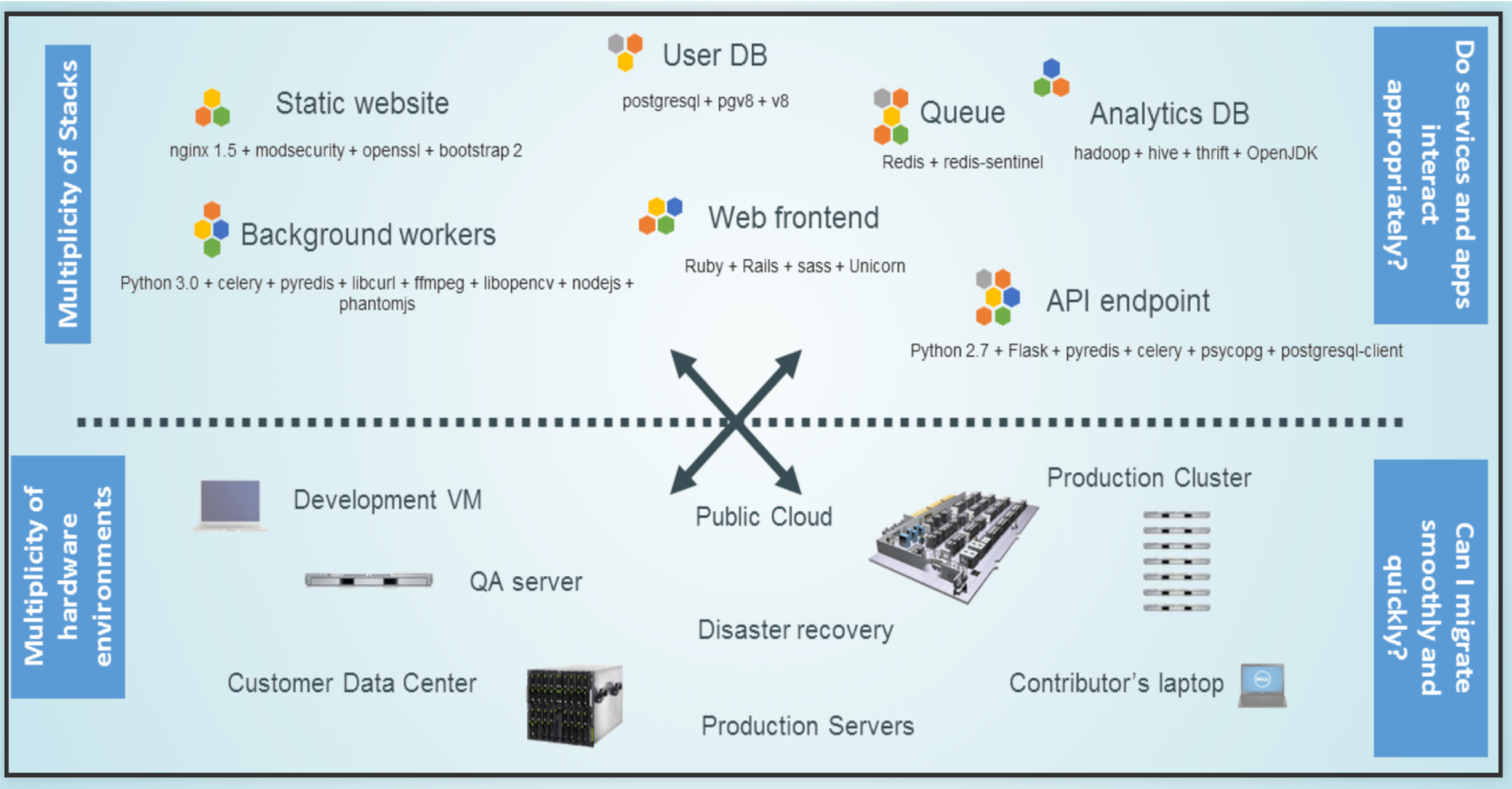
Docker

Docker
The challenge



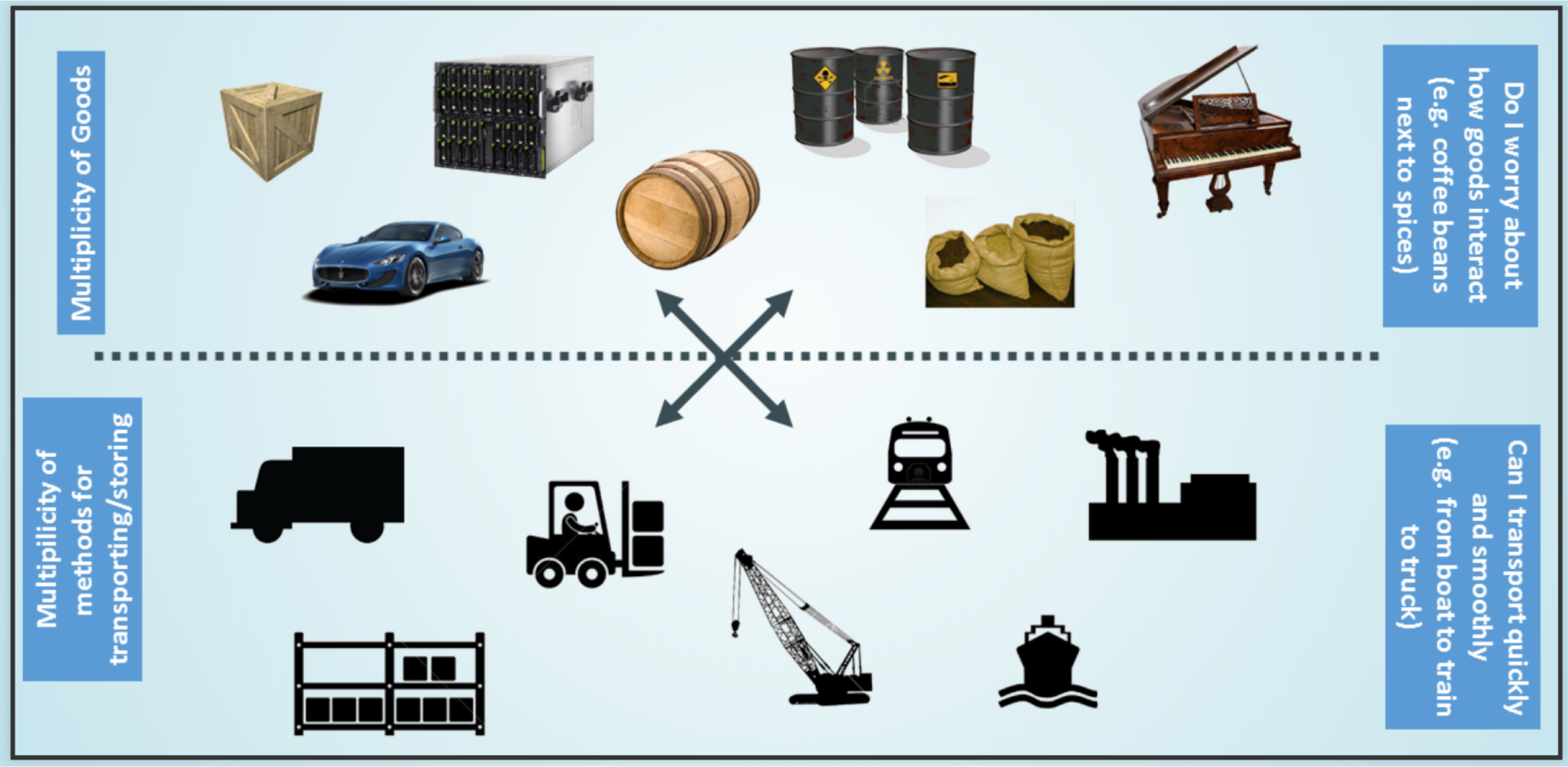
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
The "Matrix from hell"

	Development VM	QA Server	Single Prod Server	Onsite Cluster	Public Cloud	Contributor's laptop	Customer Servers
Static website	?	?	?	?	?	?	?
Web frontend	?	?	?	?	?	?	?
Background workers	?	?	?	?	?	?	?
User DB	?	?	?	?	?	?	?
Analytics DB	?	?	?	?	?	?	?
Queue	?	?	?	?	?	?	?

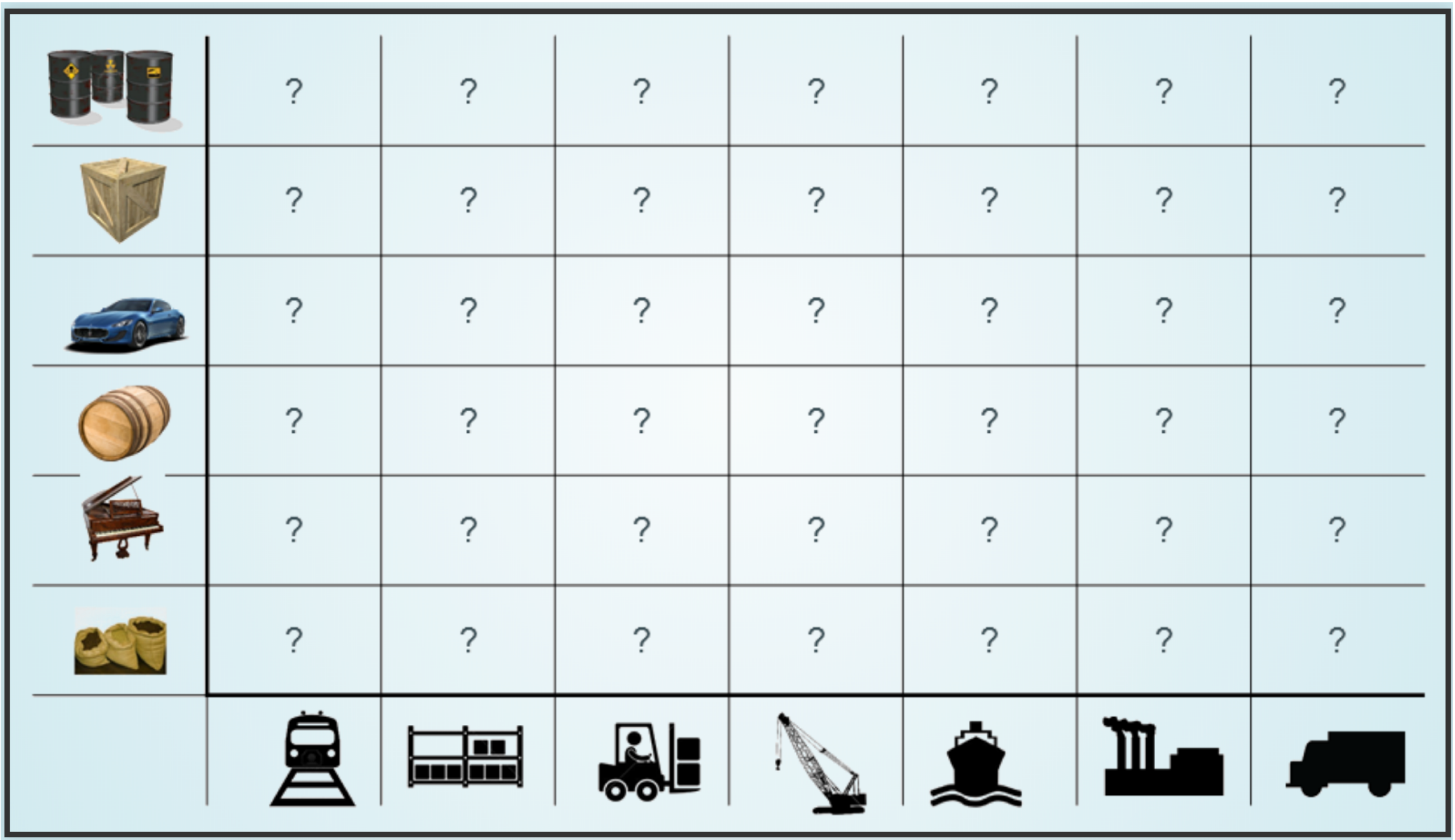
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Cargo Transport Pre-1960



Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Also a Matrix from Hell

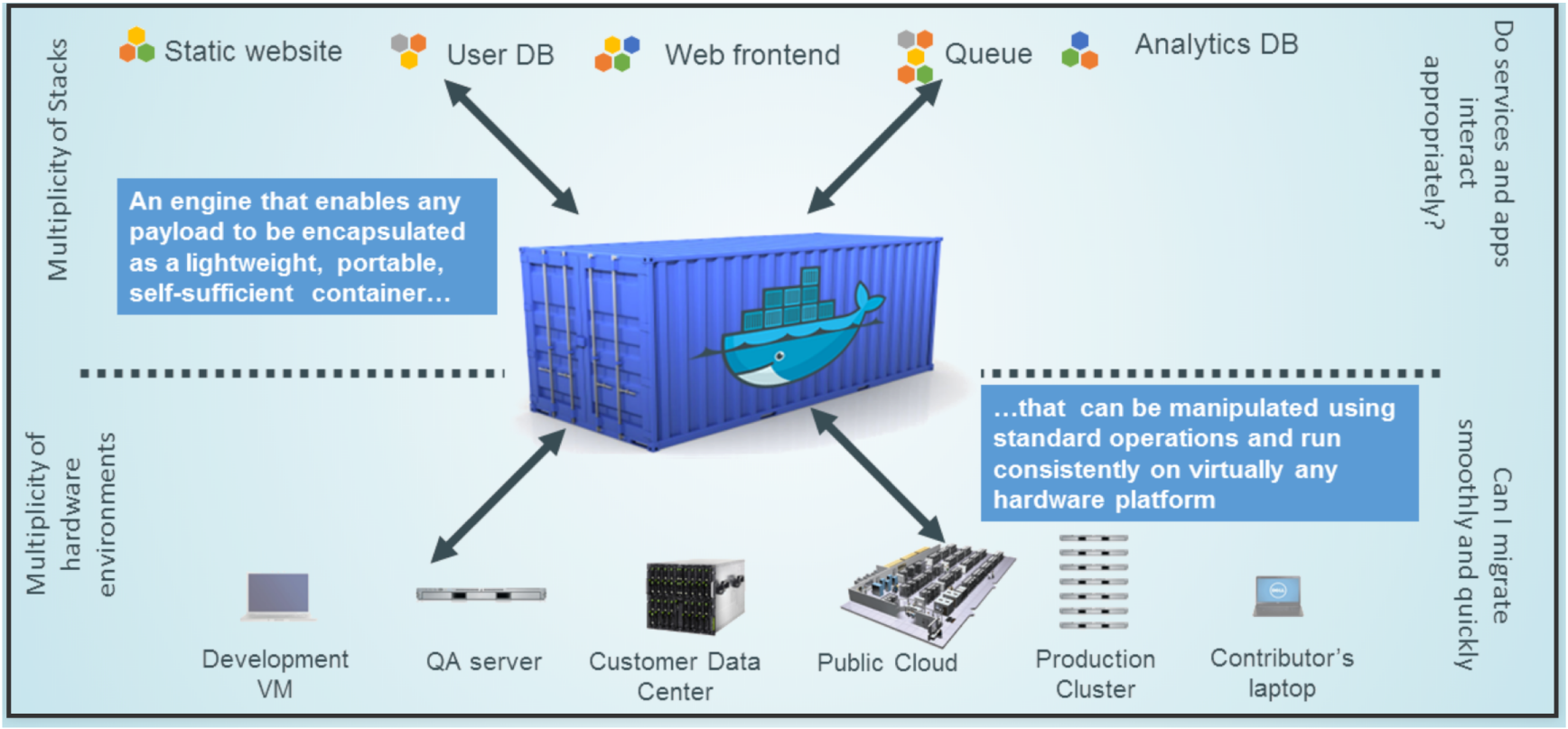


Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Solution: Intermodal Shipping Container

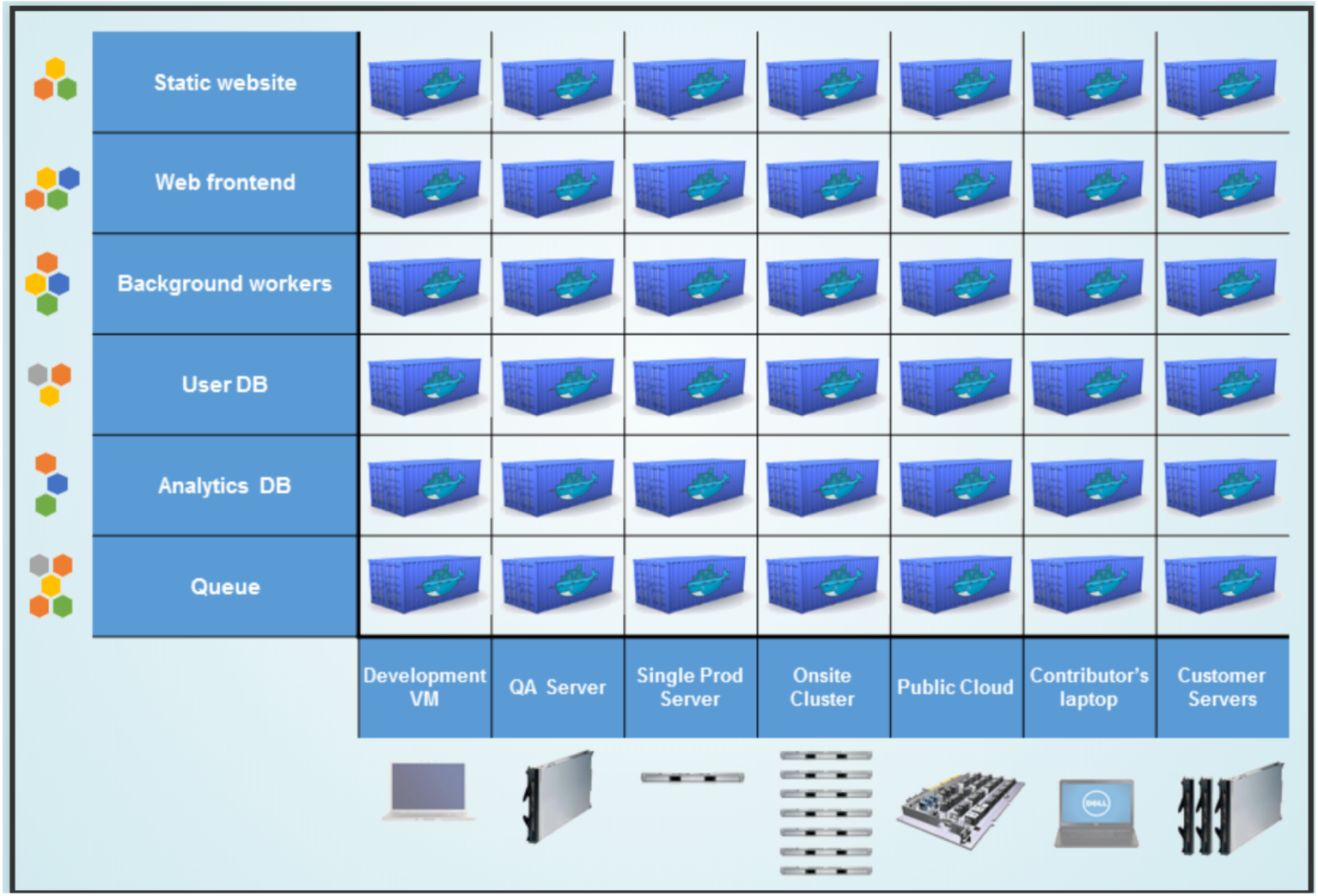
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Docker is a Container System for Code



Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

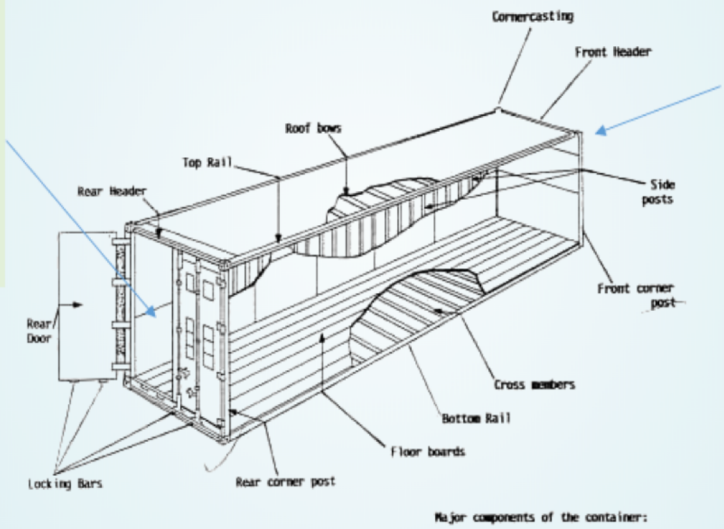
Docker
Docker Eliminates the Matrix from Hell



Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Why it Works: Separation of Concerns

- Dan the Developer
  - Worries about what's "inside" the container
    - His code
    - His Libraries
    - His Package Manager
    - His Apps
    - His Data
  - All Linux servers look the same

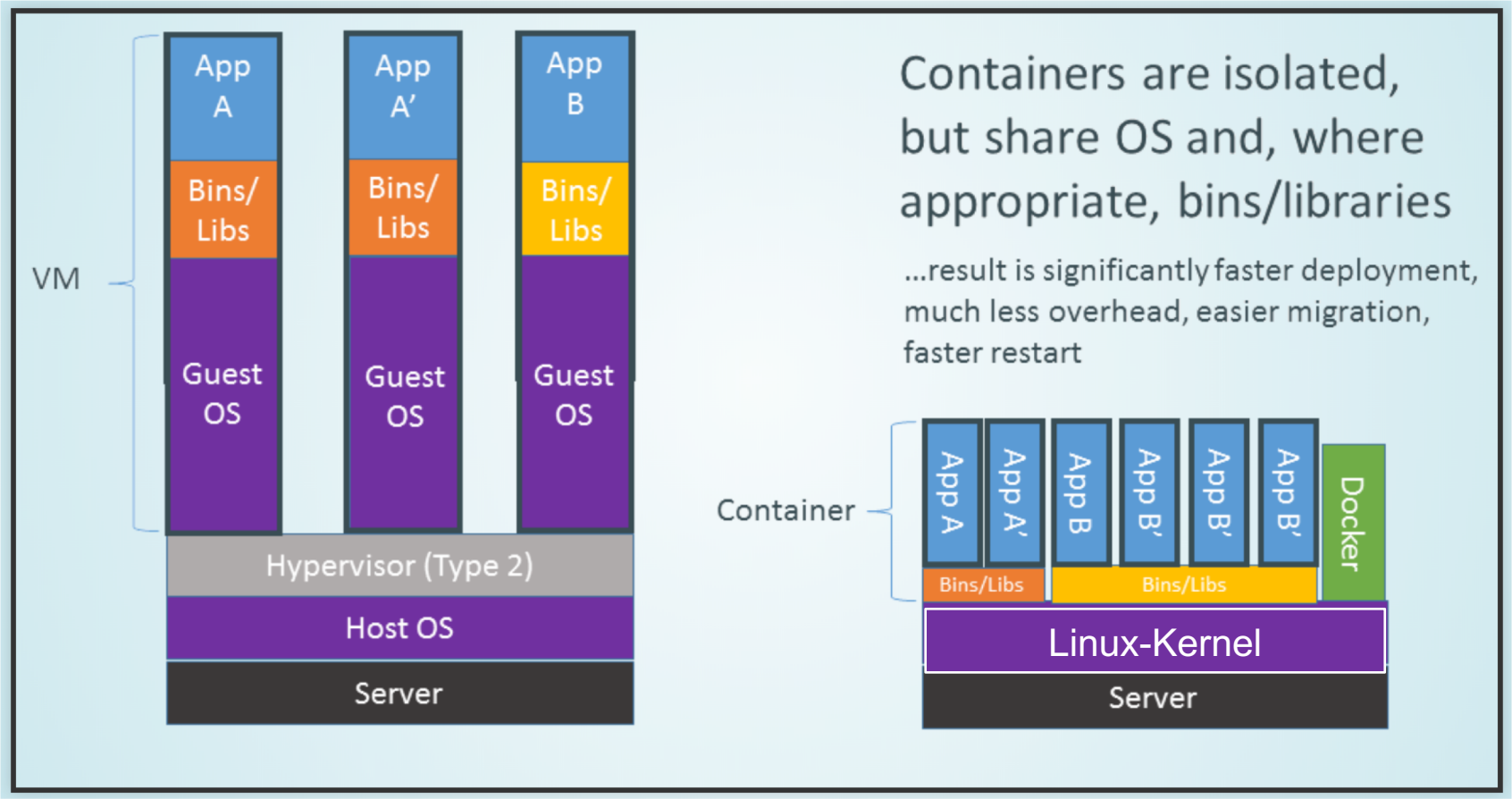


Major components of the container:

- Oscar the Ops Guy
  - Worries about what's "outside" the container
    - Logging
    - Remote access
    - Monitoring
    - Network config
  - All containers start, stop, copy, attach, migrate, etc. the same way

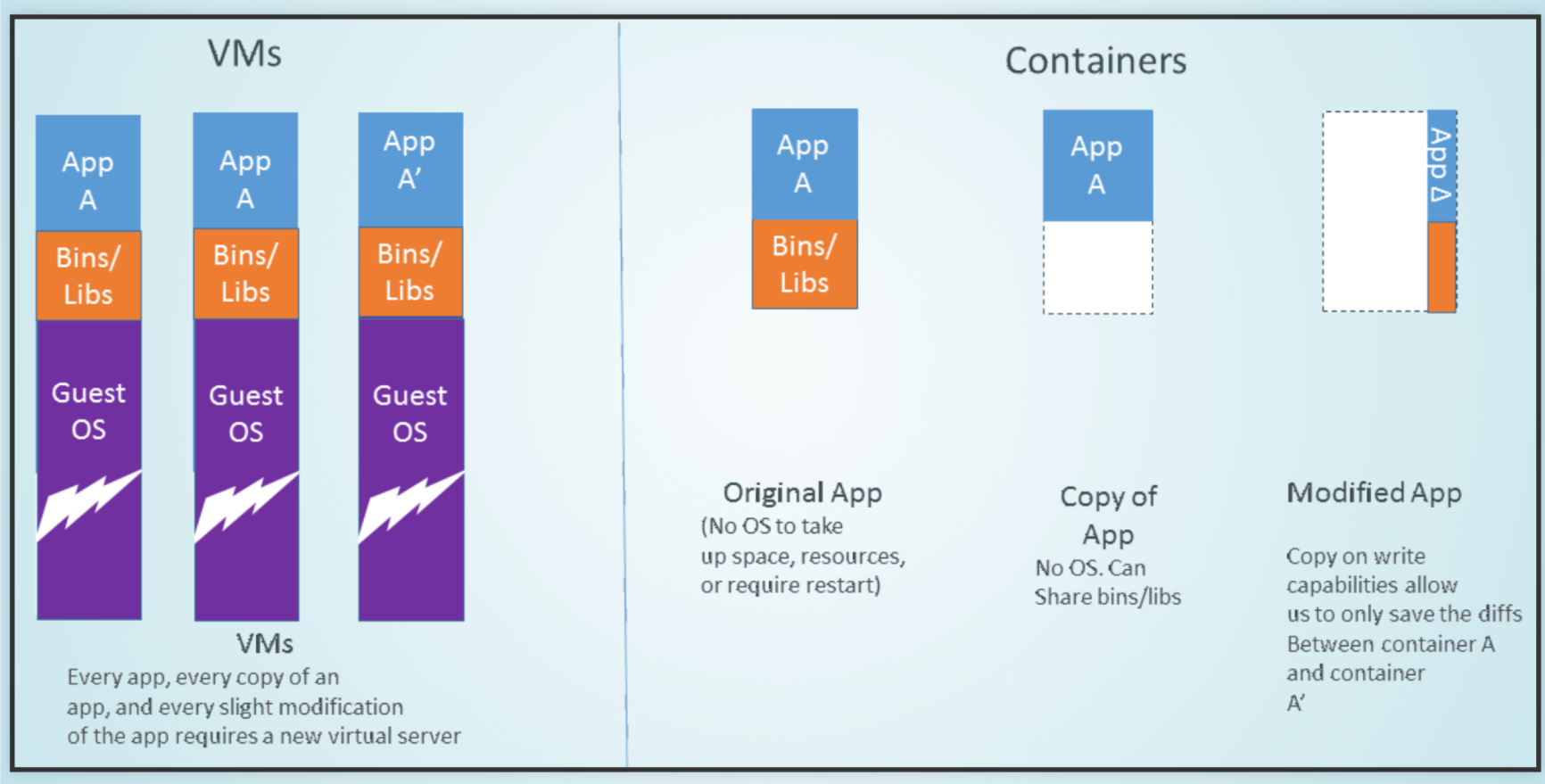
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
VMs vs Containers



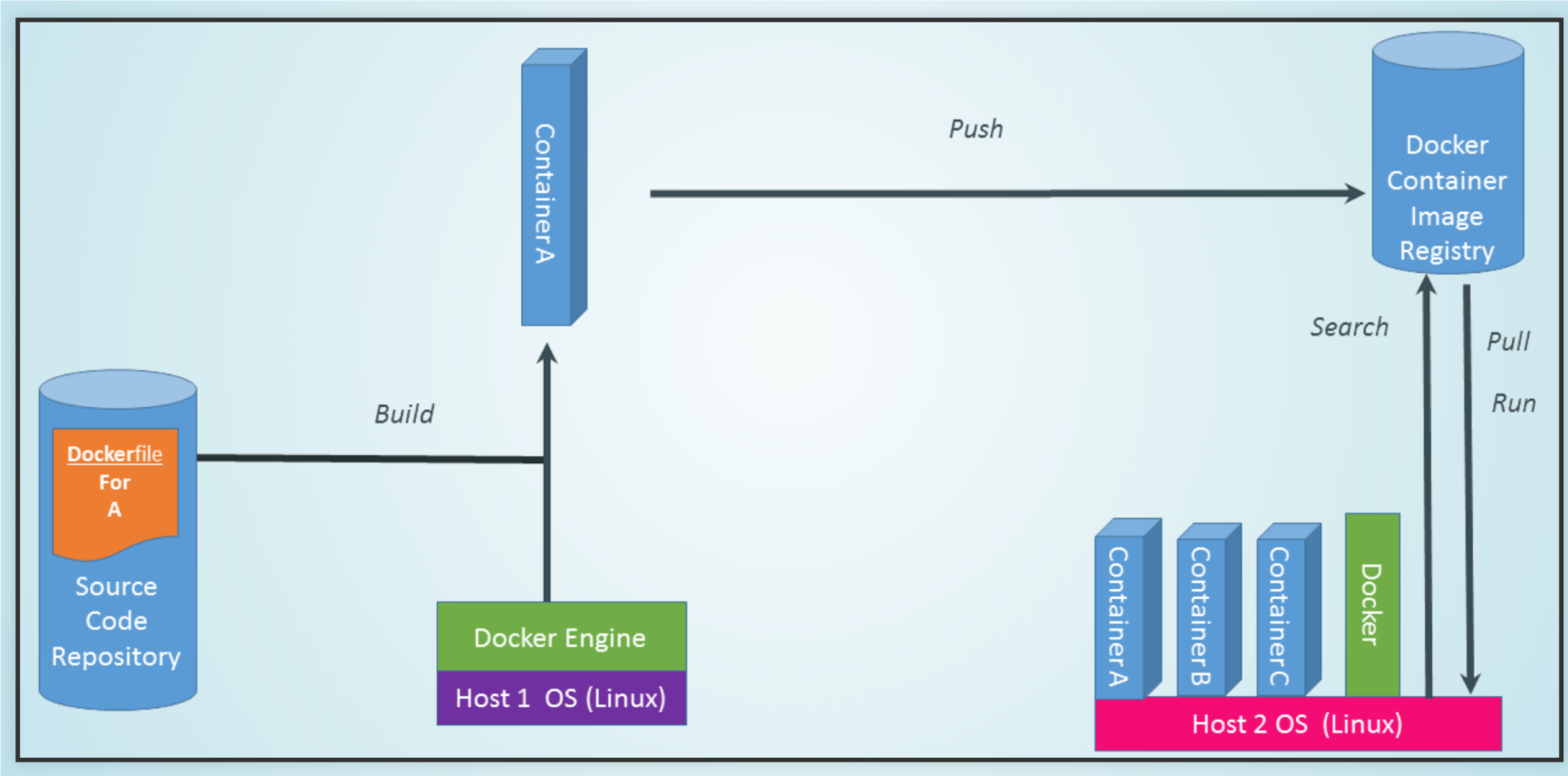
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Why are Docker Containers Lightweight?



Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
What are the Basics of a Docker System?



Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Install

Docker is available in two editions:

- Community Edition (CE)
- Enterprise Edition (EE)

Capabilities	Docker Engine - Community	Docker Engine - Enterprise	Docker Enterprise
Container engine and built in orchestration, networking, security	✓	✓	✓
Certified infrastructure, plugins and ISV containers		✓	✓
Image management			✓
Container app management			✓
Image security scanning			✓

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Supported Platforms

Supported platforms

Docker CE is available on multiple platforms. Use the following tables to choose the best installation path for you.

DESKTOP

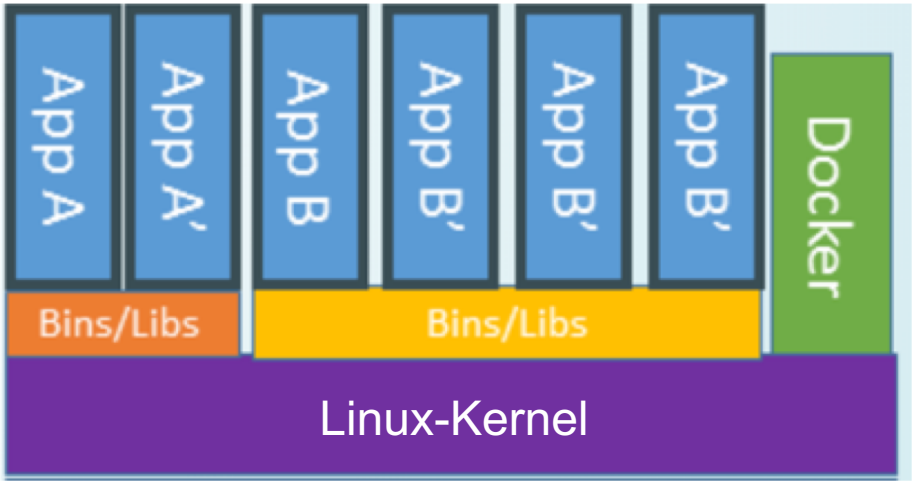
Platform	x86_64
Docker Desktop for Mac (macOS)	✓
Docker Desktop for Windows (Microsoft Windows 10)	✓

SERVER

Platform	x86_64 / amd64	ARM	ARM64 / AARCH64	IBM Power (ppc64le)	IBM Z (s390x)
CentOS	✓		✓		
Debian	✓	✓	✓		
Fedora	✓		✓		
Ubuntu	✓	✓	✓	✓	✓

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

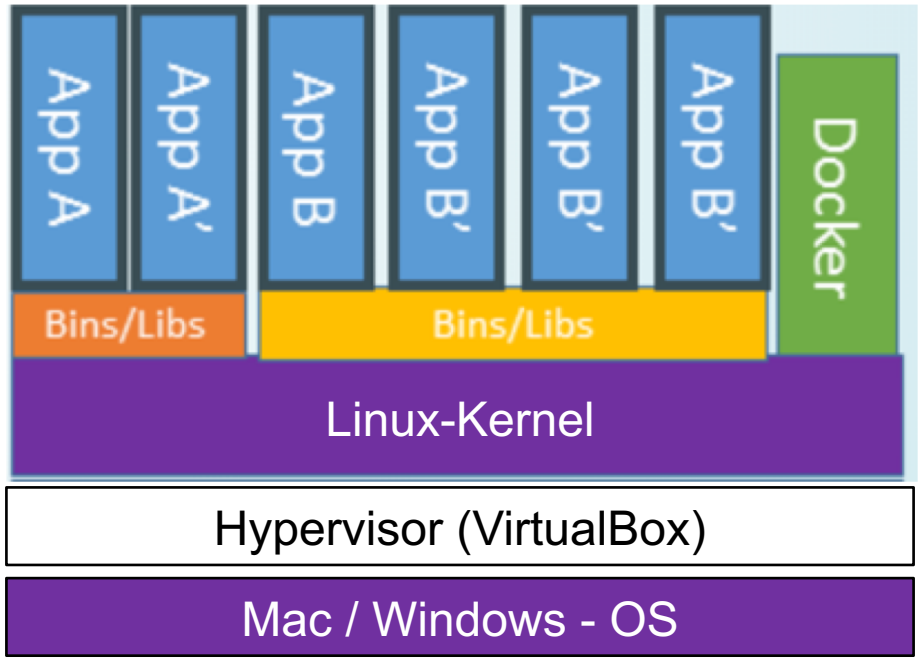
Docker
How can Docker run on Mac / Windows?



MacOs / Windows don't
have a Linux-Kernel

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
How can Docker run on Mac / Windows?

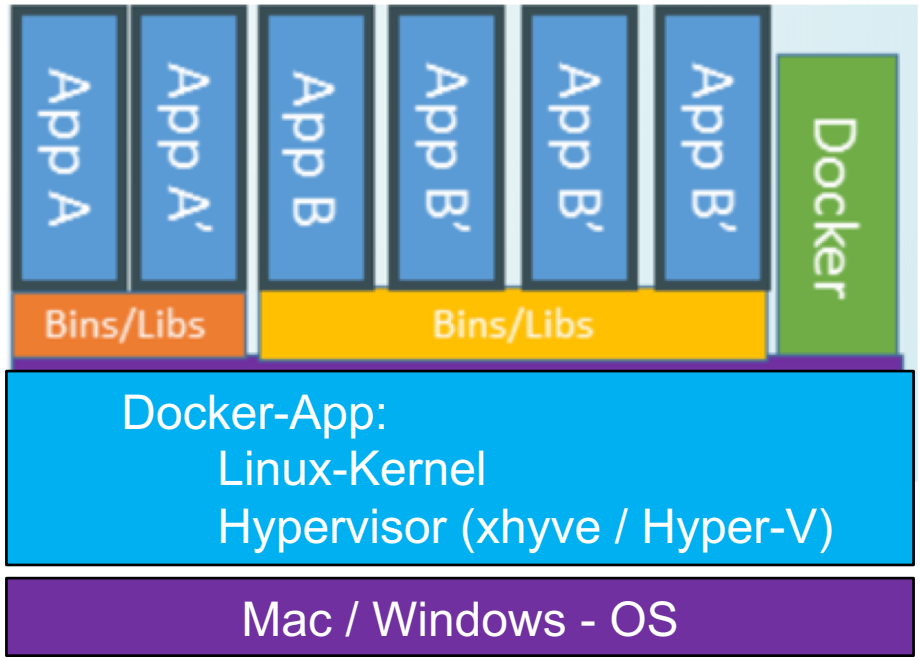


Up to June 2016:

VirtualBox on Mac / Windows + lightweight Linux (boot2docker)

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
How can Docker run on Mac / Windows?



Up to June 2016:

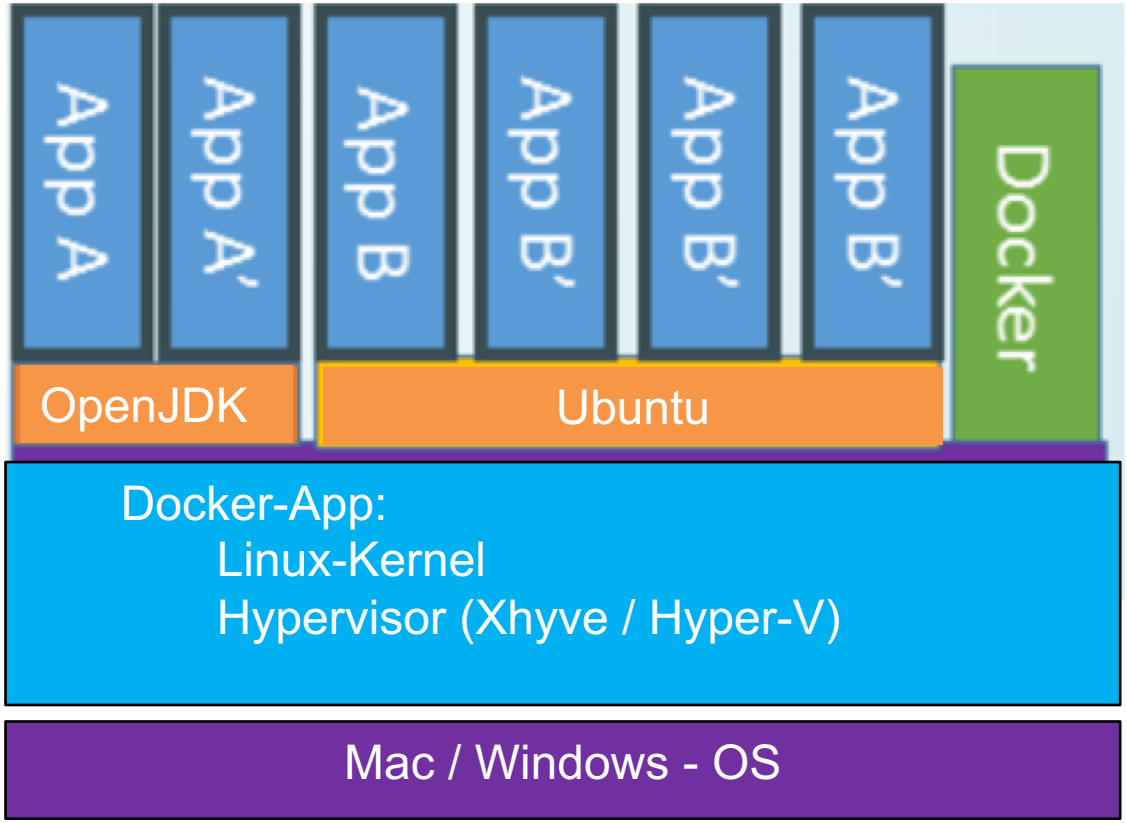
VirtualBox on Mac / Windows + lightweight Linux (boot2docker)

From June 2016:

Docker App with integrated Hypervisor (xhyve / Hyper-V)
and lightweight Linux

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Which are popular Bins/Libs?



Libs (Images):

- Ubuntu
- OpenJDK
- Maven
- Nodejs
- MySQL
- PhP
- ros
- amazonlinux
- …

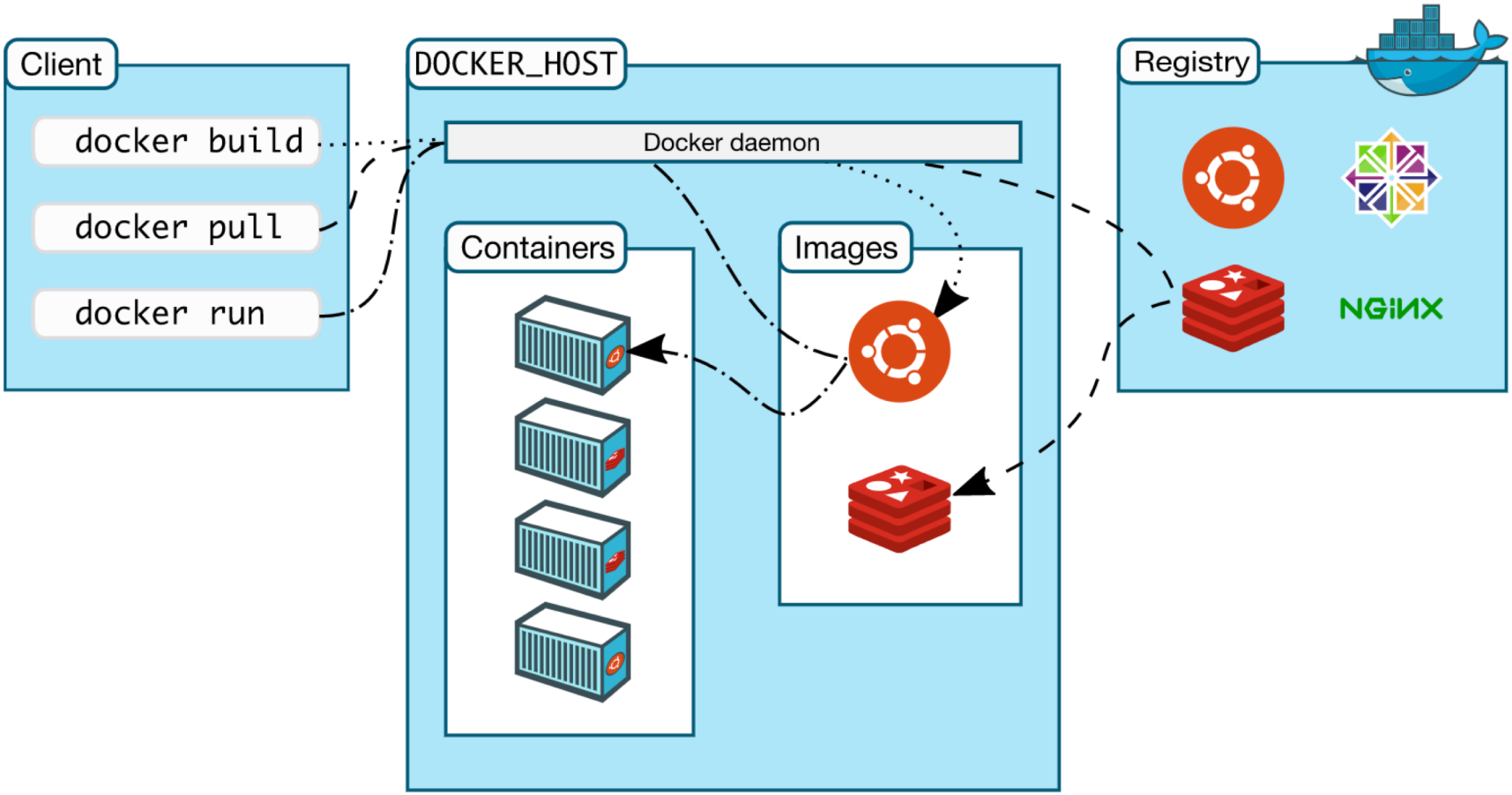
Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Start with Docker – Client & Server Engine

```
~ :\> docker version
Client: Docker Engine - Community
 Version:           18.09.2
 API version:       1.39
 Go version:        go1.10.8
 Git commit:        6247962
 Built:             Sun Feb 10 04:12:39 2019
 OS/Arch:           darwin/amd64
 Experimental:      false

Server: Docker Engine - Community
 Engine:
  Version:          18.09.2
  API version:      1.39 (minimum version 1.12)
  Go version:       go1.10.6
  Git commit:       6247962
  Built:            Sun Feb 10 04:13:06 2019
  OS/Arch:          linux/amd64
  Experimental:     false
~ :\>
```

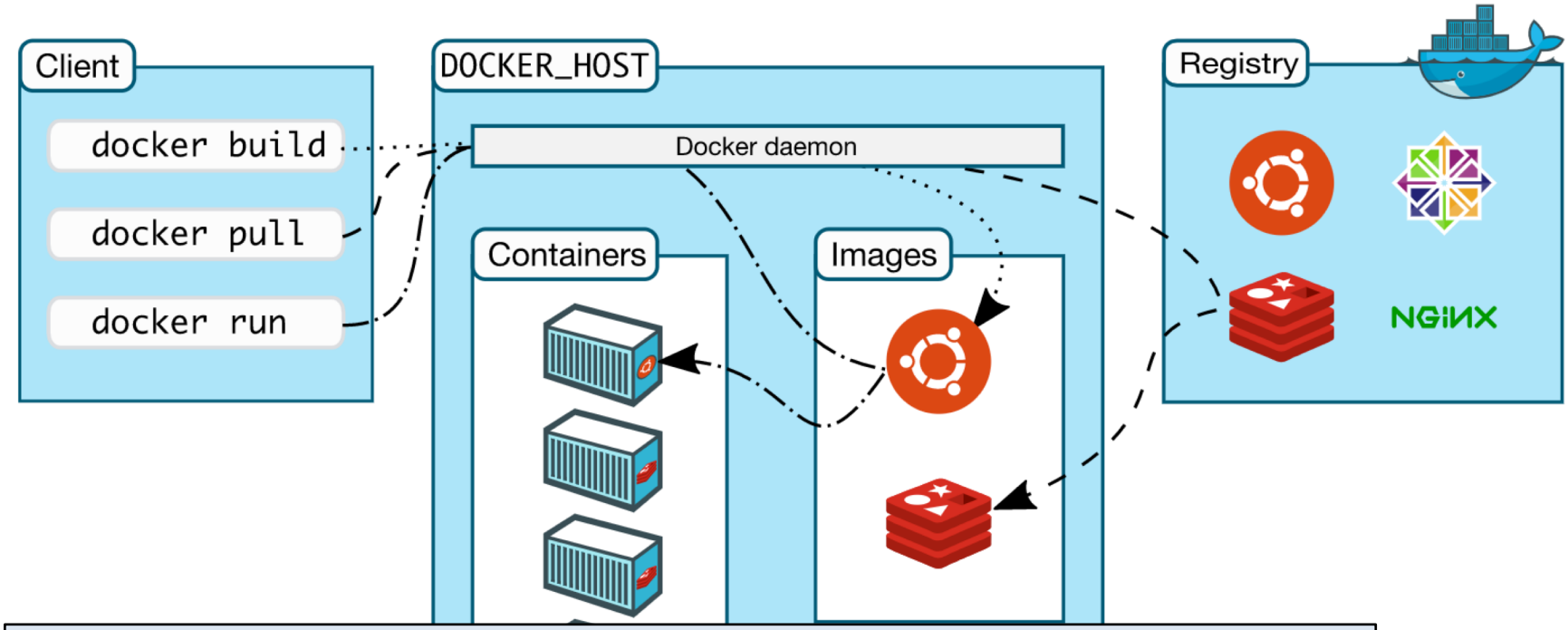

Source: https://pointful.github.io/docker-intro/#/2 http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Client-Server Architecture



Source: https://docs.docker.com/engine/docker-overview/#docker-architecture

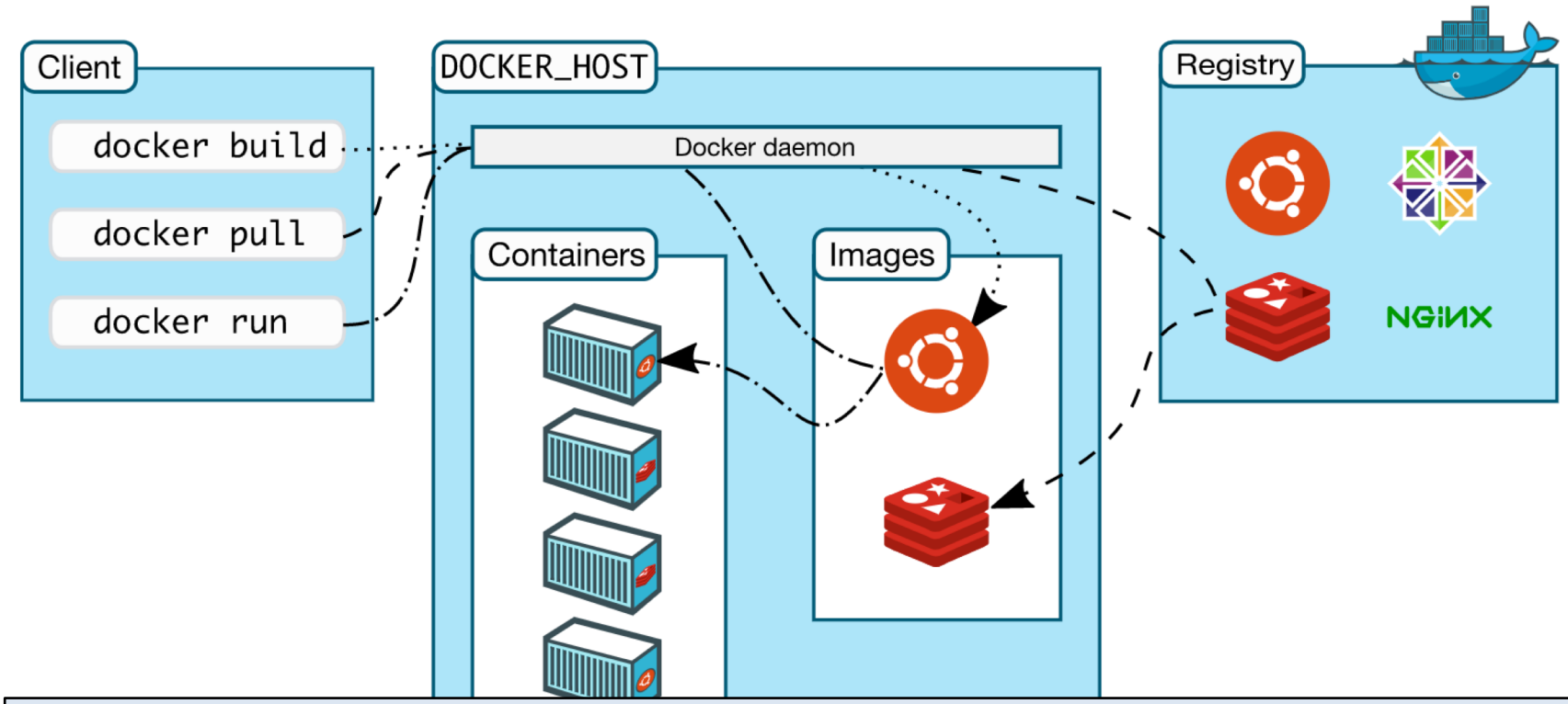
Docker
Client-Server Architecture



- The **Docker daemon** Receives and processes incoming Docker API requests.
- The **Docker client** Command line tool - the docker binary. Talks to the Docker daemon via the Docker API.
- **Docker Hub** Registry Public image registry. The Docker daemon talks to it via the registry API.

Source: https://docs.docker.com/engine/docker-overview/#docker-architecture

Docker Objects



Images: A read-only file used to execute code in a Docker container.

Container: A container is a runnable instance of an image. You can create, start, stop, move, or delete a container.

Docker
Getting started

1. Download an image

```
docker pull <image>
```

```
docker pull jpetazzo/clock
```

Show all installed images

```
docker images
```

2. Create container from image

```
docker create --name <container-name> <image-name>
```

```
docker create --name myFirstContainer jpetazzo/clock
```

Show all installed containers

```
docker ps -a
```

3. Start container

```
docker start <container-id/name>
```

```
docker start myFirstContainer
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Getting started

4. Attach stdin, stdout, stderr

```
docker attach <container-id/name>
```

```
docker attach myFirstContainer
```

5. Docker enter running container

```
docker exec -it [container-id] bash
```

```
docker exec -it myFirstContainer sh
```

6. Stop container

```
docker stop <container-id/name>
```

```
docker stop myFirstContainer
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker - Start with Docker Container

Download, create and start a container:

```
docker pull jpetazzo/clock
```

```
docker create --name myFirstContainer jpetazzo/clock
```

```
docker start myFirstContainer
```

Shortcut for pull, create, start:

```
docker run <image>
```

```
docker run jpetazzo/clock
```

- Pulls from hub only if the image is not available locally
- Starts in attached mode (not running in background: Ctrl+C stops process)
- Name of container is generated by random
- *docker run* always creates a new container

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker - Start with Docker Images

Search for an image

```
docker search <image>
```

```
docker search busybox
```

Download an image

```
docker pull <image>
```

```
docker pull busybox
```

Show all installed images

```
docker images
```

Remove images

```
docker rmi <image_name:version/image-id>
```

Remove all (images & containers)

```
docker system prune -a
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker - Start with Docker Containers

Run a container from an image

```
docker run <image>
```

```
docker run busybox
```

Run from image interactive with TTY-terminal

```
docker run -it busybox
```

Run from image with command

```
docker run busybox echo 'Hallo World!!!'
```

Show all containers

```
docker ps -a
```

Remove container

```
docker rm <container_id>
```

Remove all container

```
docker container prune
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker - Start with Docker Container

Run from image and define own container name

```
docker run --name <myContName> <image>
```

Run from image and remove container after exit

```
docker run --rm <image>
```

```
docker run --rm busybox
```

Run from image detachted mode (close terminal and keep container running)

```
docker run -d <image>
```

Run from image and publish all exposed ports to random ports

```
docker run -P <image>
```

Run from image and publish map port 7979 to 8080

```
docker run -p 7979:8080 <image>
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Often used commands

For interaction inside the container

```
docker run –it --rm --name <myContainerName> <image>
```

Run from image in interactive mode, give container own name and remove container after usage.

For running in server-mode

```
docker run –dP --rm --name <myContainerName> <image>
```

Run from image in detached mode (background), publish all exposed ports to random ports , remove the container after usage and give the container an own name.

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Generate own docker image

Two ways to generate own image

1. **Create an image from an existing container:** In this case, you start with an existing image, customize it with the changes you want, then build a new image from it.

2. **Use a Dockerfile:** In this case, you use a file of instructions — the Dockerfile — to specify the base image and the changes you want to make to it.

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Create an image from an existing container

Create and start container form "suitable" base image. Don't remove after stop. Map port, e.g 7979 to 8080

```
docker run -it –p 7979:8080
           --name=mywebapp01 openjdk:8-jre-alpine
```

Inside the container: download the executable jar (fat-jar).

```
// form internet: wget -O myApp.jar http://...

docker cp Thorntail-1.0-SNAPSHOT-thorntail.jar
mywebapp01:/myApp.jar

java -jar -Djava.net.preferIPv4Stack=true
          -Djava.net.preferIPv4Addresses=true
          myApp.jar
```

(Force java to use IPv4 inside docker form Thorntail, see: here)

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Create an image from an existing container (2/2)

The container is working. Build an image form it.

```
docker commit -m "Message" -a "Author Name"
[containername] [imagename]
```

```
docker commit -m "My first webapp" -a "Grabowski"
mywebapp01 grabow/mywebimage:v1
```

Login to Docker Hub.

```
docker login
```

Upload the image to Docker Hub.

```
docker push grabow/mywebimage:v1
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Use a Dockerfile (1/2)

The Dockerfile (**`Dockerfile`**) is essentially the build instructions to build the image.

```
FROM openjdk:8-jre-alpine

RUN mkdir webapp

COPY Thorntail-1.0-SNAPSHOT-thorntail.jar /webapp/myApp.jar

EXPOSE 8080
```

Dockerfile

Use **`docker build`** to generate an Docker-Image from Dockerfile.

```
docker build -t <image_name:version> <dir_of_Dockerfile>
```

```
docker build -t grabow/mynextapp:v1 .
```

Upload the image to Docker Hub.

```
docker push grabow/mynextapp:v1
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Use a Dockerfile (2/2)

```
FROM openjdk:8-jre-alpine

RUN mkdir webapp

COPY Thorntail-1.0-SNAPSHOT-thorntail.jar /webapp/myApp.jar

EXPOSE 8080

WORKDIR /webapp

CMD ["java", "-jar", "-Djava.net.preferIPv4Stack=true",
             "-Djava.net.preferIPv4Addresses=true",
             "myApp.jar"]
```

WORKDIR: Create and change to folder

CMD: Execute the following command as default, if container is started

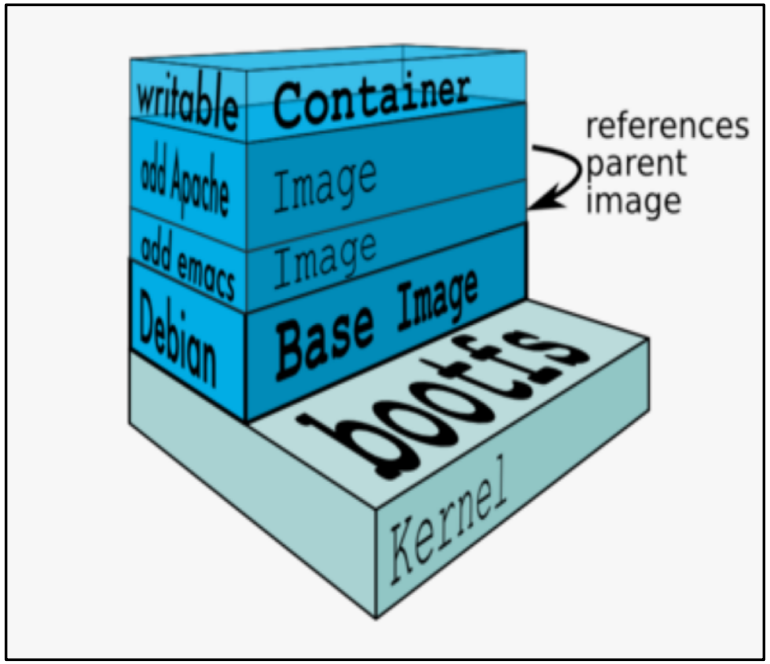
Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Optimize Dockerfiles – Layers

```
FROM openjdk:8-jre-alpine

RUN mkdir webapp

COPY minitext.txt /webapp/.

COPY Thorntail-1.0-SNAPSHOT-
thorntail.jar
/webapp/myApp.jar

EXPOSE 8080

WORKDIR /webapp

RUN apk --no-cache add curl

CMD ["java", "-jar", "-Djava.
net.preferIPv4Stack=true","-
Djava.net.preferIPv4
Addresses=true", "myApp.jar"]
```

The docker image is organized in read-only layers.

Each command generates a new layer.



Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Optimize Dockerfiles – Layers

```
FROM openjdk:8-jre-alpine

RUN mkdir webapp

COPY minitext.txt /webapp/.

COPY Thorntail-1.0-SNAPSHOT-
thorntail.jar
/webapp/myApp.jar

EXPOSE 8080

WORKDIR /webapp

RUN apk --no-cache add curl

CMD ["java", "-jar", "-Djava.
net.preferIPv4Stack=true","-
Djava.net.preferIPv4
Addresses=true", "myApp.jar"]
```

The order matters

Layers are cached: If one of the files have different contents than on the previous run, the layer cache is invalidated,
and **all** subsequent commands will be executed.

E.g: If this file has changed, all subsequent commands are executed.

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Optimize Dockerfiles – Layers

```
FROM openjdk:8-jre-alpine

RUN apk --no-cache add curl

RUN mkdir webapp

EXPOSE 8080

WORKDIR /webapp

COPY Thorntail-1.0-SNAPSHOT-
thorntail.jar
/webapp/myApp.jar

COPY minitext.txt /webapp/.

CMD ["java", "-jar", "-Djava.
net.preferIPv4Stack=true","-
Djava.net.preferIPv4
Addresses=true", "myApp.jar"]
```

The order matters

Layers are cached: If one of the files have different contents than on the previous run, the layer cache is invalidated,
and **all** subsequent commands will be executed.

Re-order to optimze performance

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Optimize Dockerfiles – Layers

```
FROM openjdk:8-jre-alpine

RUN apk --no-cache add curl

RUN mkdir webapp

EXPOSE 8080

WORKDIR /webapp

COPY large.zip .

RUN unzip large.zip

RUN rm large.zip
```

The number of layers matter

Each layer is stored in the image.

Unused layer blow the image.

- Unused layer:
  The zipped file is not used in the finale image

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Optimize Dockerfiles – Layers

```
FROM openjdk:8-jre-alpine

RUN apk --no-cache add curl

RUN mkdir webapp

EXPOSE 8080

WORKDIR /webapp

RUN curl http://xyz/large.zip
   -O /webapp \
   && unzip /webapp/large.zip
         -d/webapp \
   && rm /webapp/large.zip
```

The number of layers matter

Each layer is stored in the image.

Unused layer blow the image.

- Merge the command with && operator into one single command

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Dockerfiles for Development

Change into project dir (project must be inside docker context)

```
cd /Users/wiggel/NetBeansProjects/EA1/Thorntail
```

Generate Dockerfile

```
FROM maven:3.5.4-jdk-9
WORKDIR /webappdev
COPY pom.xml .
RUN mvn package -DskipTests
COPY src src
RUN mvn package -DskipTests
RUN echo "done!"
```

Dockerfile

Build Image

```
docker build -t grabow/mynextapp:v2 .
```

Run image in container

```
docker run -it -p 8080:8080 grabow/mynextapp:v2 bash
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Dockerfiles for Development

Change into project dir (project must be inside docker context)

```
cd /Users/wiggel/NetBeansProjects/EA1/Thorntail
```

Generate Dockerfile

```
FROM maven:3.5.4-jdk-9
WORKDIR /webappdev
COPY pom.xml .
RUN mvn package -DskipTests
COPY src src
RUN mvn package -DskipTests
RUN echo "done!"
```

Dockerfile

Build image

```
docker build -t grabow/mynextapp:v2 .
```

Run image in container

```
docker run -it -p 8080:8080 grabow/mynextapp:v2 bash
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Dockerfiles for Development

Generate Dockerfile with default start of web-server

```
FROM maven:3.5.4-jdk-9
WORKDIR /webappdev
COPY pom.xml .
RUN mvn package -DskipTests
COPY src src
RUN mvn package -DskipTests
RUN echo "done!"
CMD ["java", "-jar", "-Djava.net.preferIPv4Stack=true",
             "-Djava.net.preferIPv4Addresses=true",
             "myApp.jar"]
```

Dockerfile

Build image

```
docker build -t grabow/mynextapp:v3 .
```

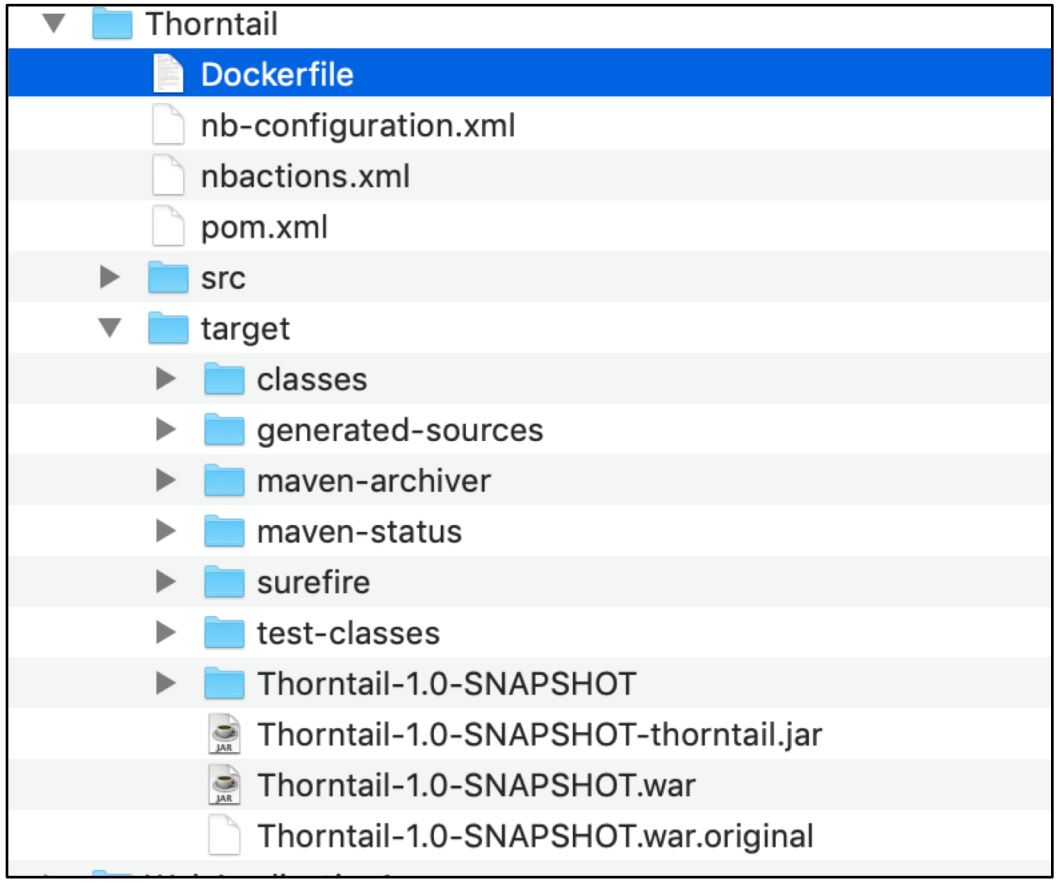
Run image in container

```
docker run -it -p 8080:8080 grabow/mynextapp:v3
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Docker Context

A build's context is the set of files located in the specified PATH or URL.



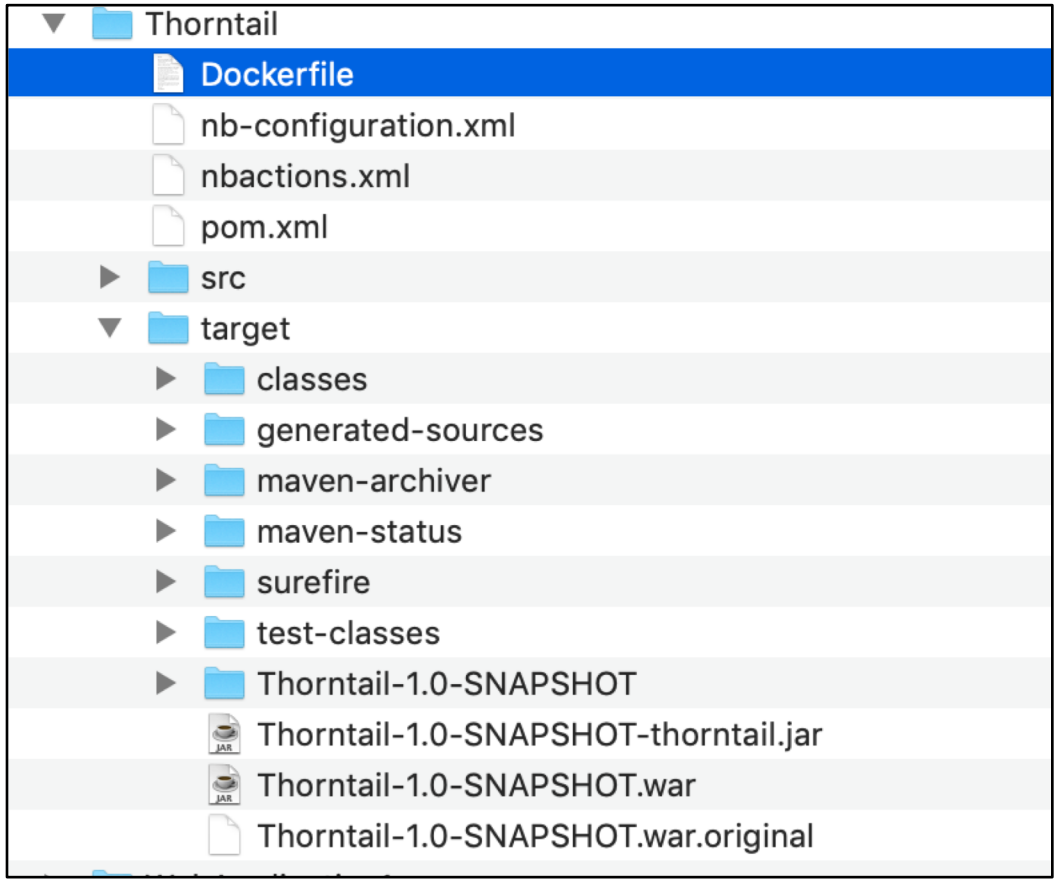
If we build from inside the folder, where Dockerfile is located, all files and (sub-)folders are part of docker context.

The Docker client packs all **build context** files into tar archive and uploads this archive to the Docker server.

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Docker Context

.dockerignore-file prevents from uploading unnecessary files.

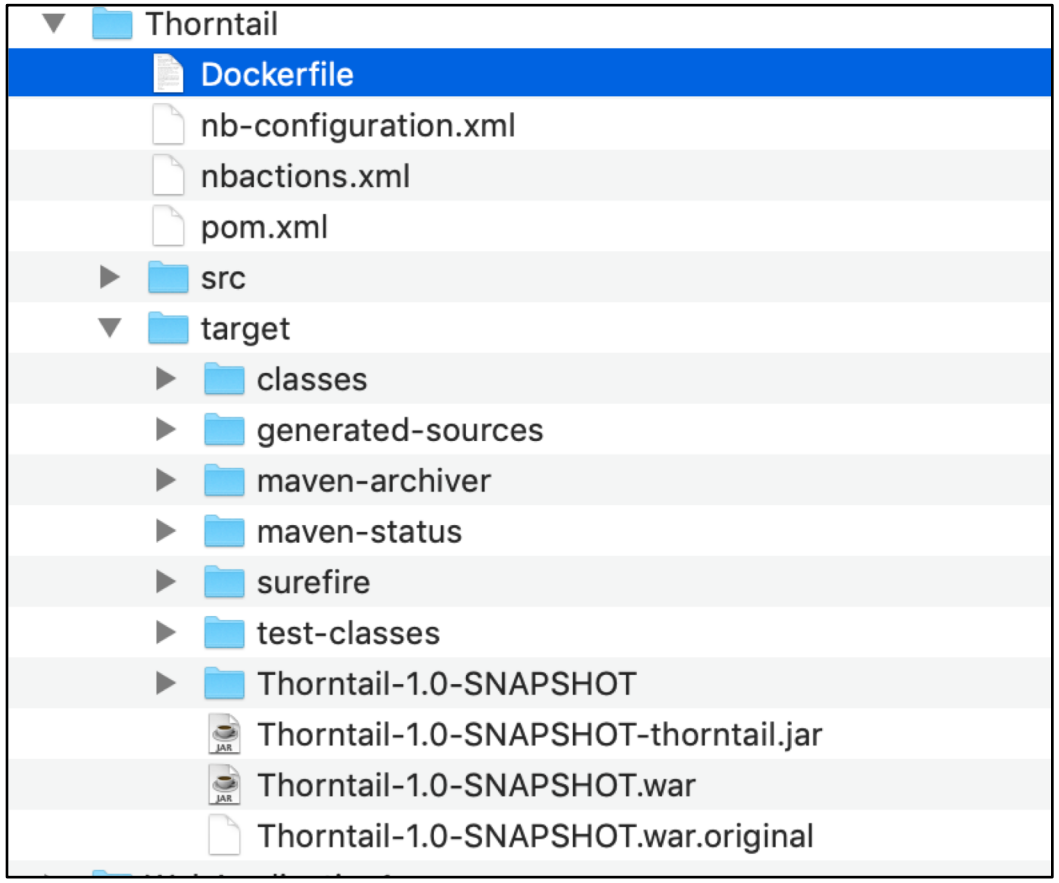


The .dockerignore file is the tool, that can help you to define the Docker **build context** you really need. Using this file, you can specify **ignore rules** and **exceptions** fro m these rules for files and folder, that won't be included in the **build context** and thus won't be packed into an archive and uploaded to the Docker server.

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Docker Context

.dockerignore-file prevents from uploading unnecessary files.



.dockerignore for Java

```
.git
/target
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Docker Context

Can't access files outside docker-context

```
FROM maven:3.5.4-jdk-9

WORKDIR /webappdev

COPY /Users/wiggel/NetBeansProjects/EA1/Thorntail/pom.xml .

RUN mvn package -DskipTests

COPY /Users/wiggel/NetBeansProjects/EA1/Thorntail/src src

RUN mvn package -DskipTests

RUN echo "done!"
```

Dockerfile

Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Multi-Staging – Two different FROM in one Dockerfile

```
# Multistage Docker build.
# 1st stage, build the app
FROM maven:3.5.4-jdk-9 as build
WORKDIR /helidon
COPY pom.xml .
RUN mvn package -DskipTests
ADD src src
RUN mvn package -DskipTests
RUN echo "done!"

# 2nd stage, build the runtime image
FROM openjdk:8-jre-slim
WORKDIR /helidon
# Copy the binary built in the 1st stage
COPY --from=build /helidon/target/mywebapp.jar ./
COPY --from=build /helidon/target/libs ./libs
CMD ["java", "-jar", "mywebapp.jar"]
```

For development

```
docker build --target build -t grabow/myweb:v1 .
```


Source: https://helidon.io/docs/latest/#/guides/04_dockerfile

Docker
Multi-Staging – Two different FROM in one Dockerfile

```
# Multistage Docker build.
# 1st stage, build the app
FROM maven:3.5.4-jdk-9 as build
WORKDIR /helidon
COPY pom.xml .
RUN mvn package -DskipTests
ADD src src
RUN mvn package -DskipTests
RUN echo "done!"

# 2nd stage, build the runtime image
FROM openjdk:8-jre-slim
WORKDIR /helidon
# Copy the binary built in the 1st stage
COPY --from=build /helidon/target/mywebapp.jar ./
COPY --from=build /helidon/target/libs ./libs
CMD ["java", "-jar", "mywebapp.jar"]
```

For production

```
docker build -t grabow/myweb:v1 .
```


Source: https://helidon.io/docs/latest/#/guides/04_dockerfile

Docker
Appendix

Docker
Container

Create container from image

```
docker create --name <container-name> <image-name>
```

```
docker create --name myFirstContainer busybox
```

Docker start container

```
docker start <container-id/name>
```

Docker stop container

```
docker stop <container-id/name>
```

Docker enter running container

```
docker exec -it [container-id] bash
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Thorntail & Docker

From: https://docs.thorntail.io/2.4.0.Final/#setting-system-properties-using-the-command-line_thorntail

Add the following config to ensure IPv4 for ThornTail in Docker!

```
...
<version>2.4.0.Final</version>
...
<configuration>
  <properties>
       <java.net.preferIPv4Stack>
              true
       </java.net.preferIPv4Stack>
 </properties>
</configuration>
...
```


Source: http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf

Docker
Links

- Docker tutorial simple: https://docker-curriculum.com/#docker-run
- Simple deplyoment https://stackify.com/guide-docker-java/
- Create Image (2 ways): https://www.mirantis.com/blog/how-do-i-create-a-new-docker-image-for-my-application/
- Docker tutorial (2014): http://wiki.zenoss.org/download/core/drich_slides/DockerSlides.pdf