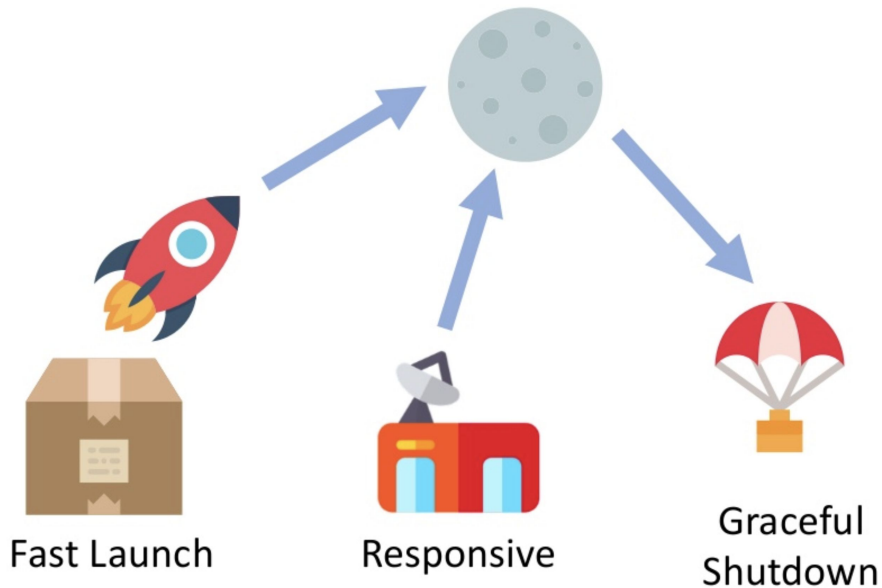


Small Host =
Fewer Concurrent
Processes

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

9- Disposability

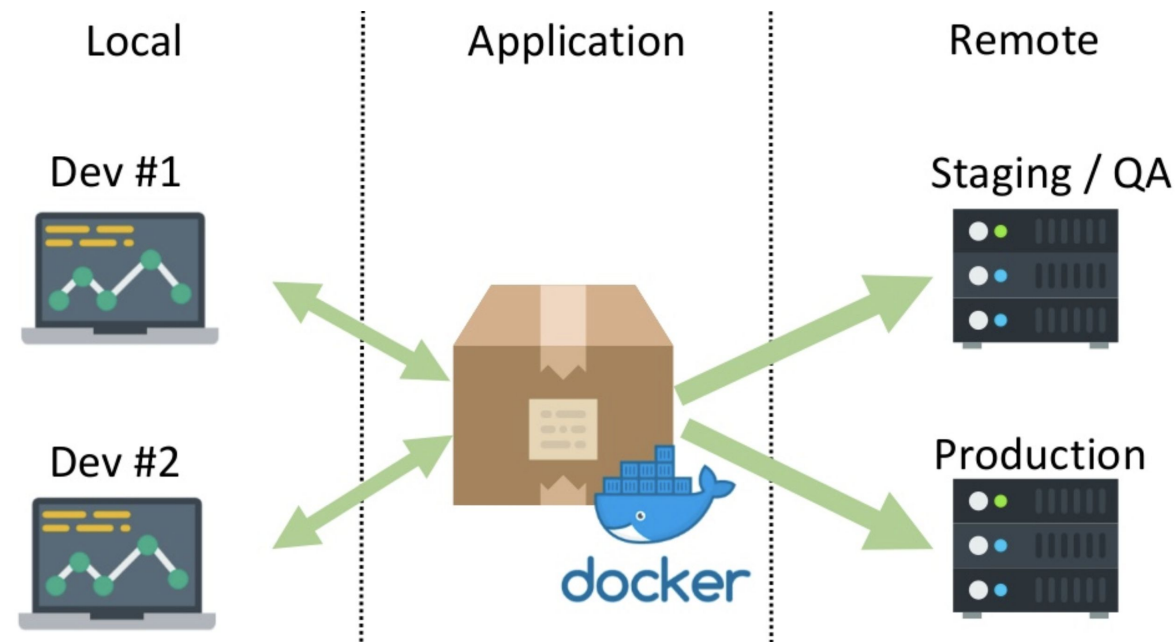
Maximize robustness with fast startup and graceful shutdown



- 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

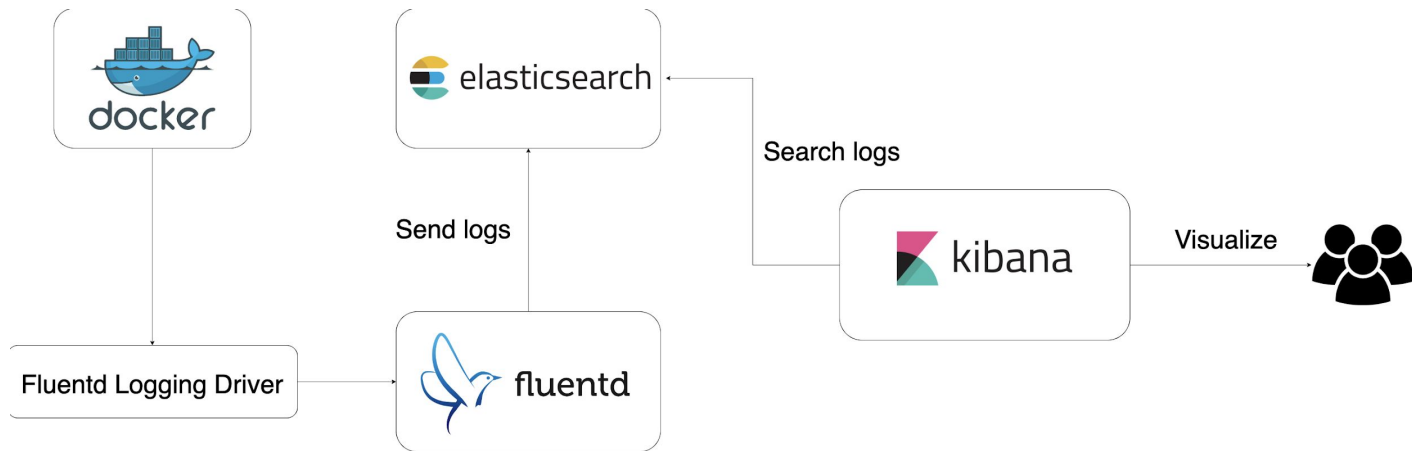
Keep development, staging, and production as similar as possible



- **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
```



12- Admin processes

Run admin/management tasks as one-off processes



THANKS!