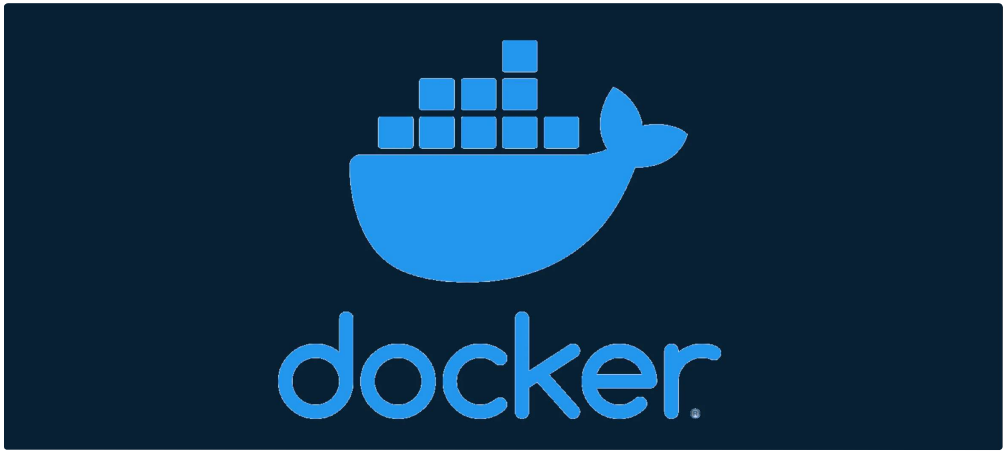
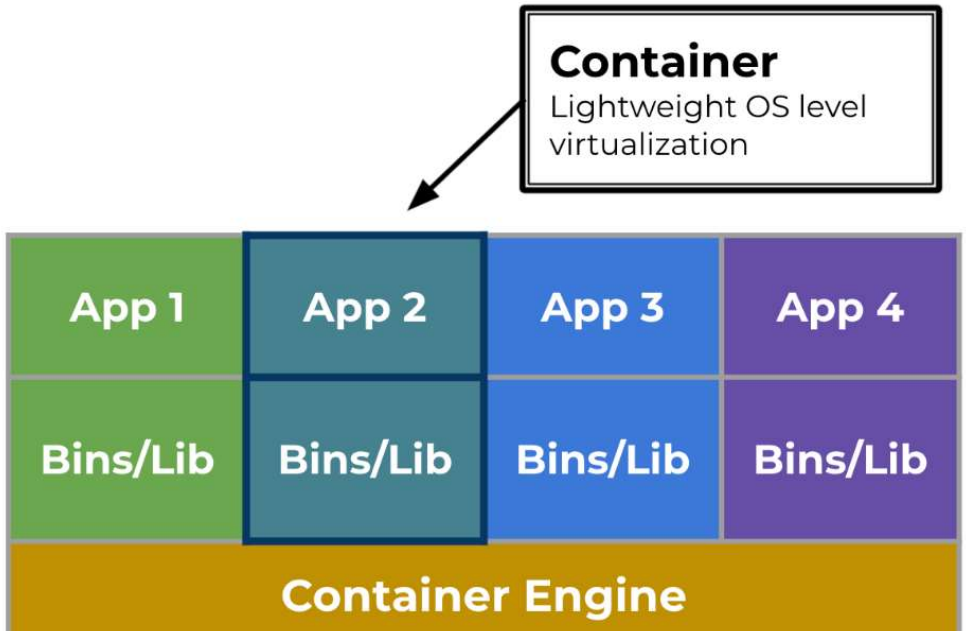
Share

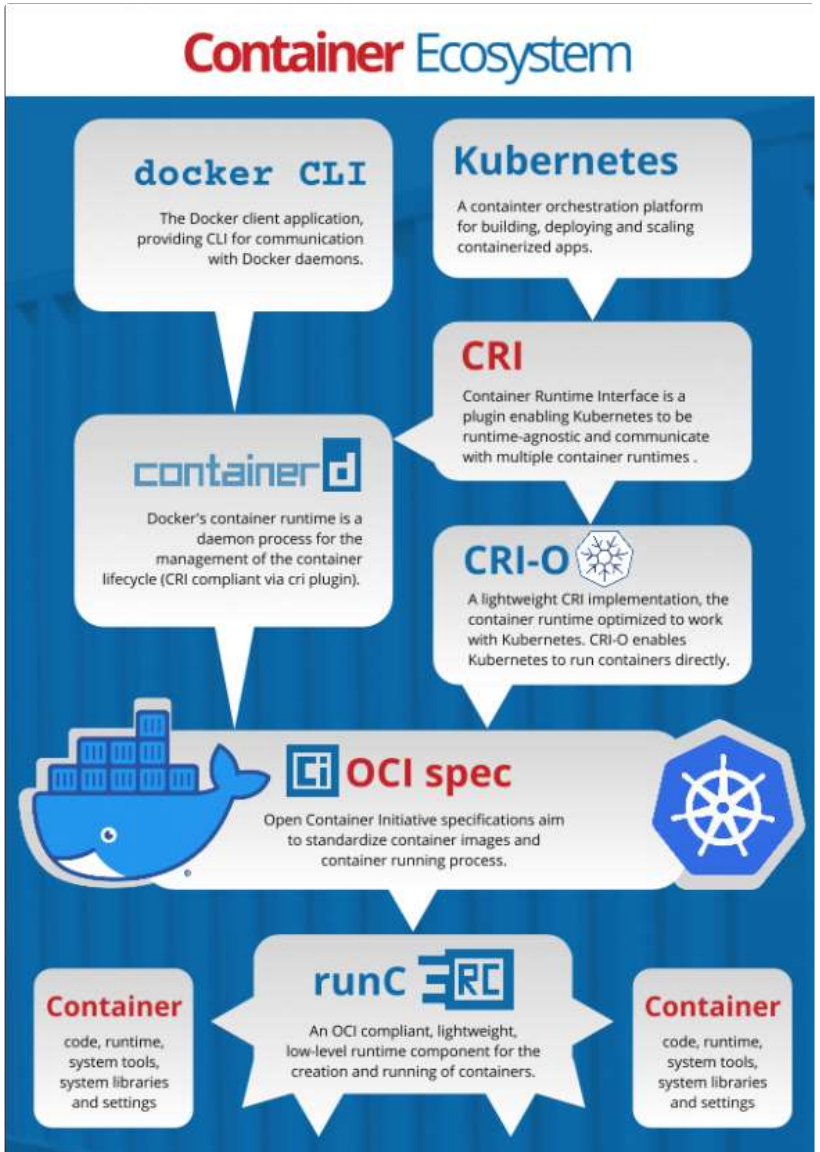
2- WHAT IS CONTAINER

Sezgin KEŞCİOĞLU tarafından oluşturuldu
Son güncelleme: Kas 23, 2022

WHAT IS CONTAINER







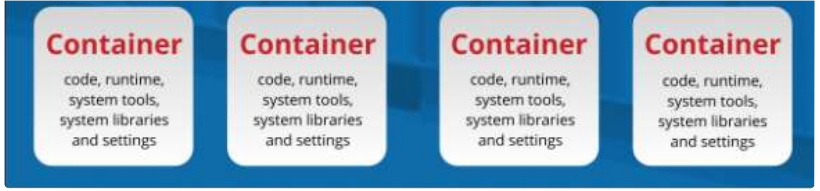
Container Runtime Interface (CRI)

CRI is the protocol that Kubernetes uses to control the different runtimes that create and manage containers.

CRI is an abstraction for any kind of container runtime you might want to use. So CRI makes it easier for Kubernetes to use different container runtimes.

Instead of the Kubernetes project needing to manually add support for each runtime, the CRI API describes how Kubernetes interacts with each runtime. So then, it's down to the runtime to actually manage containers. As long as it obeys the CRI API, it can do whatever it

CONTAINER



likes.

Open Container Initiative (OCI)

The OCI is a group of tech companies who maintain a specification for the container image format, and how containers should be run.

The idea behind the OCI is that you can choose between different runtimes which conform to the spec. Each of these runtimes have different lower-level implementations.

containerd

containerd is a high-level container runtime that came from Docker, and implements the CRI spec. It pulls images from registries, manages them and then hands over to a lower-level runtime, which actually creates and runs the container processes.

containerd was separated out of the Docker project, to make Docker more modular.

So Docker uses *containerd* internally itself. When you install Docker, it will also install *containerd*.

containerd implements the Kubernetes **Container Runtime Interface** (CRI), via its *cri* plugin.

CRI-O

CRI-O is another high-level container runtime which implements the Container Runtime Interface (CRI). It's an alternative to *containerd*. It pulls container images from registries, manages them on disk, and launches a lower-level runtime to run container processes.

It was born out of Red Hat, IBM, Intel, SUSE and others. It was specifically created from the ground up as a container runtime for Kubernetes. It provides the ability to start, stop and restart containers, just like *containerd*.

DEFINITION OF CONTAINER - Dockerfile

```
### WARNING: This file is AUTOGENERATED. See containerfile.template to make changes for GitHub Pull Requests#
# RHEL Universal Base Image (RHEL UBI) is a stripped down, OCI-compliant,
# base operating system image purpose built for containers. For more information
```

```
# see https://developers.redhat.com/products/rhel/ubi
#
FROM registry.access.redhat.com/ubi8/ubi-minimal
MAINTAINER CrowdStrike Solutions Architects <integrations@crowdstrike.com>
USER root

ARG VERSION

# VCS_REF=$(git rev-parse --short HEAD)
ARG VCS_REF
ARG FALCON_PKG

#
# Friendly reminder that generated container images are from an open source
# project, and not a formal CrowdStrike product.
#
### Required OpenShift Labels
LABEL name="CrowdStrike Falcon Sensor" \
      maintainer="integrations@crowdstrike.com" \
      vendor="CrowdStrike Community" \
      version=$VERSION \
      release=$VCS_REF \
      summary="CrowdStrike Falcon Sensor" \
      description="The falcon-sensor package provides the Crowdstrike Falcon Sensor daemon and kernel modules."

### add licenses to this directory
COPY licenses /licenses

### Copy falcon-sensor.rpm into container
COPY $FALCON_PKG falcon-sensor.rpm

RUN REPOLIST="ubi-8-baseos" \
    INSTALL_PKGS="libnl3 net-tools zip openssl hostname iproute procps" && \
    microdnf -y update --disablerepo "*" --enablerepo ubi-8-baseos --setopt=tsflags=nodocs && \
    microdnf -y install --disablerepo "*" --enablerepo ${REPOLIST} --setopt=tsflags=nodocs ${INSTALL_PKGS} && \
    rpm -ivh falcon-sensor.rpm && \
    microdnf clean all && rm -rf /var/cache/yum && \
    rm -f falcon-sensor.rpm

#
# Copy the entrypoint script into the container and make sure
# that its executable. Add the symlink for backwards compatability
#
COPY entrypoint.sh /

ENV PATH ".:/bin:/usr/bin:/sbin:/usr/sbin"
WORKDIR /opt/CrowdStrike

VOLUME /var/log
ENTRYPOINT ["/entrypoint.sh"]
```

RHEL Universal Base Image (RHEL UBI)

File: Dockerfile.ubi-minimal

CONTAINER LAB SESSION

Genişletmek için buraya tıklayın...

Lets create a container image.

Inftrastructure: Windows 10 Pro (Base OS) , Wifi Network (192.168.1.0/24, 192.168.1.1 Gateway/Firewall, External Network (WLAN): Vodafone Net)

Our Host OS is Ubuntu Server 22 (VM/HyperV) , Bridged Network (192.168.1.0/24)

System Libraries: Debian GNU/Linux 11 (bullseye) Kernel

Container Engine: Docker version 20.10.18, build b40c2f6

Container Network (192.168.1.0/24) : 8888 (Uses vm host's bridge network to outside traffic)

Runtime : OpenJDK 64-Bit Server VM 18.9 (build 11.0.16+8, mixed mode, sharing)

App: CONFIG-SERVER-0.0.1-SNAPSHOT.jar

DOCKER FILE FOR CONTAINE IMAGE CREATION

```
#Base image tag
FROM openjdk:11

#Copy your source file into container
ADD build/libs/CONFIG-SERVER-0.0.1-SNAPSHOT.jar CONFIG-SERVER-0.0.1-SNAPSHOT.jar


#Command to run when container starts up
ENTRYPOINT ["java","-jar","/CONFIG-SERVER-0.0.1-SNAPSHOT.jar"]


#Port exposing documentation for generated image
EXPOSE 8888
```

CONTAINER IMAGE BUILDER SHELL SCRIPT

```
echo "-------------------------------------------------------------------------------"
echo "JDK11 Installation"
echo "-------------------------------------------------------------------------------"
sudo apt install openjdk-11-jre-headless
echo "JDK Installation completed. OK"


echo "-------------------------------------------------------------------------------"
echo "Gradle Build Started For JAR Generation"
echo "-------------------------------------------------------------------------------"
chmod +x ./gradlew
./gradlew build
echo "Build finished. OK"


```

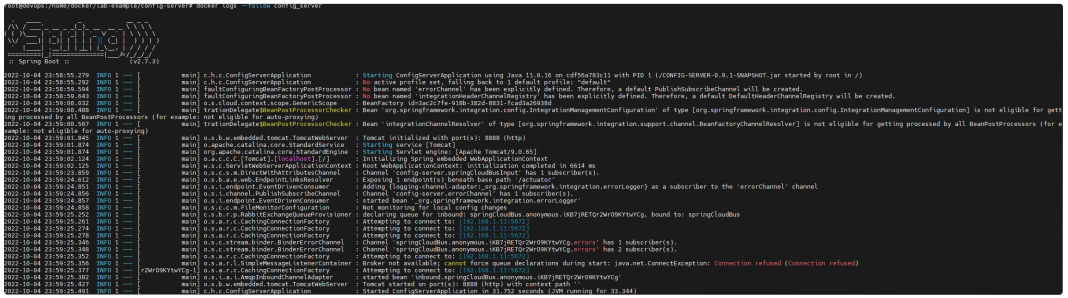
```
-rw-r--r--  1 root root 10147 Oct  4 23:39 README.md
-rw-r--r--  1 root root  1072 Oct  4 23:39 HELP.md
-rw-r--r--  1 root root  2747 Oct  4 23:39 gradlew.bat
-rwxr-xr-x  1 root root  8188 Oct  4 23:39 gradlew
drwxr-xr-x  3 root root  4096 Oct  4 23:39 gradle
-rw-r--r--  1 root root   321 Oct  4 23:39 Dockerfile
-rw-r--r--  1 root root   999 Oct  4 23:39 build.gradle
drwxr-xr-x  9 root root  4096 Oct  4 23:39 build
-rw-r--r--  1 root root  2365 Oct  4 23:58 build.sh
root@devops:/home/docker/lab-example/config-server#
```

Project Folder Review

```
root@devops:/home/docker/lab-example/config-server# sh build.sh
------------------------------------------------------------------------------------------------------------------------
JDK11 Installation
------------------------------------------------------------------------------------------------------------------------
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
openjdk-11-jre-headless is already the newest version (11.0.16+8-0ubuntu1~22.04).
0 upgraded, 0 newly installed, 0 to remove and 53 not upgraded.
JDK Installation completed. OK
------------------------------------------------------------------------------------------------------------------------
Gradle Build Started For JAR Generation
------------------------------------------------------------------------------------------------------------------------

BUILD SUCCESSFUL in 3s
7 actionable tasks: 7 up-to-date
Build finished. OK
------------------------------------------------------------------------------------------------------------------------
Starting Docker Image Building From Dockerfile
------------------------------------------------------------------------------------------------------------------------
Sending build context to Docker daemon  52.94MB
Step 1/4 : FROM openjdk:11
 ---> 47a932d998b7
Step 2/4 : ADD build/libs/CONFIG-SERVER-0.0.1-SNAPSHOT.jar CONFIG-SERVER-0.0.1-SNAPSHOT.jar
 ---> 02c7444b6bac
Step 3/4 : ENTRYPOINT ["java","-jar","/CONFIG-SERVER-0.0.1-SNAPSHOT.jar"]
 ---> Running in 4b05294e20b9
Removing intermediate container 4b05294e20b9
 ---> 39f32096ffea
Step 4/4 : EXPOSE 8888
 ---> Running in 04b9f70ebffd
Removing intermediate container 04b9f70ebffd
 ---> eebcb7bb3d1f
Successfully built eebcb7bb3d1f
Successfully tagged config_server:latest
------------------------------------------------------------------------------------------------------------------------
Checking the port is reserved on this host
------------------------------------------------------------------------------------------------------------------------
Port is not reserved. OK
------------------------------------------------------------------------------------------------------------------------
Starting image:config_server:latest Internal Port:8888 External Port:8888 Container name:config_server
------------------------------------------------------------------------------------------------------------------------
cdf56a783c11400959ffee144bc0144e93e7af840e786a167954d9395281ecde
Image started. OK
::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
Check logs by cmd => docker logs --follow config_server
Connect to container => docker exec -it config_server
Stop container => docker stop config_server
Remove Image => docker image rm -f config_server:latest
------------------------------------------------------------------------------------------------------------------------
root@devops:/home/docker/lab-example/config-server# docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED          STATUS          PORTS                                                 NAMES
cdf56a783c11   config_server:latest   "java -jar /CONFIG-S…"   25 seconds ago   Up 24 seconds   0.0.0.0:8888->8888/tcp, :::8888->8888/tcp   config_server
```

Build & Run Container



Check Container Logs

```
echo "------------------------------------------------------------------------------------------------------------------------"
echo "Starting Docker Image Building From Dockerfile"
echo "------------------------------------------------------------------------------------------------------------------------"
docker build --progress=plain -t config_server .

docker stop config_server


echo "------------------------------------------------------------------------------------------------------------------------"
echo "Checking the port is reserved on this host"
echo "------------------------------------------------------------------------------------------------------------------------"
if lsof -Pi :8888 -sTCP:LISTEN -t >/dev/null ; then
    echo "ERROR: The port 8888 is used in this host. Change the port number."
else
    echo "Port is not reserved. OK"
    echo "------------------------------------------------------------------------------------------------------------------------"
    echo "Starting image:config_server:latest Internal Port:8888 External Port:8888 Cont
    echo "------------------------------------------------------------------------------------------------------------------------"
    docker run -d --name config_server config_server:latest -p 8888:8888
    echo "Image started. OK"
    echo "::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::
    echo "Check logs by cmd => docker logs --follow config_server"
    echo "Connect to container => docker exec -it config_server"
    echo "Stop container => docker stop config_server"
    echo "Remove Image => docker image rm -f config_server:latest"
    echo "------------------------------------------------------------------------------------------------------------------------"
fi


```

Docker cheat sheet also can be view from this file.

Docker Cheat Sheet.txt

What is Docker?

Technically Speaking: New container tools for Red Hat Enterprise Linux

+ Etiket ekle

İlgili sayfalar

RED HAT | OPENSHIFT 4
HYPER-EDU
Birlikte düzenlendi

10- OPENSHIFT BUILDCONFIG CONCEPT
HYPER-EDU
Birlikte düzenlendi

12- OPENSHIFT DEPLOYMENTCONFIG CONCEPT
HYPER-EDU
Birlikte düzenlendi

Tepki ekleyen ilk siz olun