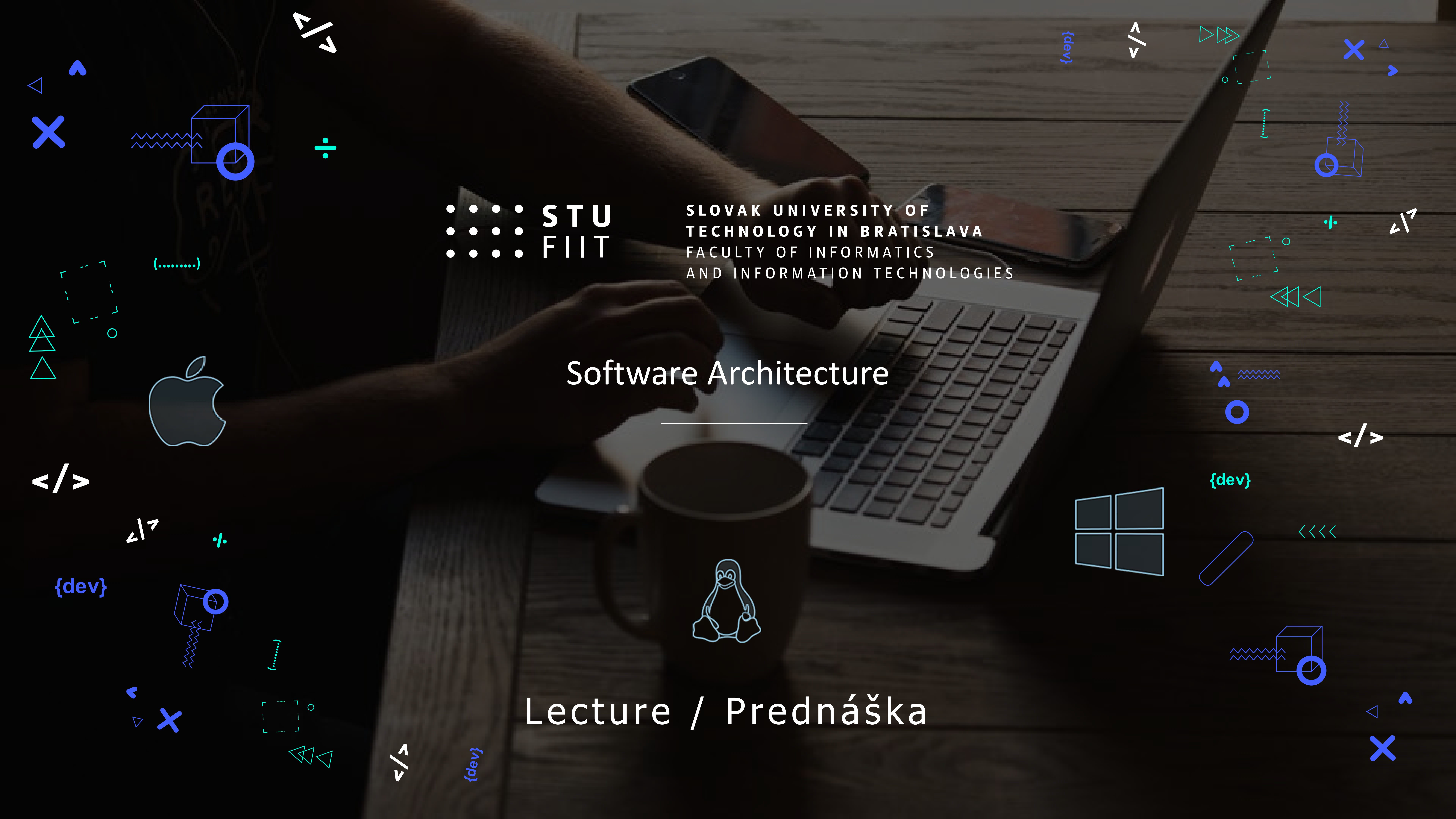


STU FIIT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Software Architecture

Lecture / Prednáška

STU FIIT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

Software Architecture

Microservices and Containers

Mgr. Pavle Dakić, PhD. student

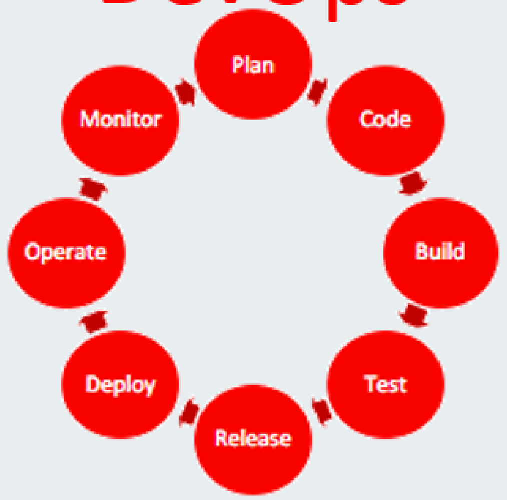
pavle.dakic@stuba.sk

Overview

- Evolution of Computing
- Historic Timeline of Unix Containers
- The challenge in new matrix and the complexity of microservices
- What is containerization?
- What is a container?
- Benefits of containerization
- Types of containerization – OCI
- Microservices and containerization
- What is Docker?
- Practical example

Evolution of Computing

Development Process	Application Architecture	Deployment and Packaging	Application Infrastructure
Waterfall	Monolithic	Physical Server	Datacenter
Agile	N-Tier	Virtual Servers	Hosted
DevOps	Microservices	Containers	Cloud

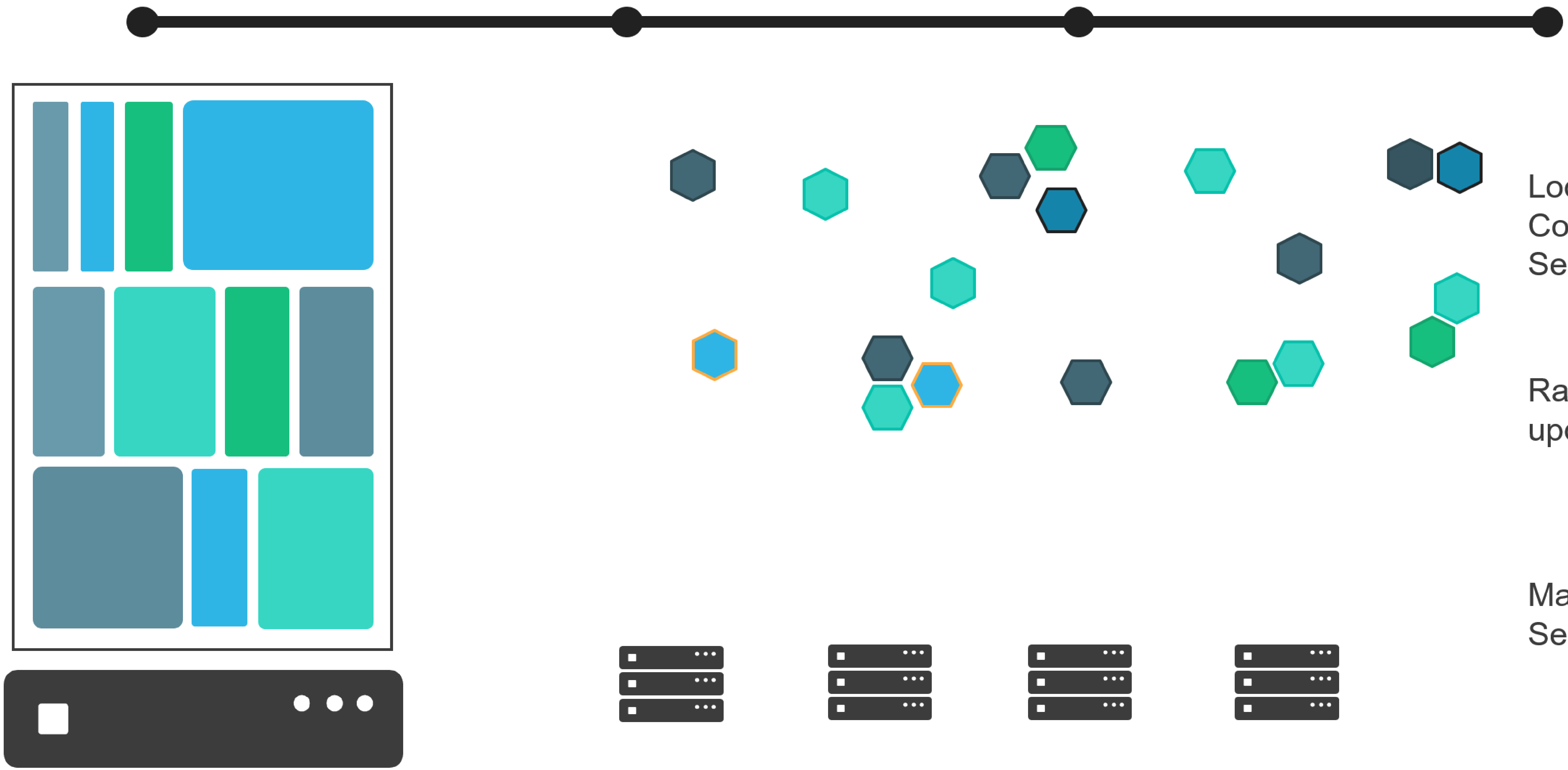


Source: docs.docker.com

History and Multi-Dimensional Evolution of Computing #2

~2000

Today



Monolithic

Slow changing

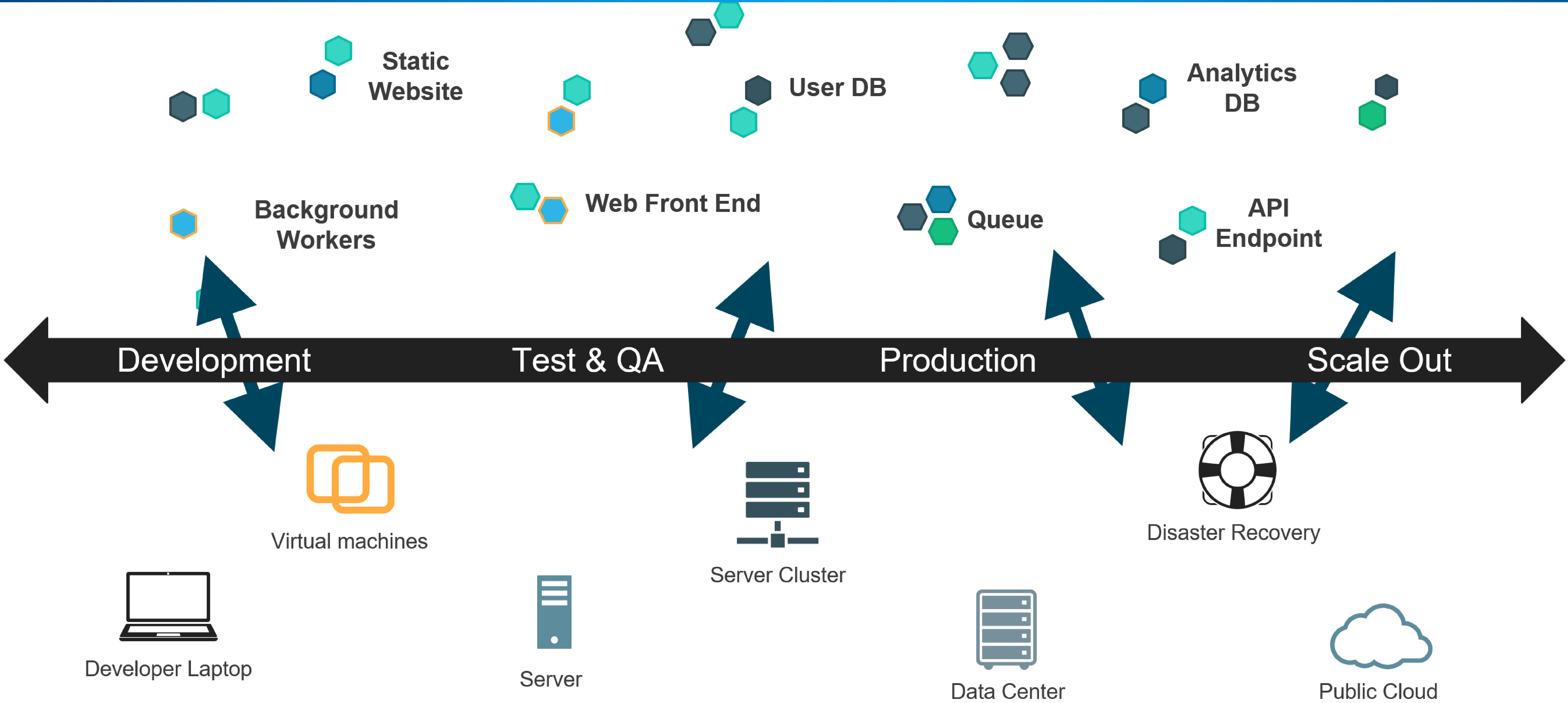
Big Servers

Loosely Coupled Services

Rapidly updated

Many Small Servers

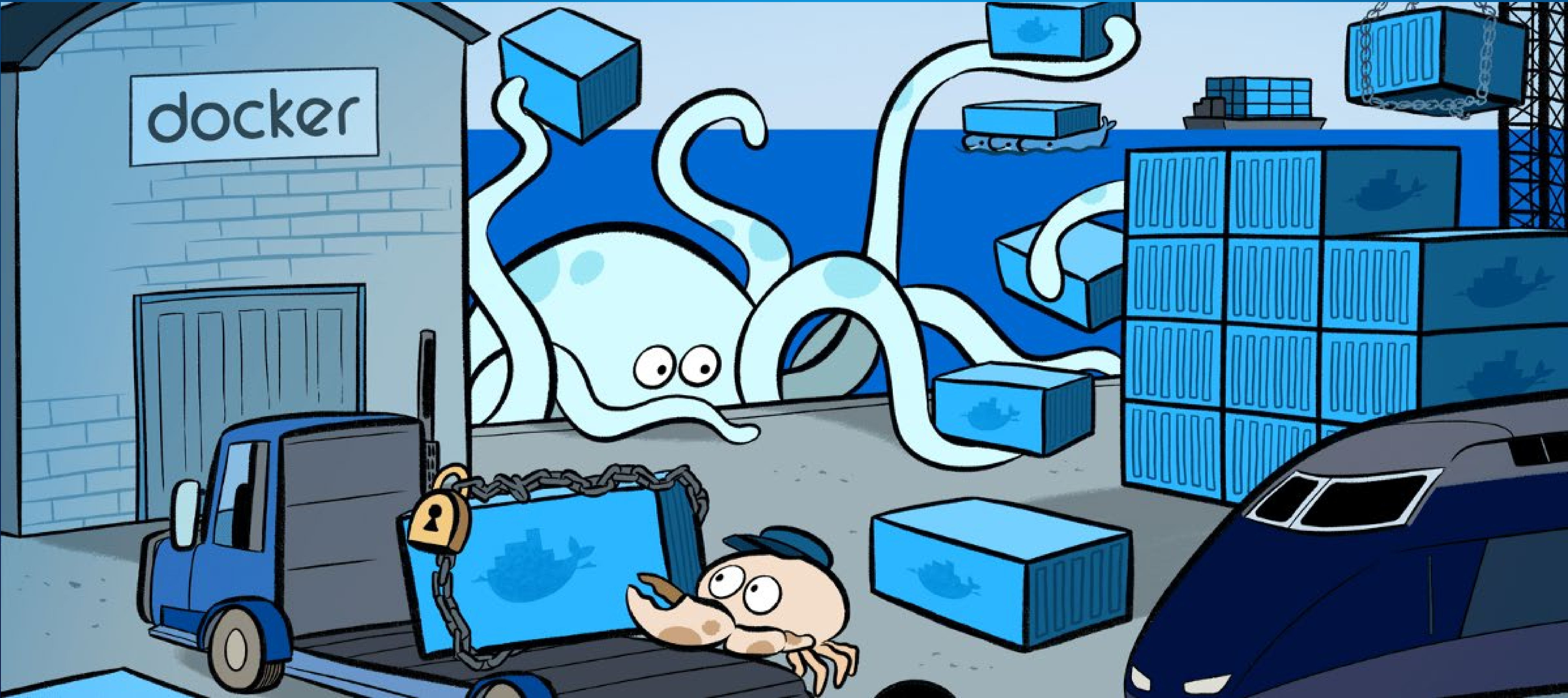
The challenge in new matrix and the complexity of microservices



Historic Timeline of Unix Containers

OCI	2015
rkt	2014
Docker	2013
LMCTFY	2013
Warden	2011
LCX	2008
AIX (6.1)WPARS	2007
cgroups in Linux Kernel (2.6.24)	2007
Process Containers	2006
openvz	2005
Oracle Solaris Zones	2004
Linux vserver	2001
FreeBSD Jails	2000
UNIX V7 added chroot	1979

What is containerization?



What is containerization?

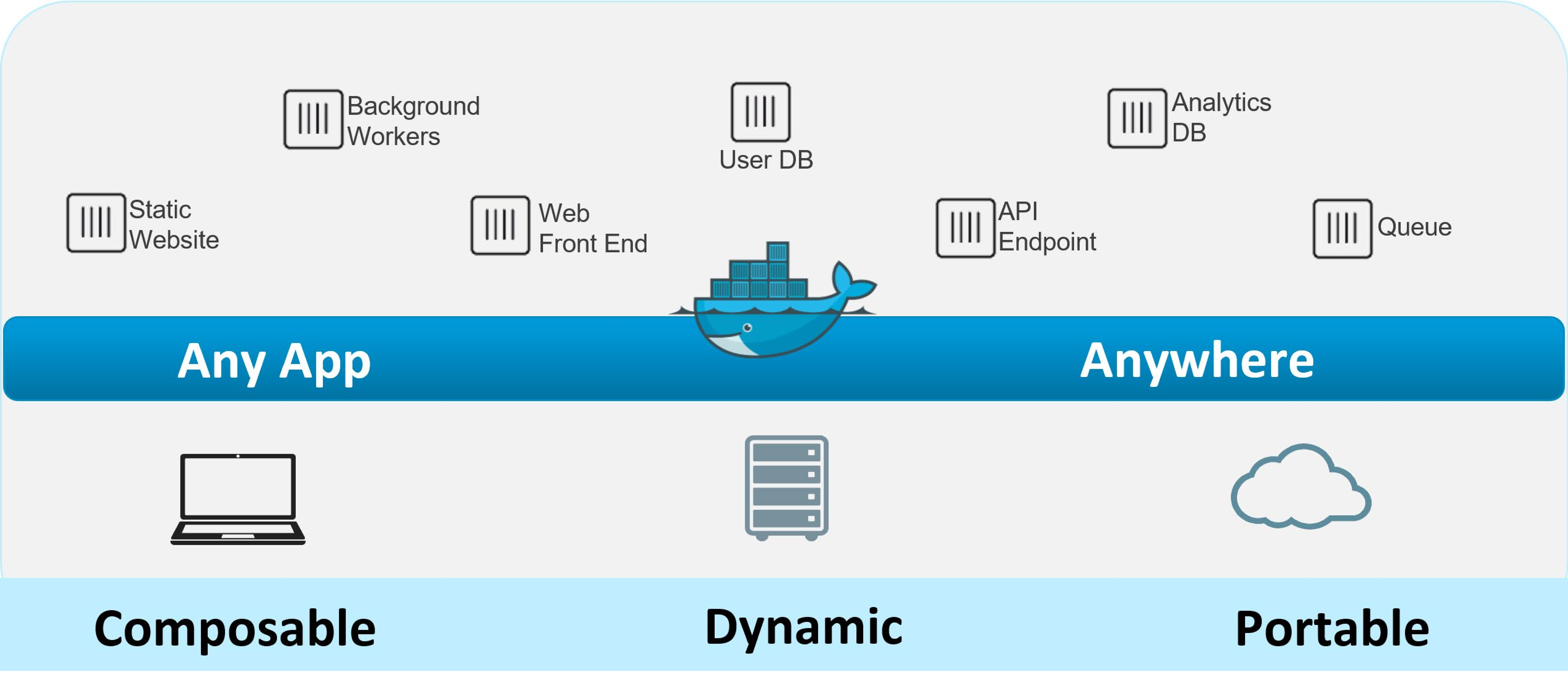
- Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies

- Create a single lightweight executable called a container that runs consistently on any infrastructure

- More portable and resource-efficient than virtual machines (VMs)

- Containers - compute units of modern cloud-native applications

Use of containers

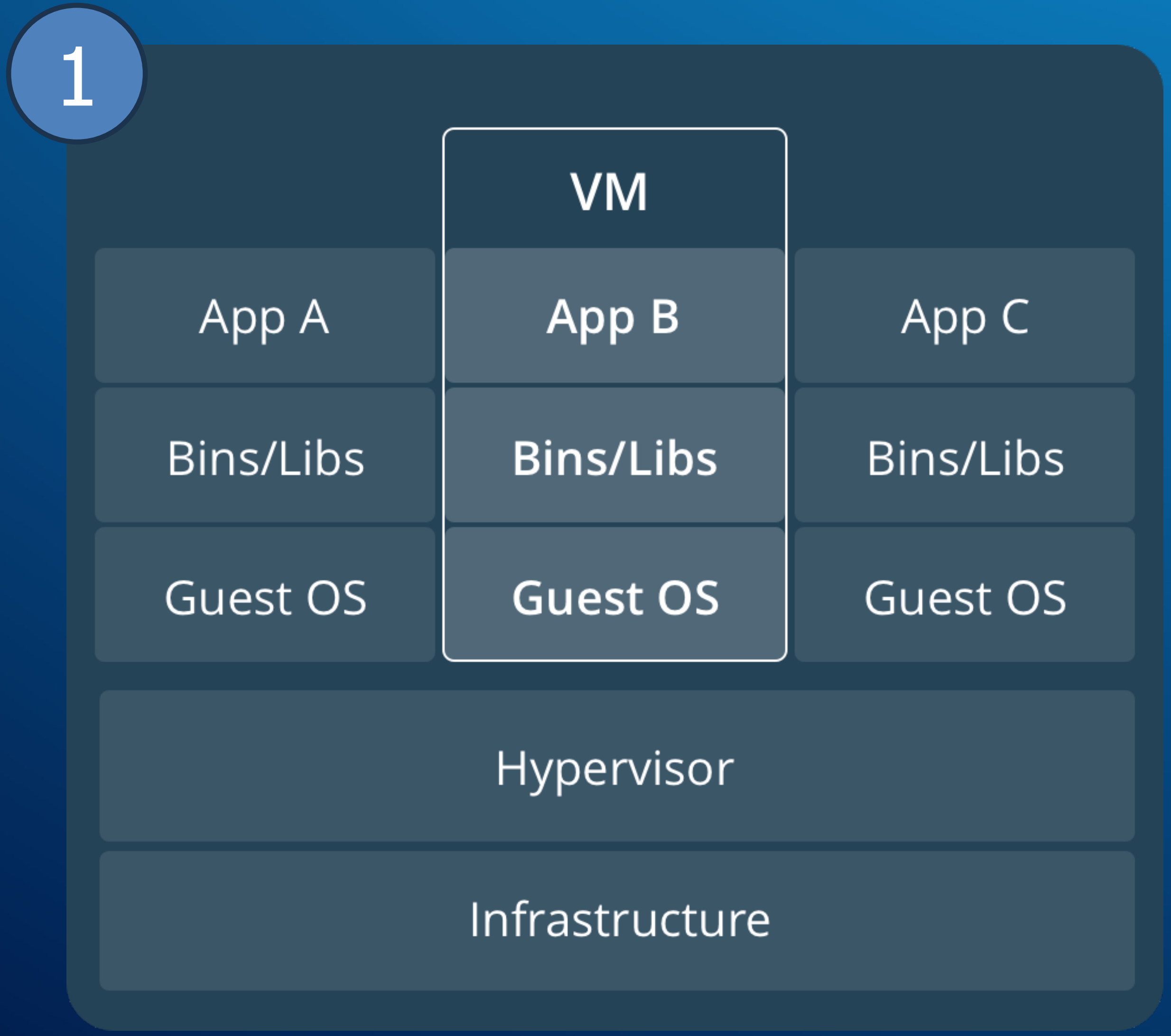


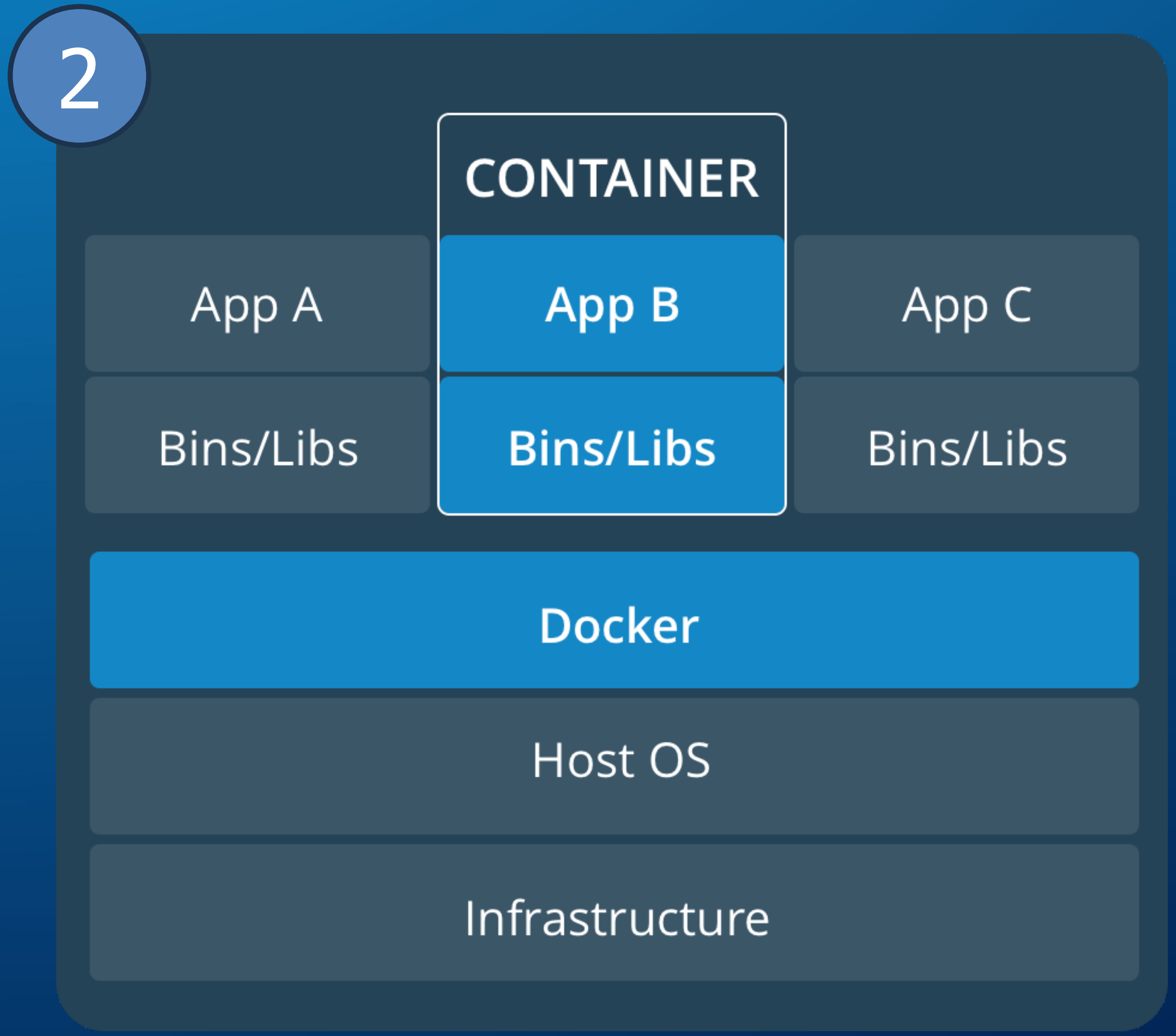
Application containerization

- Containers encapsulate an application as a single executable package of software
- Bundles application code together with all of the related configuration files, libraries, and dependencies required for it to run
- Containerized applications are “isolated” in that they do not bundle in a copy of the operating system
- Other container layers, like common bins and libraries, can also be shared among multiple containers

What is a container?

- Container ≠ VM
- Isolated
- Share OS
- and sometimes bins/libs





Source: docs.docker.com

STU FIIT

Developers

IT Operations

BUILD

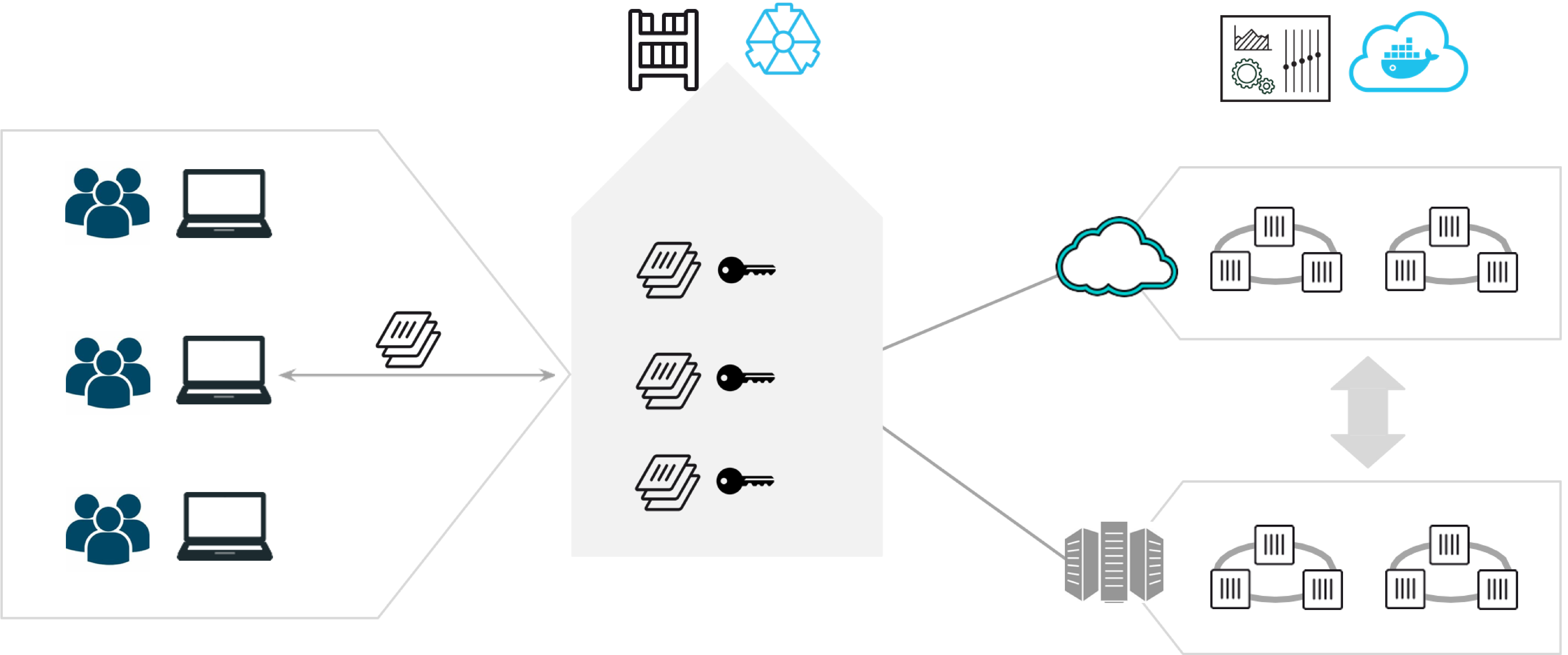
Development Environments

SHIP

Create & Store Images

RUN

Deploy, Manage, Scale



Benefits of containerization

- Containerization offers significant benefits to developers and development teams

- **Agility**: The open source Docker Engine for running containers started the industry standard for containers

- Simple developer tools and a universal packaging approach that works on both Linux and Windows operating systems

- **Speed:** Containers are often referred to as “lightweight,” meaning they share the machine’s operating system (OS) kernel

Benefits of containerization #2

- **Fault isolation:** Each containerized application is isolated and operates independently of others

- **Efficiency:** Software running in containerized environments shares the machine’s OS kernel

- Application layers within a container can be shared across containers

Benefits of containerization #3

- **Ease of management:** A container orchestration platform automates the installation, scaling, and management of containerized workloads and services

- **Security:** The isolation of applications as containers inherently prevents the invasion of malicious code from affecting other containers or the host system

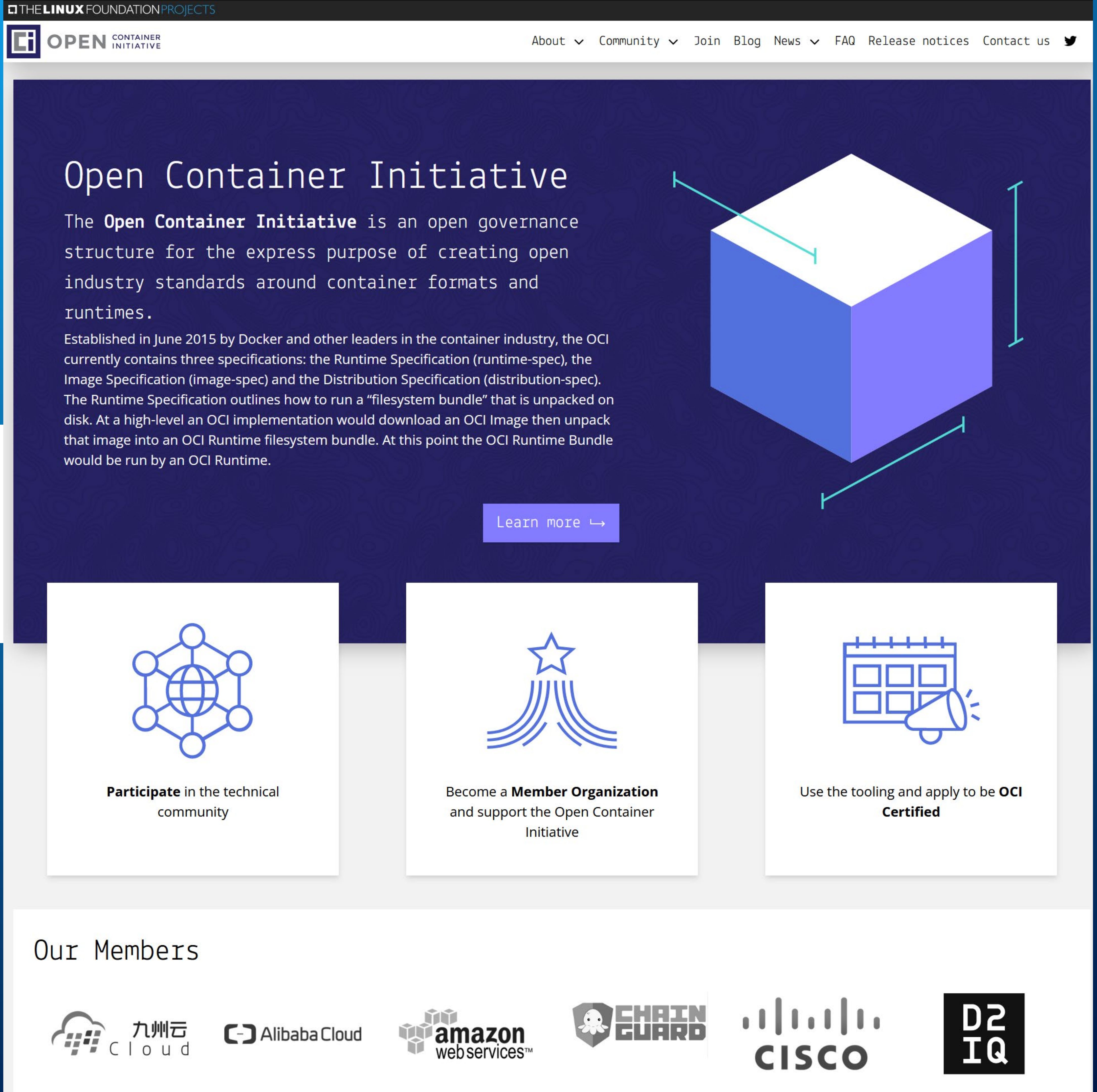
Types of containerization - OCI

- The rapid growth in interest and usage of container-based solutions

- Led to the need for standards around container technology and the approach to packaging software code

- **The Open Container Initiative (OCI),** established in June 2015 by Docker and other industry leaders https://opencontainers.org/

OPEN CONTAINER INITIATIVE



Open Container Initiative

The **Open Container Initiative** is an open governance structure for the express purpose of creating open industry standards around container formats and runtimes.

Established in June 2015 by Docker and other leaders in the container industry, the OCI currently contains three specifications: the Runtime Specification (runtime-spec), the Image Specification (image-spec) and the Distribution Specification (distribution-spec). The Runtime Specification outlines how to run a “filesystem bundle” that is unpacked on disk. At a high-level an OCI implementation would download an OCI Image then unpack that image into an OCI Runtime filesystem bundle. At this point the OCI Runtime Bundle would be run by an OCI Runtime.

Participate in the technical community

Become a **Member Organization** and support the Open Container Initiative

Use the tooling and apply to be **OCI Certified**

Our Members

Types of containerization - OCI #2

- Promoting common, minimal, open standards and specifications around container technology

- Users will not be locked into a particular vendor’s technology

- They will be able to take advantage of OCI-certified technologies that allow them to build containerized applications

Microservices and containerization

- Software companies large and small are embracing microservices as a superior approach to application development and management, compared to the earlier monolithic model

- With microservices, a complex application is broken up into a series of smaller, more specialized services, each with its own database and its own business logic

- Microservices then communicate with each other across common interfaces (like APIs) and REST interfaces (like HTTP)

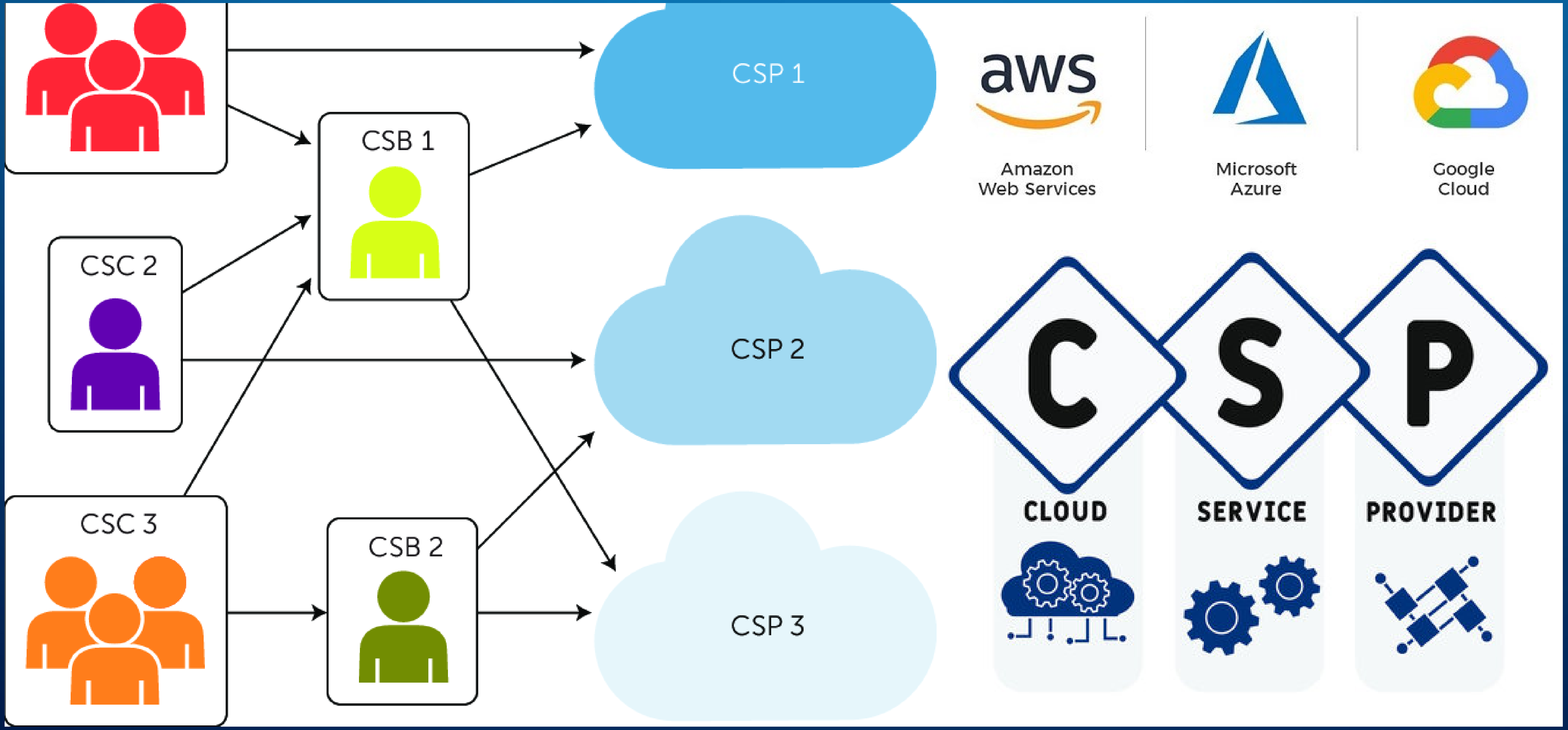
Microservices and containerization #2

- The concepts behind microservices and containerization are similar as both are software development practices
- They essentially transform applications into collections of smaller services or components which are portable, scalable, efficient and easier to manage
- Microservices and containerization work well when used together
- Containers provide a lightweight encapsulation of any application, whether it is a traditional monolith or a modular microservice

Microservices and containerization #3

- Cloud-based applications and data are accessible from any internet-connected device, allowing team members to work remotely and on-the-go

- Cloud service providers (CSPs) manage the underlying infrastructure



Microservices and containerization #4

- Saves organizations the cost of servers and other equipment and also provides automated network backups for additional reliability
- Cloud infrastructures scale on demand and can dynamically adjust
- Computing resources
- Capacity
- Infrastructure as load requirements change

TOOLBOX

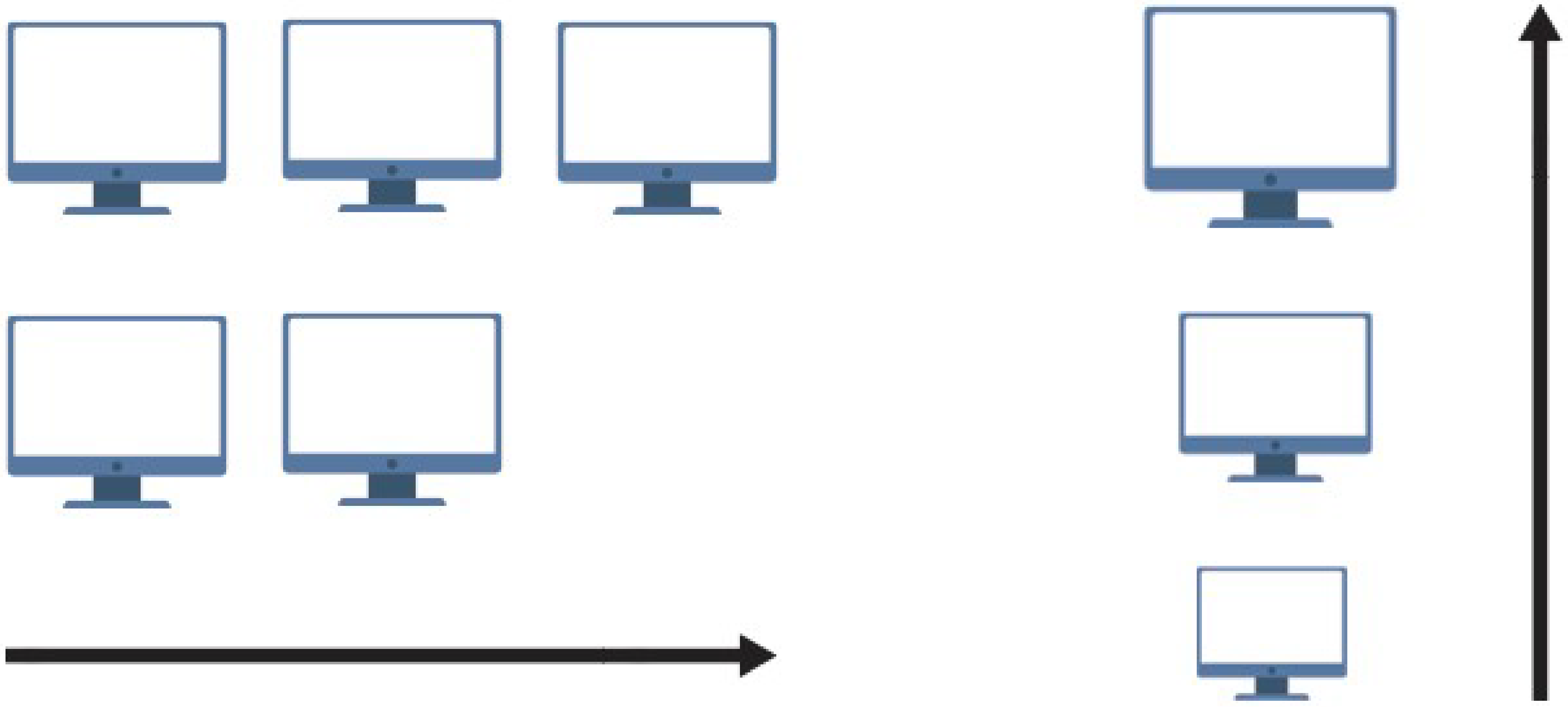
Horizontal Scaling vs. Vertical Scaling

Horizontal Scaling

Add more instances

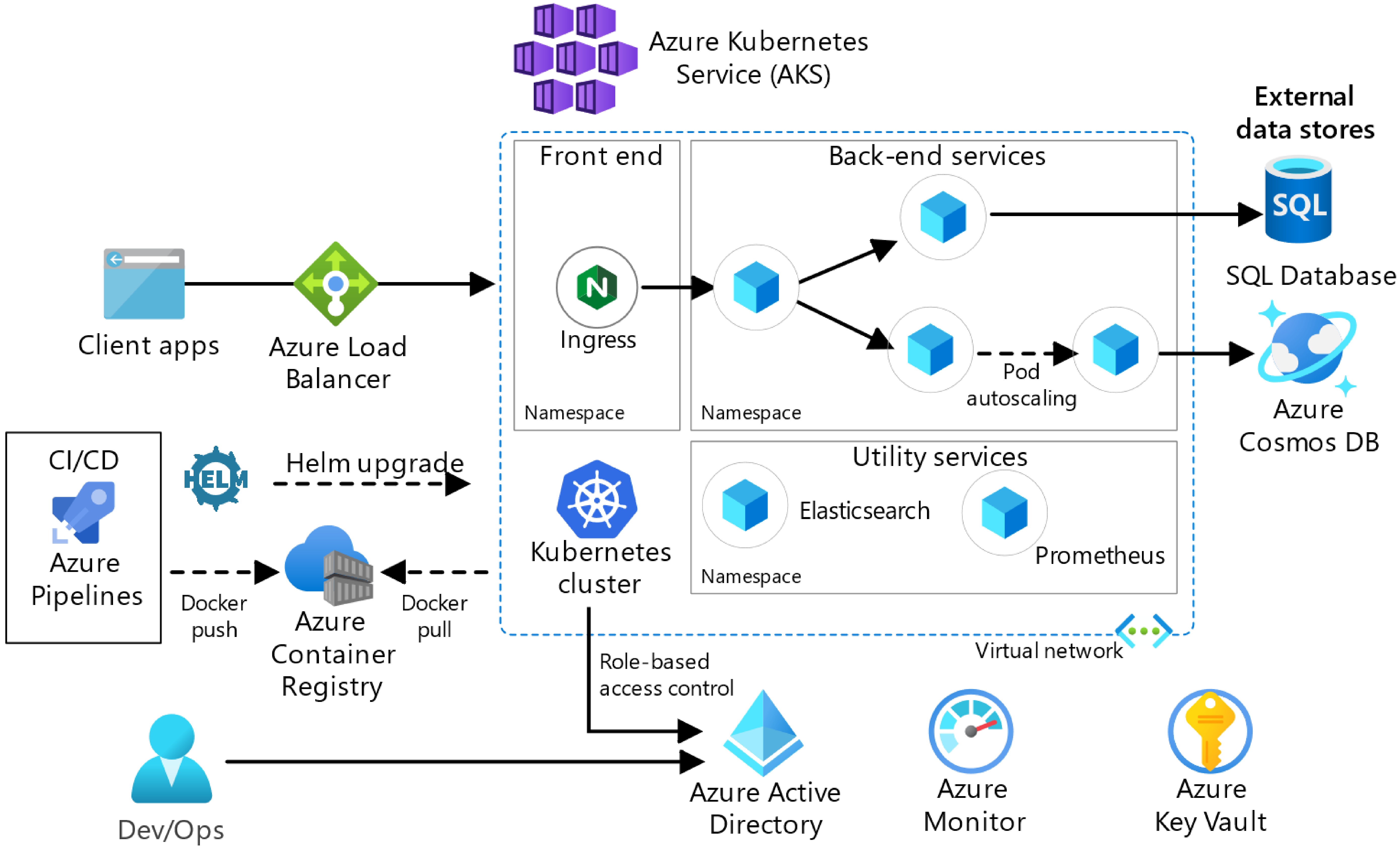
Vertical Scaling

Increase size of instances (RAM, CPU, etc.)



Microservices and containerization #4

- CSPs regularly update offerings, giving users continued access to the latest innovative technology
- Containers, microservices, and cloud computing bring application development and delivery to new levels
- This is not possible with traditional methodologies and environments
- These next-generation approaches add agility, efficiency, reliability, and security to the software development lifecycle



Source: https://learn.microsoft.com/en-us/azure/architecture/reference-architectures/containers/aks-microservices/aks-microservices

Security

- Containerized applications inherently have a level of security since they can run as isolated processes and can operate independently of other containers

- This could prevent any malicious code from affecting other containers or invading the host system

- In terms of resource efficiency, this is a plus, but it also opens the door to interference and security breaches across containers

Security #2

- For example, Linux Namespaces helps to provide an isolated view of:
  - the system to each container
  - networking
  - mount points,
  - process IDs
  - user IDs
  - inter-process communication
  - and hostname settings

- Researchers are working to further strengthen Linux container security, to automate threat detection and response across an enterprise, to monitor and enforce compliance to meet industry standards and security policies

What is Docker?

- Docker is both a Company and Technology
- While Docker has been playing a key role in adoption of the Linux container technology, they did not invent the concept of containers
- However, they have made the technology consumable by mere humans

Source: docs.docker.com

What is Docker? #2

- Docker is a software platform that allows you to build, test, and deploy applications quickly

- While packaging software into standardized units called containers

Docker Architecture

- Docker client – Command Line Interface (CLI) for interfacing with the Docker

```
docker ps
```

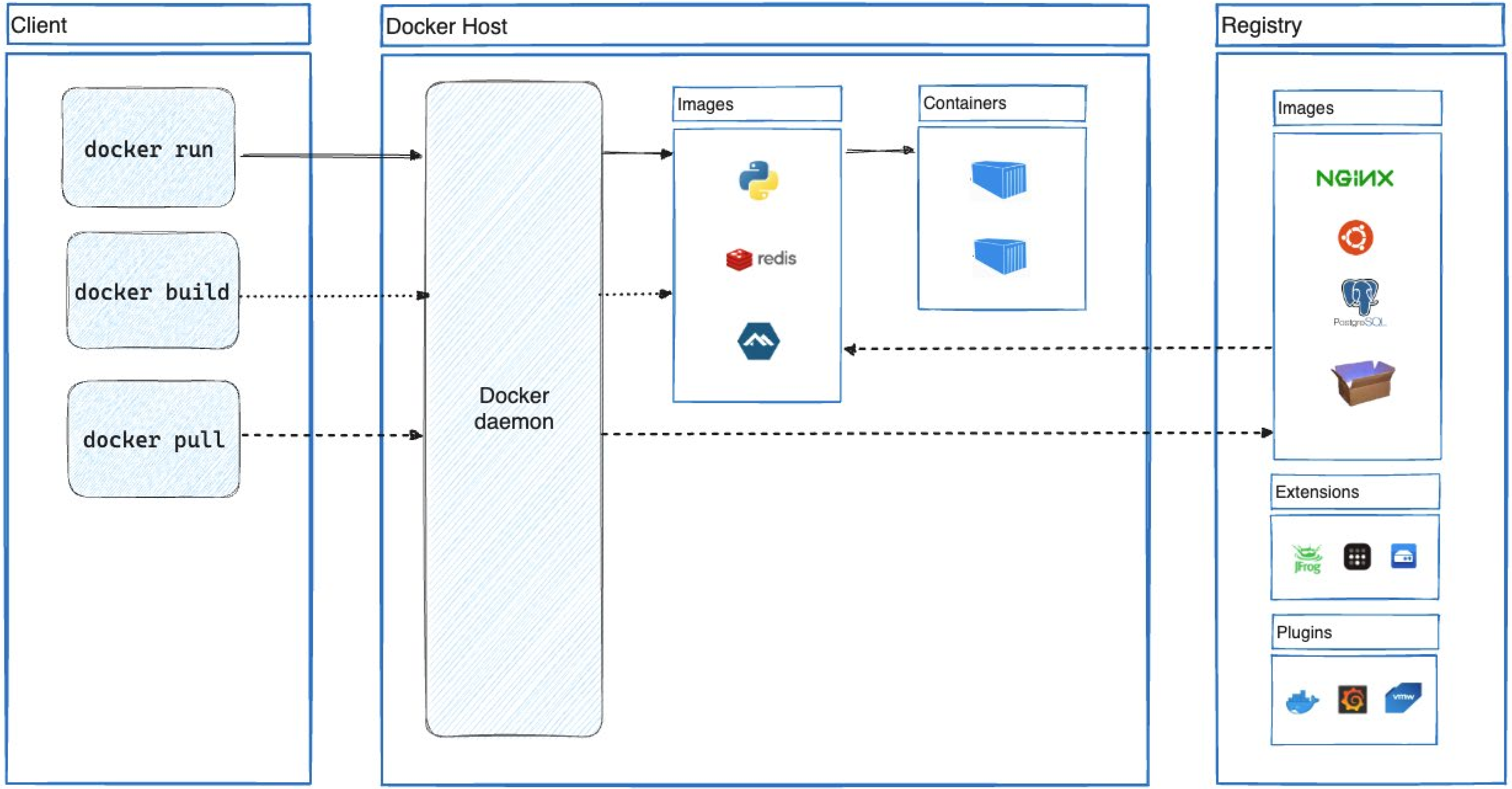
- Dockerfile – Text file of Docker instructions used to assemble a Docker Image

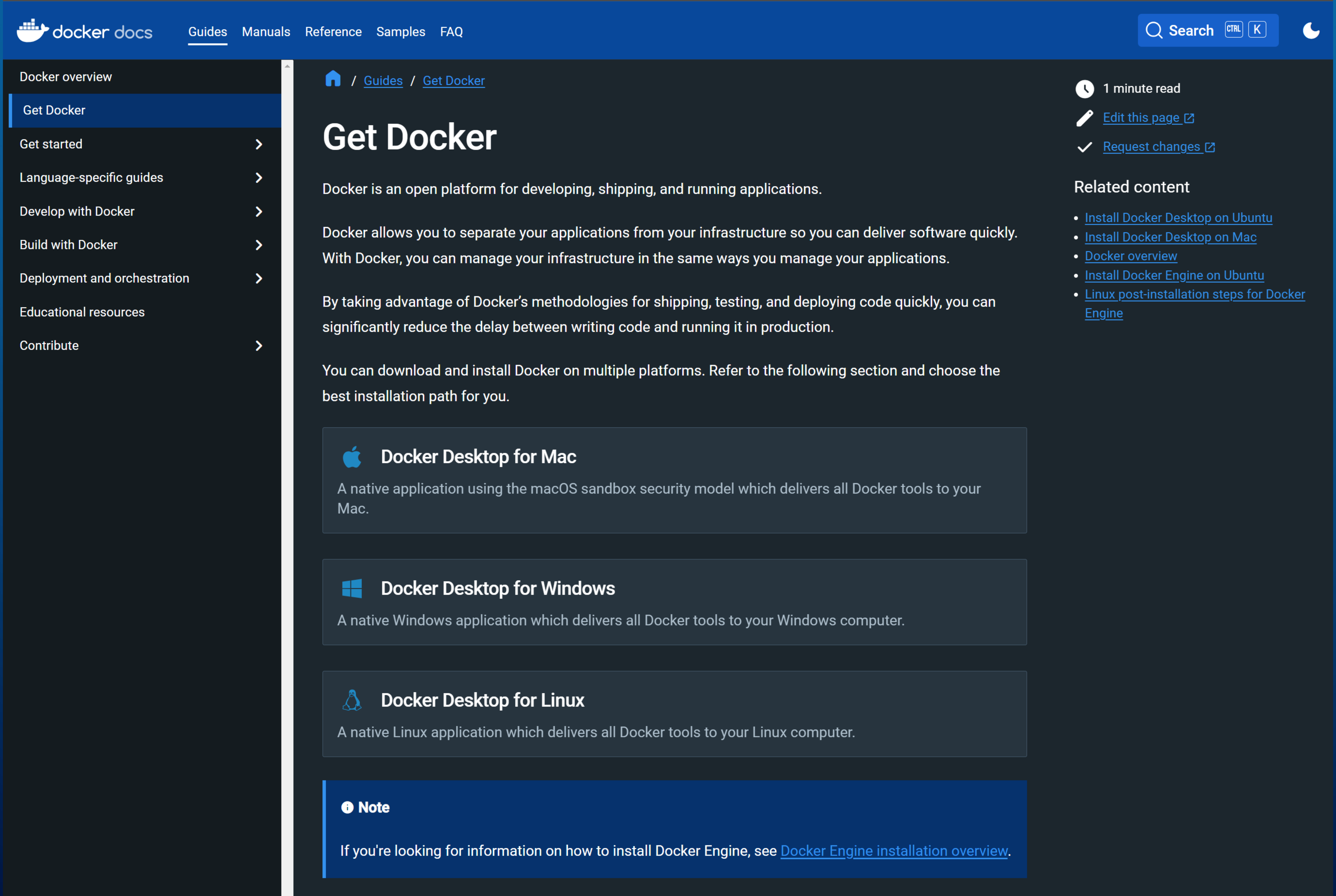
- Image – Hierarchies of files built from a Dockerfile, the file used as input to the docker build command

Source: Docker docs and https://docs.docker.com/glossary/

Docker Architecture #2

- Docker Engine - Creates, ships and runs Docker containers
- Container – Running instance of an Image using the docker run command
- Registry – Image repository
- Docker Hub (Public) or Docker Trusted Registry (Private)
  - Cloud or server based storage and distribution service for your images


Client
docker run
docker build
docker pull
Docker Host
Docker daemon
Images
redis
Containers
Registry
Images
NGINX
PostgreSQL
Extensions
JFrog
Plugins
vmw



Get Docker

Docker is an open platform for developing, shipping, and running applications.

Docker allows you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications.

By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

You can download and install Docker on multiple platforms. Refer to the following section and choose the best installation path for you.

Docker Desktop for Mac

A native application using the macOS sandbox security model which delivers all Docker tools to your Mac.

Docker Desktop for Windows

A native Windows application which delivers all Docker tools to your Windows computer.

Docker Desktop for Linux

A native Linux application which delivers all Docker tools to your Linux computer.

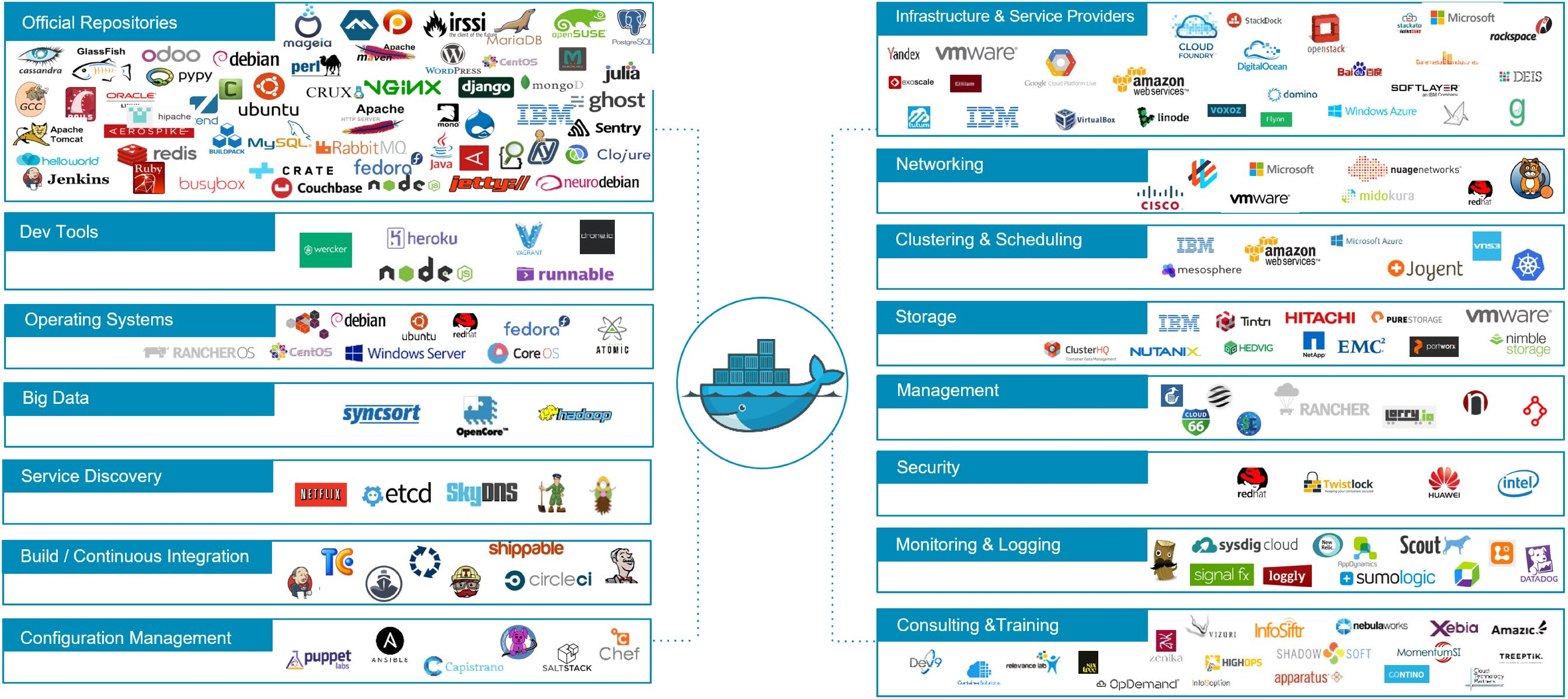
Note

If you're looking for information on how to install Docker Engine, see Docker Engine installation overview.

Related content

- Install Docker Desktop on Ubuntu
- Install Docker Desktop on Mac
- Docker overview
- Install Docker Engine on Ubuntu
- Linux post-installation steps for Docker Engine

The Docker ecosystem



Official Repositories

Dev Tools

Operating Systems

Big Data

Service Discovery

Build / Continuous Integration

Configuration Management

Infrastructure & Service Providers

Networking

Clustering & Scheduling

Storage

Management

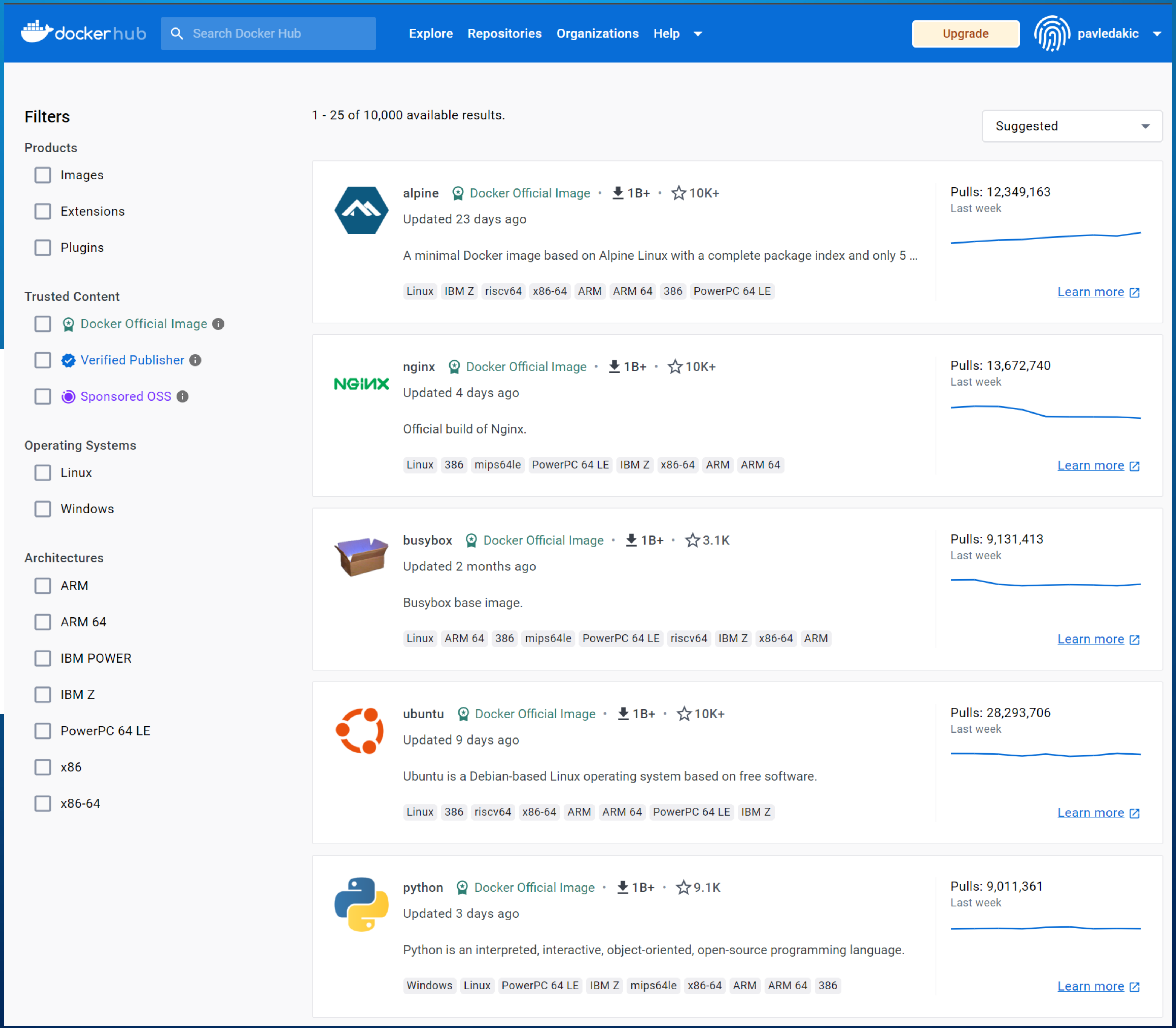
Security

Monitoring & Logging

Consulting &Training

Docker images

- Docker Hub



docker hub Search Docker Hub Explore Repositories Organizations Help Upgrade pavledakic

Filters

Products

☐ Images

☐ Extensions

☐ Plugins

Trusted Content

☐ Docker Official Image

☐ Verified Publisher

☐ Sponsored OSS

Operating Systems

☐ Linux

☐ Windows

Architectures

☐ ARM

☐ ARM 64

☐ IBM POWER

☐ IBM Z

☐ PowerPC 64 LE

☐ x86

☐ x86-64

1 - 25 of 10,000 available results.

Suggested

alpine · Docker Official Image · 1B+ · ☆ 10K+
Updated 23 days ago
A minimal Docker image based on Alpine Linux with a complete package index and only 5 ...
Linux IBM Z riscv64 x86-64 ARM ARM 64 386 PowerPC 64 LE
Pulls: 12,349,163
Last week
Learn more

nginx · Docker Official Image · 1B+ · ☆ 10K+
Updated 4 days ago
Official build of Nginx.
Linux 386 mips64le PowerPC 64 LE IBM Z x86-64 ARM ARM 64
Pulls: 13,672,740
Last week
Learn more

busybox · Docker Official Image · 1B+ · ☆ 3.1K
Updated 2 months ago
Busybox base image.
Linux ARM 64 386 mips64le PowerPC 64 LE riscv64 IBM Z x86-64 ARM
Pulls: 9,131,413
Last week
Learn more

ubuntu · Docker Official Image · 1B+ · ☆ 10K+
Updated 9 days ago
Ubuntu is a Debian-based Linux operating system based on free software.
Linux 386 riscv64 x86-64 ARM ARM 64 PowerPC 64 LE IBM Z
Pulls: 28,293,706
Last week
Learn more

python · Docker Official Image · 1B+ · ☆ 9.1K
Updated 3 days ago
Python is an interpreted, interactive, object-oriented, open-source programming language.
Windows Linux PowerPC 64 LE IBM Z mips64le x86-64 ARM ARM 64 386
Pulls: 9,011,361
Last week
Learn more

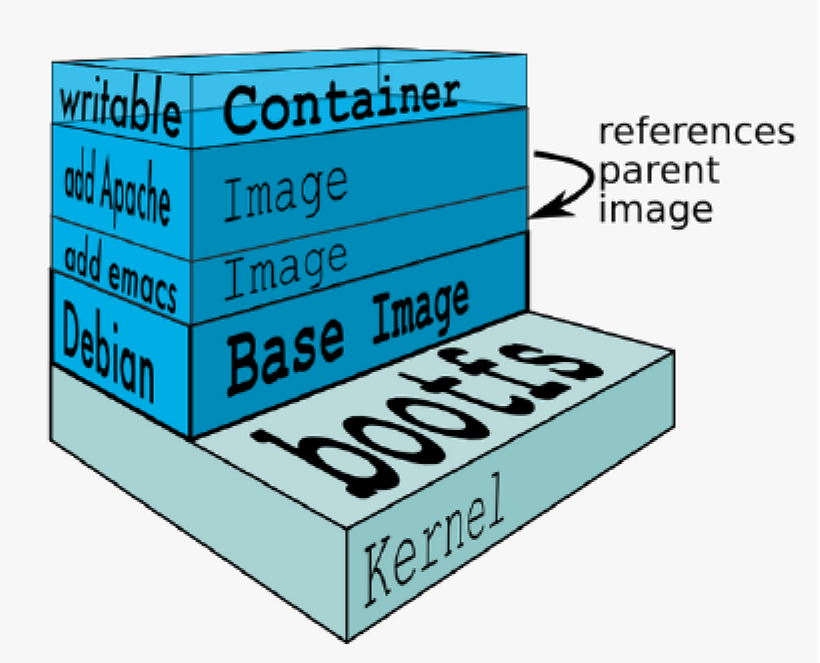
STU FIIT

Docker images #2

- Images are comprised of multiple layers, multiple layers referencing/based on another image (Union File System)

- It is possible to build your own images reading instructions from a Dockerfile

An image is a collection of files and some meta data



Dockerfile example

```
FROM centos:7
RUN yum install -y python-devel python-virtualenv
RUN virtualenv /opt/pyapp/venv
COPY runpoint.sh /opt/runpoint.sh
EXPOSE 8000
ENTRYPOINT /opt/pyapp/runpoint.sh
```

docker-compose

- Allows to run multi-container Docker applications reading instructions from a `docker-compose.yml` file

docker-compose.yml - example

1

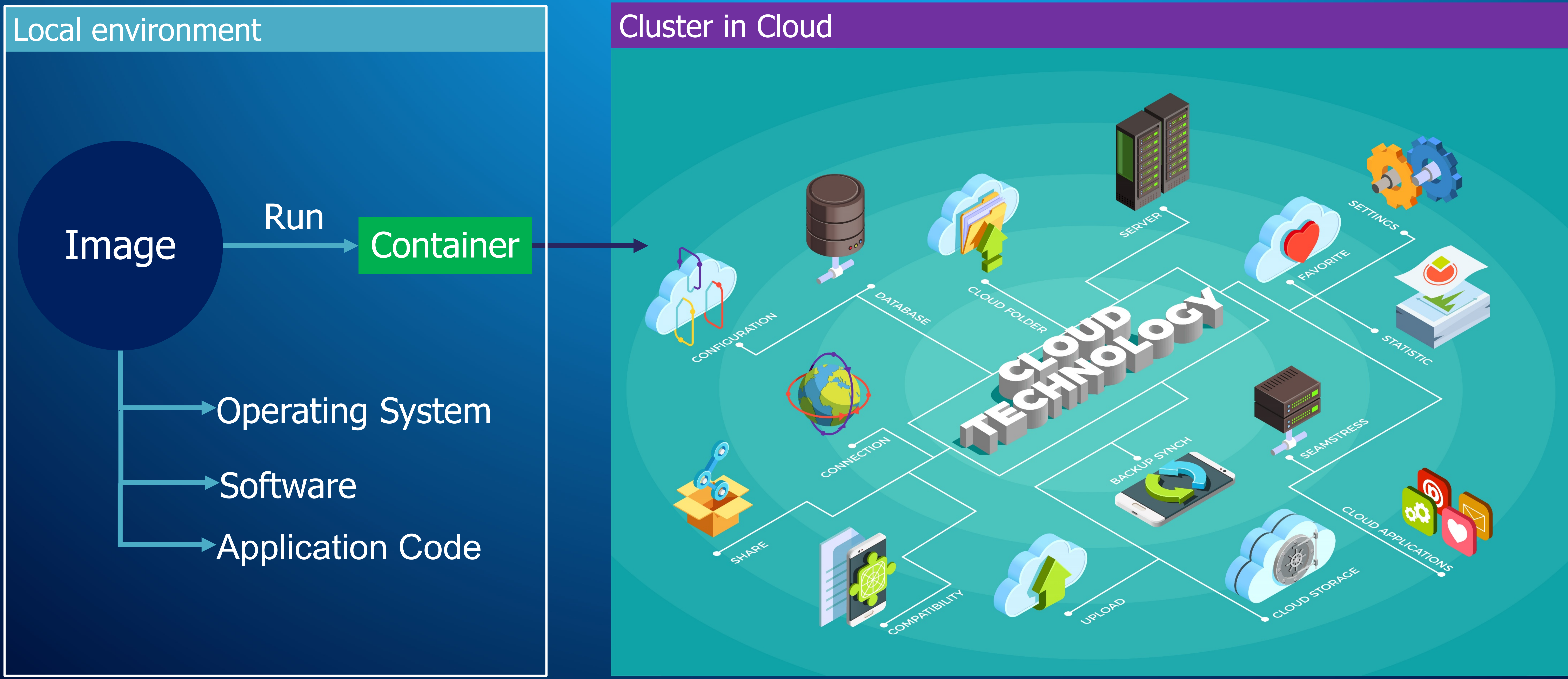
```yaml
version: "2"
services:
  example-application:
    build: ./
    ports:
      - "8000:8000"
    environment:
      - CONFIG_FILE
  db:
    image: postgres
  redis:
    image: redis
    command: redis-server --save "" --appendonly no
    ports:
      - "6179"
```

Bash - terminal

```bash
$ cd example-docker
$ docker-compose up
```

2

Docker Flow



Local environment

Image

Run

Container

Operating System

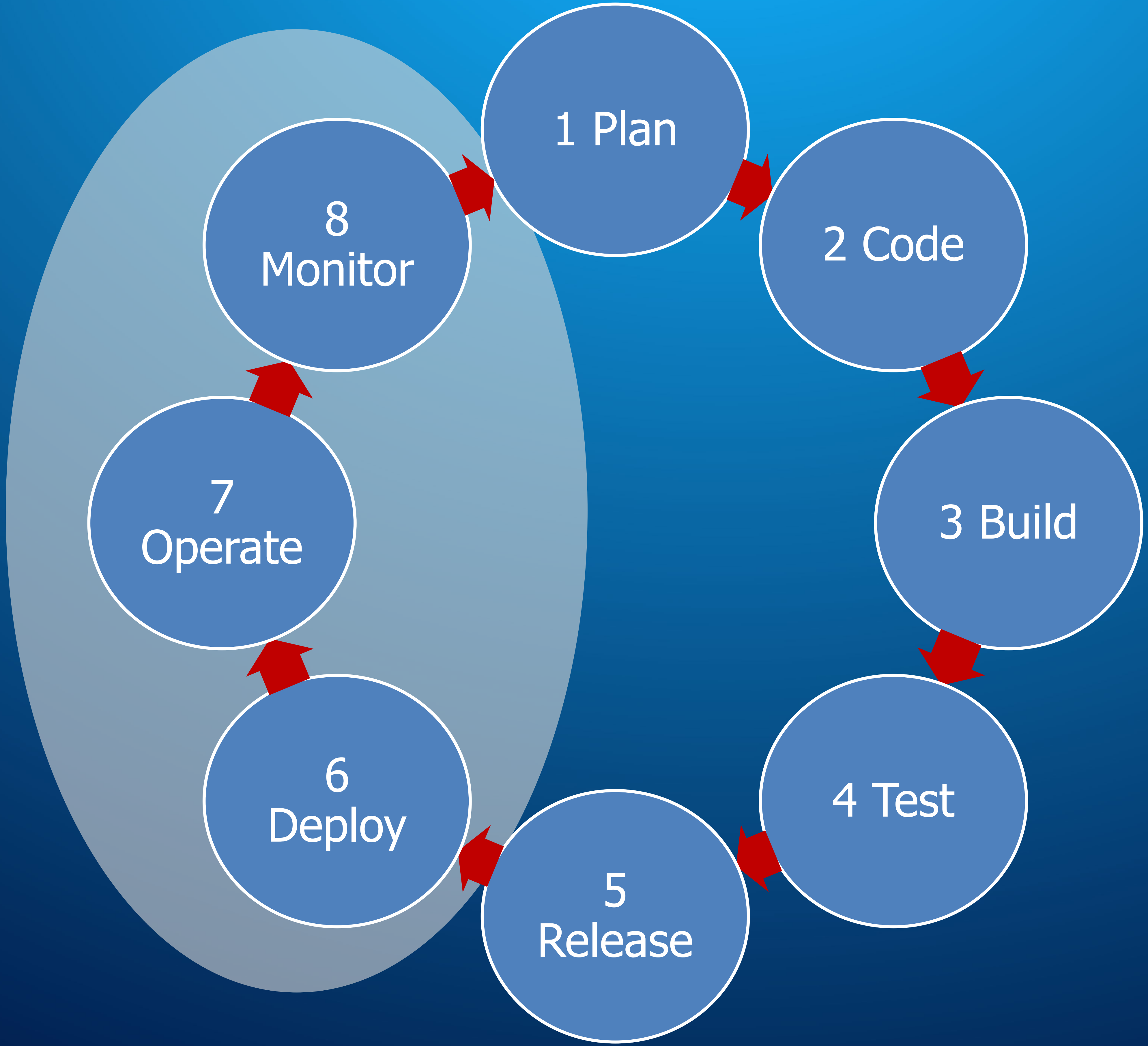
Software

Application Code

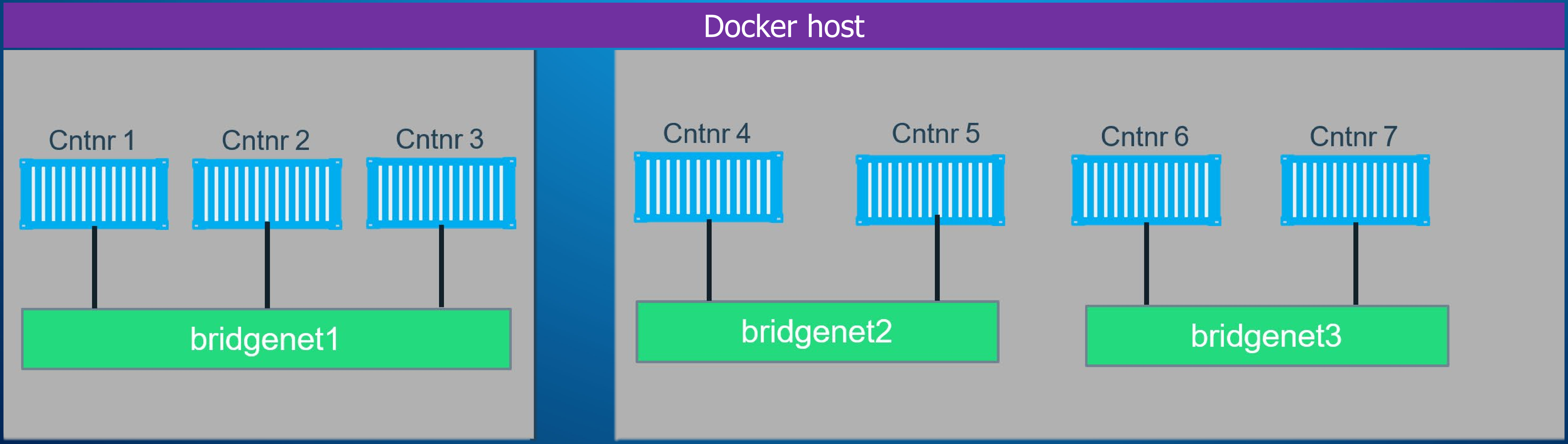
Cluster in Cloud

Source: docs.docker.com

DevOps Cycle



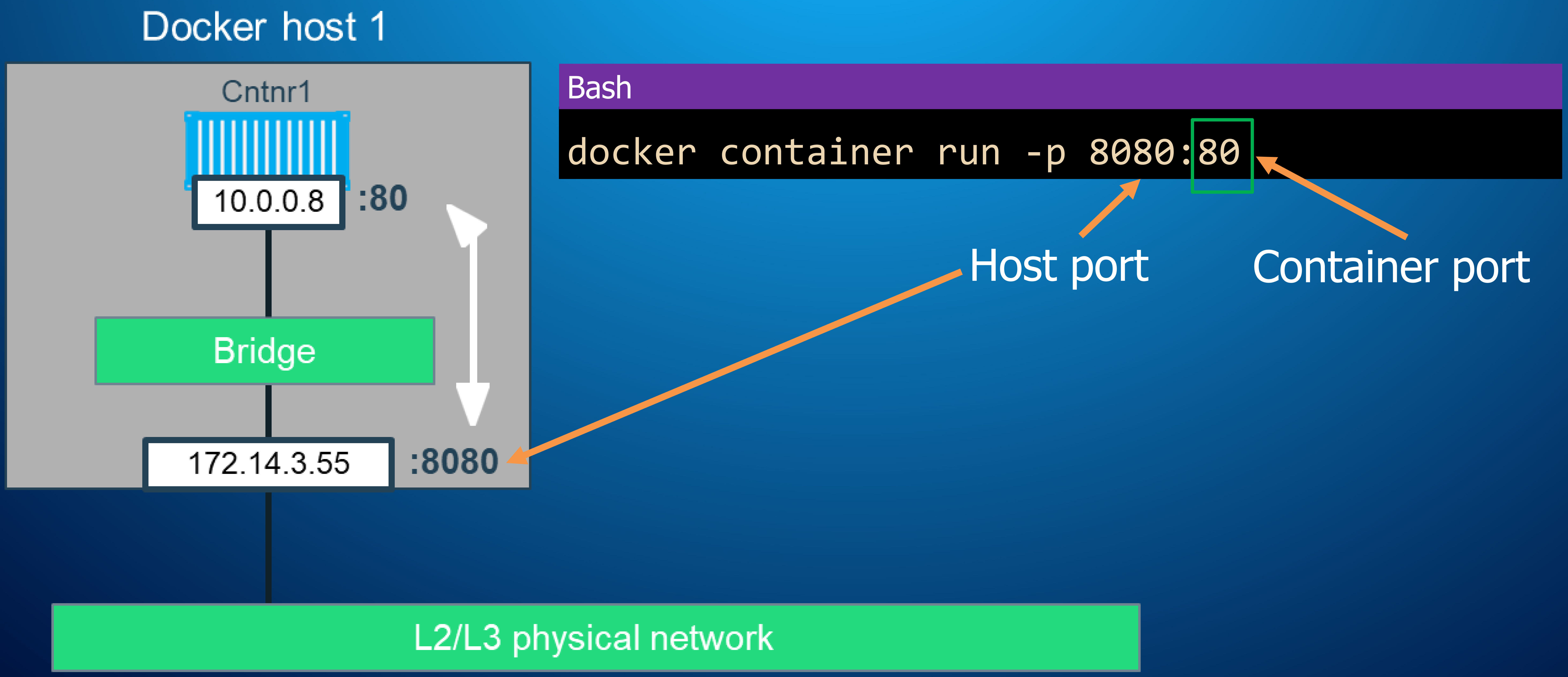
What is Docker Bridge Networking?



Bash

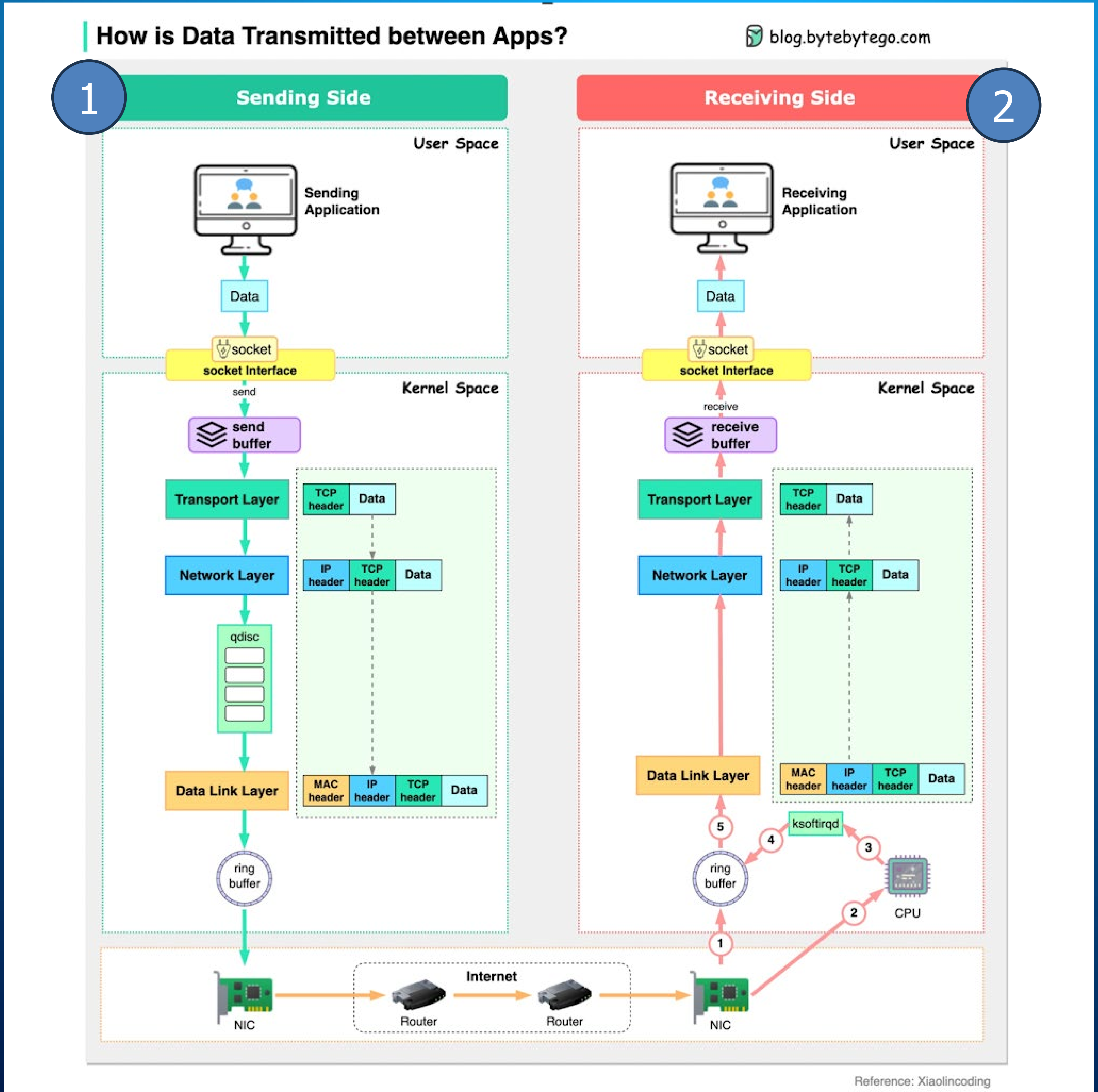
```
docker network create -d bridge --name bridgenet1
```

Docker Bridge Networking and Port Mapping



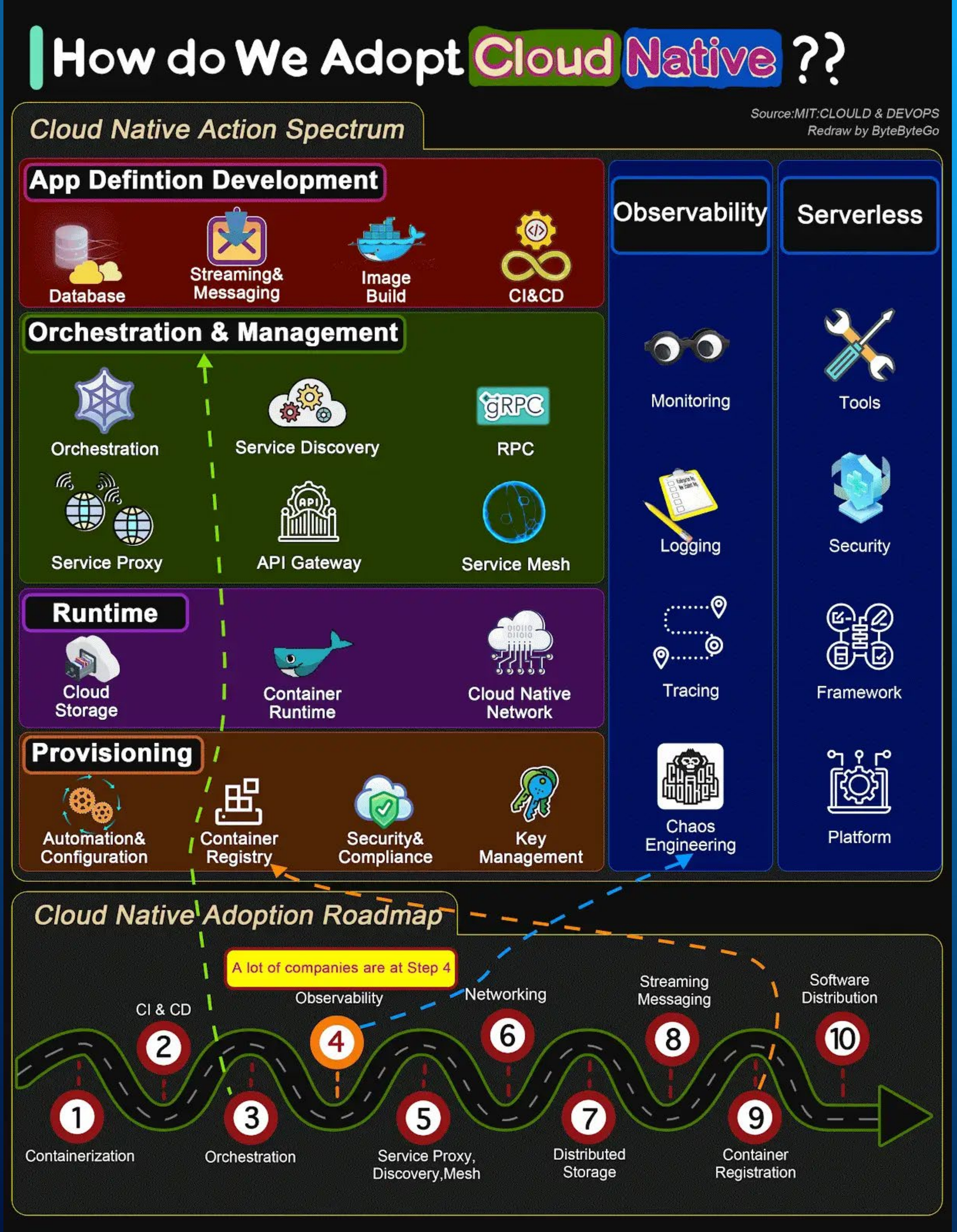
Bash

```
docker container run -p 8080:80
```



How is Data Transmitted between Apps?

Reference: Xiaolincoding



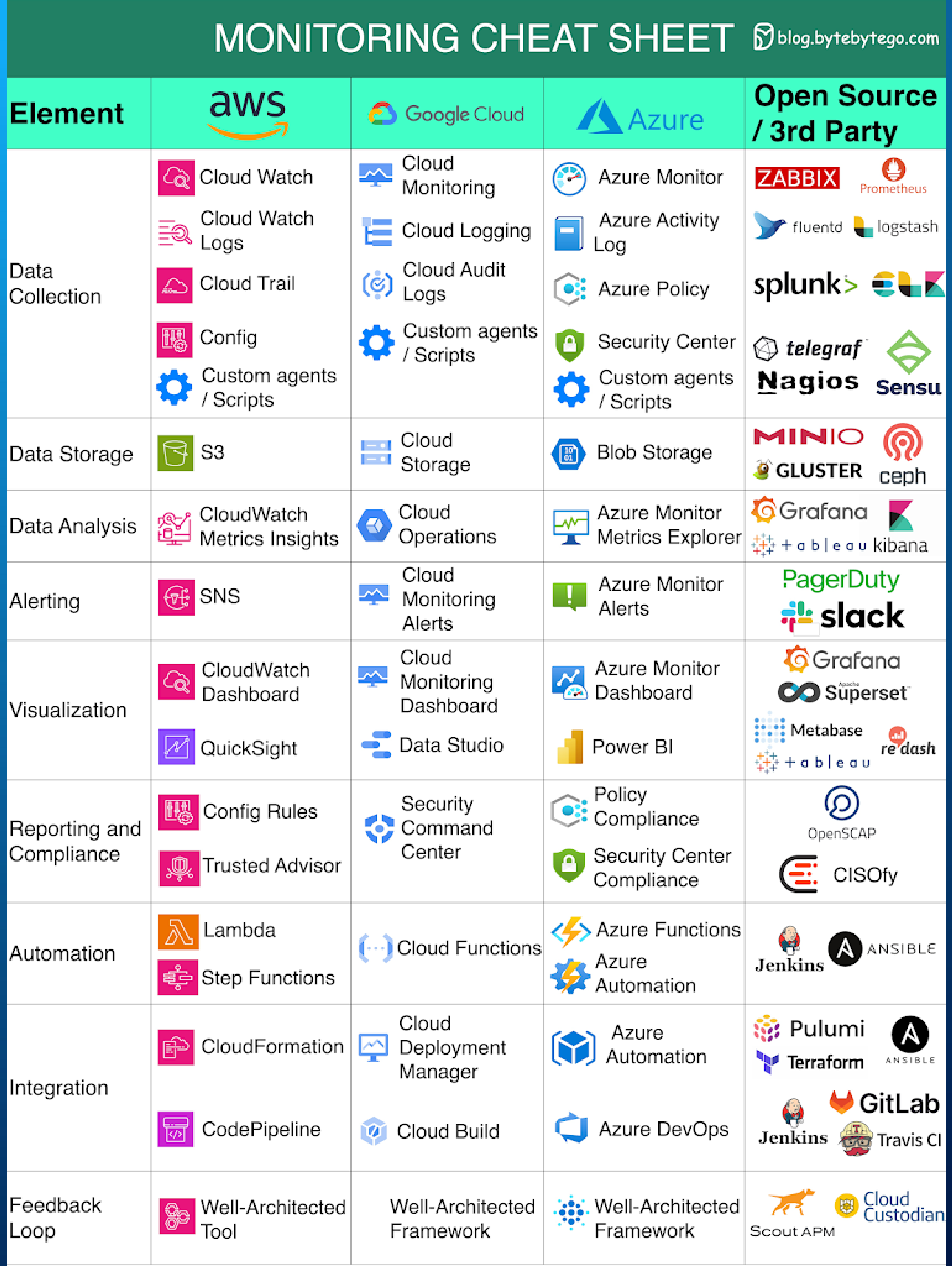
How do We Adopt Cloud Native ??

Cloud Native Action Spectrum

Source:MIT:CLOUD & DEVOPS
Redraw by ByteByteGo

Cloud Native Adoption Roadmap

A lot of companies are at Step 4



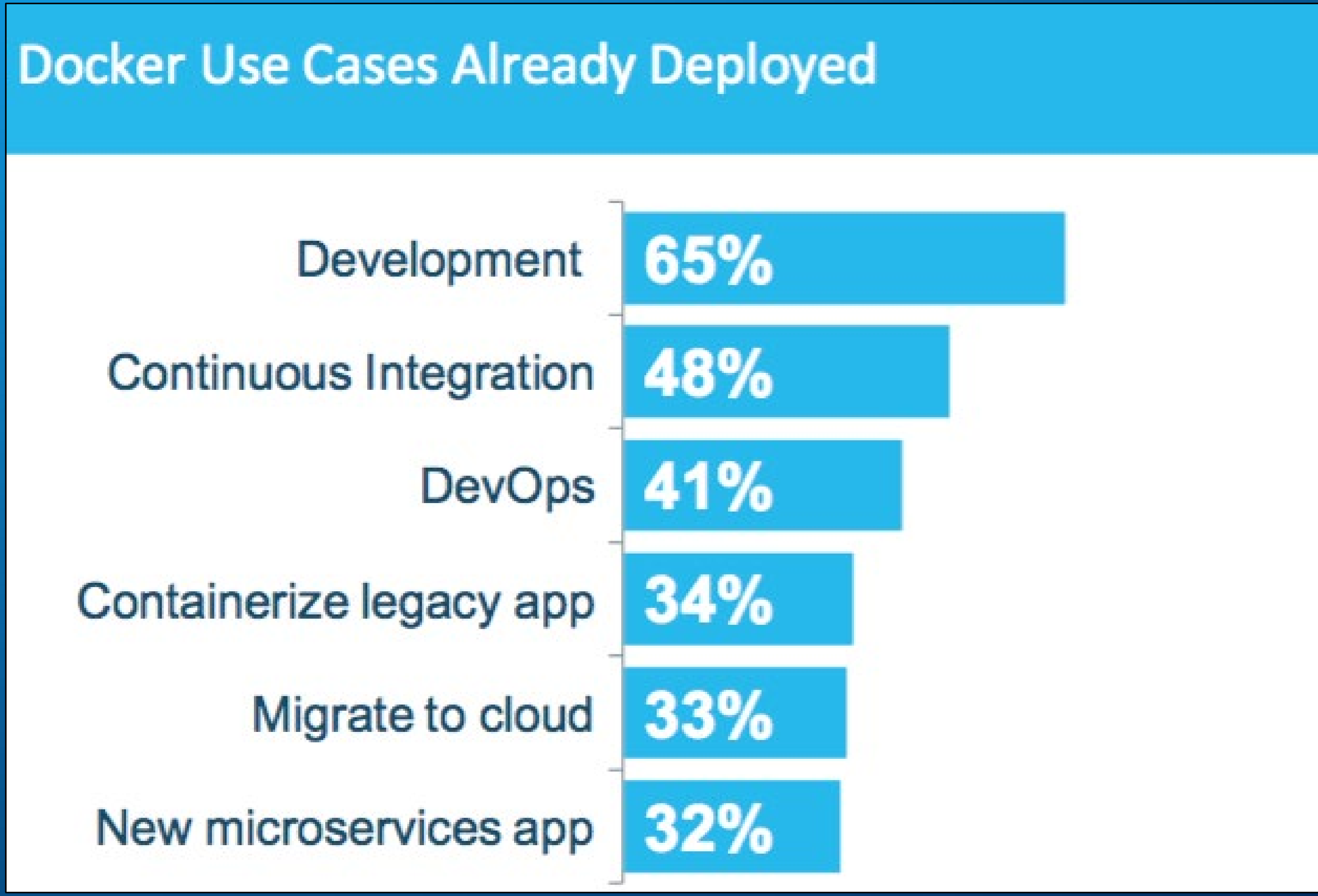
MONITORING CHEAT SHEET

blog.bytebytego.com

Element	aws	Google Cloud	Azure	Open Source / 3rd Party
Data Collection	Cloud Watch Cloud Watch Logs Cloud Trail Config Custom agents / Scripts	Cloud Monitoring Cloud Logging Cloud Audit Logs Custom agents / Scripts	Azure Monitor Azure Activity Log Azure Policy Security Center Custom agents / Scripts	ZABBIX Prometheus fluentd logstash splunk> ELK telegraf Nagios Sensu
Data Storage	S3	Cloud Storage	Blob Storage	MINIO GLUSTER ceph
Data Analysis	CloudWatch Metrics Insights	Cloud Operations	Azure Monitor Metrics Explorer	Grafana tableau kibana
Alerting	SNS	Cloud Monitoring Alerts	Azure Monitor Alerts	PagerDuty slack
Visualization	CloudWatch Dashboard QuickSight	Cloud Monitoring Dashboard Data Studio	Azure Monitor Dashboard Power BI	Grafana Superset Metabase redash tableau
Reporting and Compliance	Config Rules Trusted Advisor	Security Command Center	Policy Compliance Security Center Compliance	OpenSCAP CISOfy
Automation	Lambda Step Functions	Cloud Functions	Azure Functions Azure Automation	Jenkins ANSIBLE
Integration	CloudFormation CodePipeline	Cloud Deployment Manager Cloud Build	Azure Automation Azure DevOps	Pulumi Terraform ANSIBLE Jenkins GitLab Travis CI
Feedback Loop	Well-Architected Tool	Well-Architected Framework	Well-Architected Framework	Scout APM Cloud Custodian

How Containers are Being Used?

- Developer productivity a top use case today
- Building out CI/CD pipelines
- Consistent container image moves through pipeline
- Preventing “it worked in dev” syndrome
- Application modernization and portability are also key adoption drivers (Prem <-> cloud)



STU FIIT

Source: THE EVOLUTION OF THE MODERN SOFTWARE SUPPLY CHAIN, DOCKER SURVEY 2016

Why Containers?

Why developers care for containers?

- Quickly create ready-to-run packaged applications, low cost deployment and replay
- Automate testing, integration, packaging
- Reduce / eliminate platform compatibility issues
- Support next gen applications (microservices)

Why management cares?

- Improves **speed** and frequency of releases, reliability of deployments
- Makes app lifecycle efficient, consistent and repeatable – configure once, run many times
- Eliminate environment inconsistencies between development, test, production
- Improve production application resiliency and scale out / in on demand

Good Use Cases for Containers

Ready to Run Application Stacks

- Excellent for Dev/Test setups
- Deployment in Seconds, not Hours/Days
- Start Up, Tear Down Quickly

New App Dev & Microservices

- Refactor all or part of legacy app
- Containers are great for Microservices

One-Time Run Jobs and Analytics

- Run the Job / Analysis and quit

Front-End App Servers

- Highly horizontally scalable
- Fast A/B
- Rolling Deployments
- Traditional Technologies - Backend

Server Density

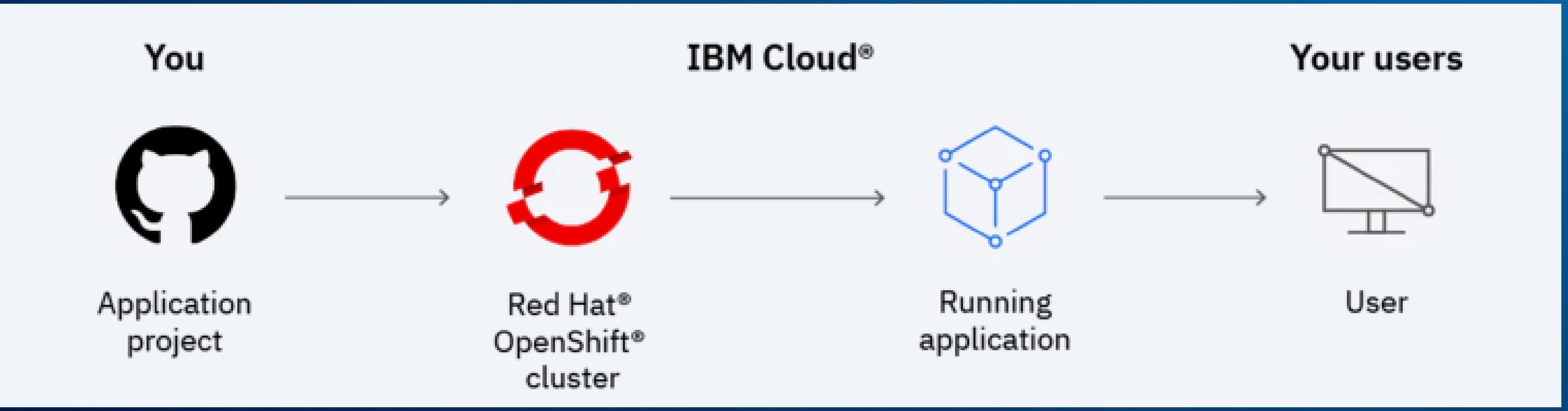
- Containers can use dynamic ports
- Run many of the same app on a server
  - instead of one per VM

OpenShift

- A layer called OpenShift can be added to Docker and Kubernetes to make it simpler and more accessible for developers to build apps

https://www.ibm.com/products/openshift

OPENSHIFT® by Red Hat®



- Web-based Console
- Command-Line Tool
- Logs and metrics
- Templates

OpenShift goal → ready-for-production and scaling

Self - Service

Multi - Language

Automation

Collaboration

Multi - Tenant

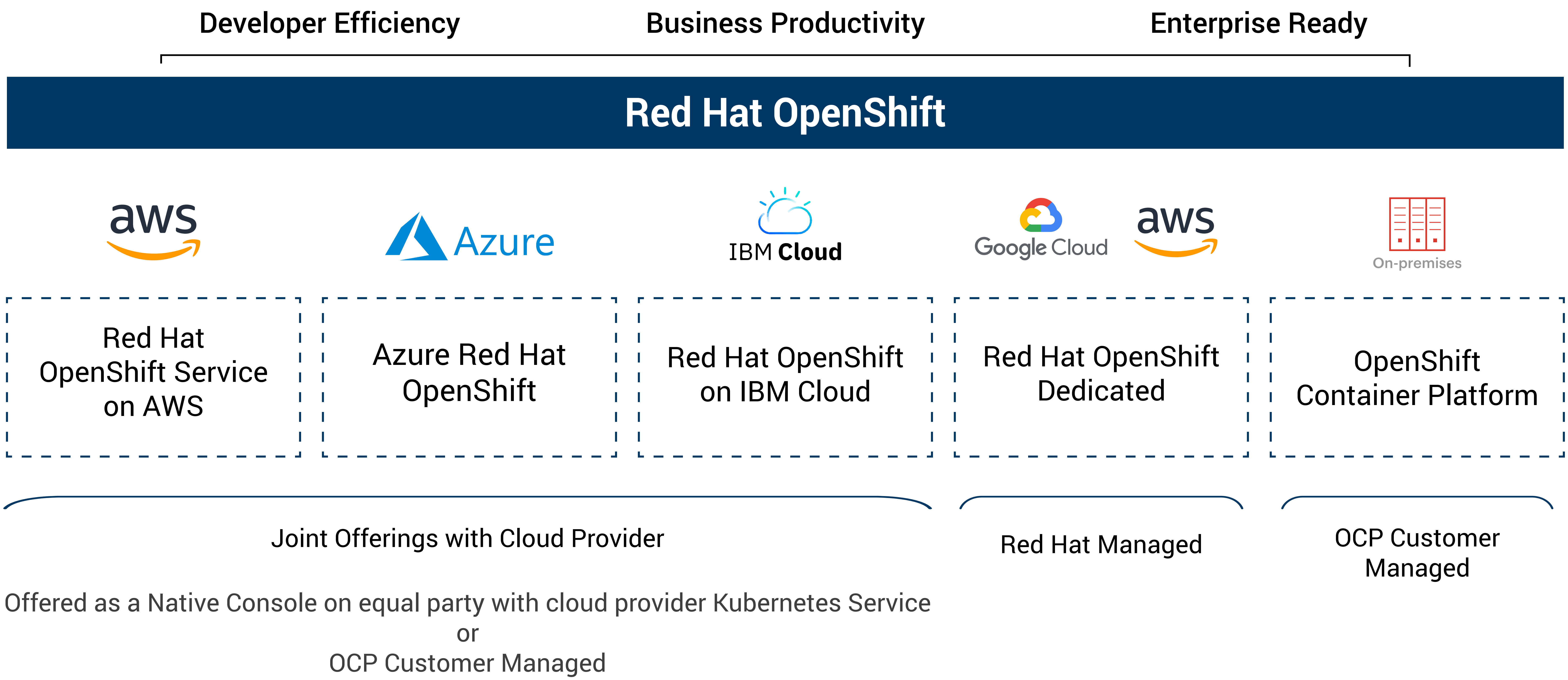
Standards - Based

Web - Scale

Open Source

Enterprise Grade

Secure



Developer Efficiency

Business Productivity

Enterprise Ready

Red Hat OpenShift

Red Hat OpenShift Service on AWS

Azure Red Hat OpenShift

Red Hat OpenShift on IBM Cloud

Red Hat OpenShift Dedicated

OpenShift Container Platform

Joint Offerings with Cloud Provider

Red Hat Managed

OCP Customer Managed

Offered as a Native Console on equal party with cloud provider Kubernetes Service
or
OCP Customer Managed

Cloud environment and testing for students

- Azure Dev Tools for Teaching
- https://aka.ms/devtoolsforteaching

- IBM Cloud free tier
- https://www.ibm.com/cloud/free

- Oracle Cloud Free Tier
- https://www.oracle.com/sk/cloud/free/

QUESTIONS?

STU FIIT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS
AND INFORMATION TECHNOLOGIES

Literature

1. https://www.ibm.com/topics/containerization
2. https://docs.docker.com
3. https://www.royalcyber.com/technologies/red-hat-openshift/

Software Architecture

Coffee Break

Mgr. Pavle Dakić, PhD. student

pavle.dakic@stuba.sk

Software Architecture

Exercises

Mgr. Pavle Dakić, PhD. student

pavle.dakic@stuba.sk

STU FIIT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA
FACULTY OF INFORMATICS AND INFORMATION TECHNOLOGIES

```
document
catch(e){_DumpExcep
for ( c = 0 ; c < argc ; c)
printf(d. Command line argument passed
return 0;
include <stdio>
main(int argc, char arg)
{
print("Data decoding");
scan("d", n);
rray");
```

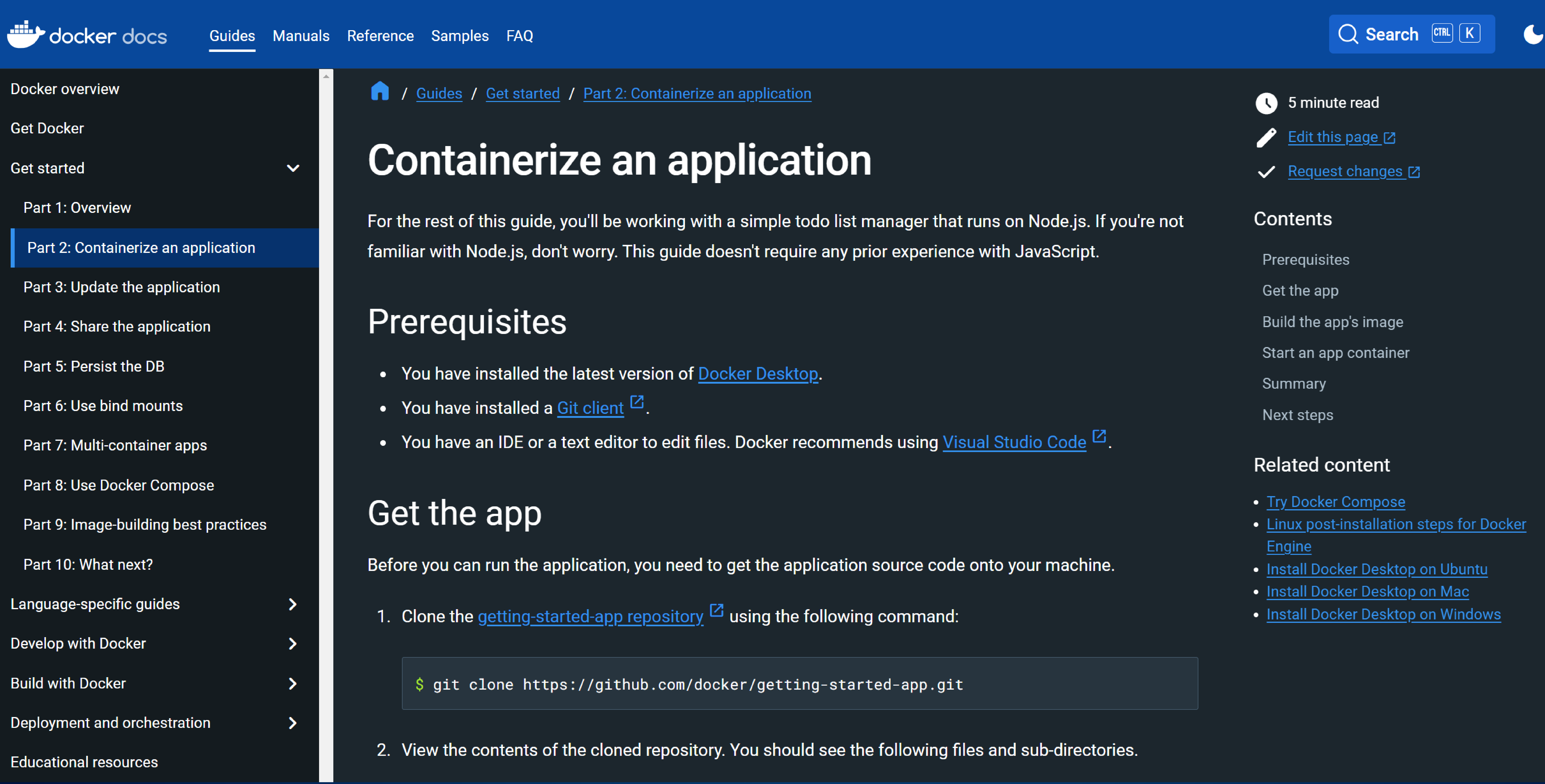
Basic Docker commands

Bash - terminal

```
docker image pull node:latest
docker image ls
docker container run –d –p 5000:5000 –-name node node:latest
docker container ps
docker container stop node(or <container id>)
docker container rm node (or <container id>)
docker image rmi (or <image id>)
docker build –t node:2.0 .
docker image push node:2.0
docker --help
```

List Docker networks

```
docker network ls
docker network inspect bridge
```



Containerize an application

For the rest of this guide, you'll be working with a simple todo list manager that runs on Node.js. If you're not familiar with Node.js, don't worry. This guide doesn't require any prior experience with JavaScript.

Prerequisites

- You have installed the latest version of Docker Desktop.
- You have installed a Git client.
- You have an IDE or a text editor to edit files. Docker recommends using Visual Studio Code.

Get the app

Before you can run the application, you need to get the application source code onto your machine.

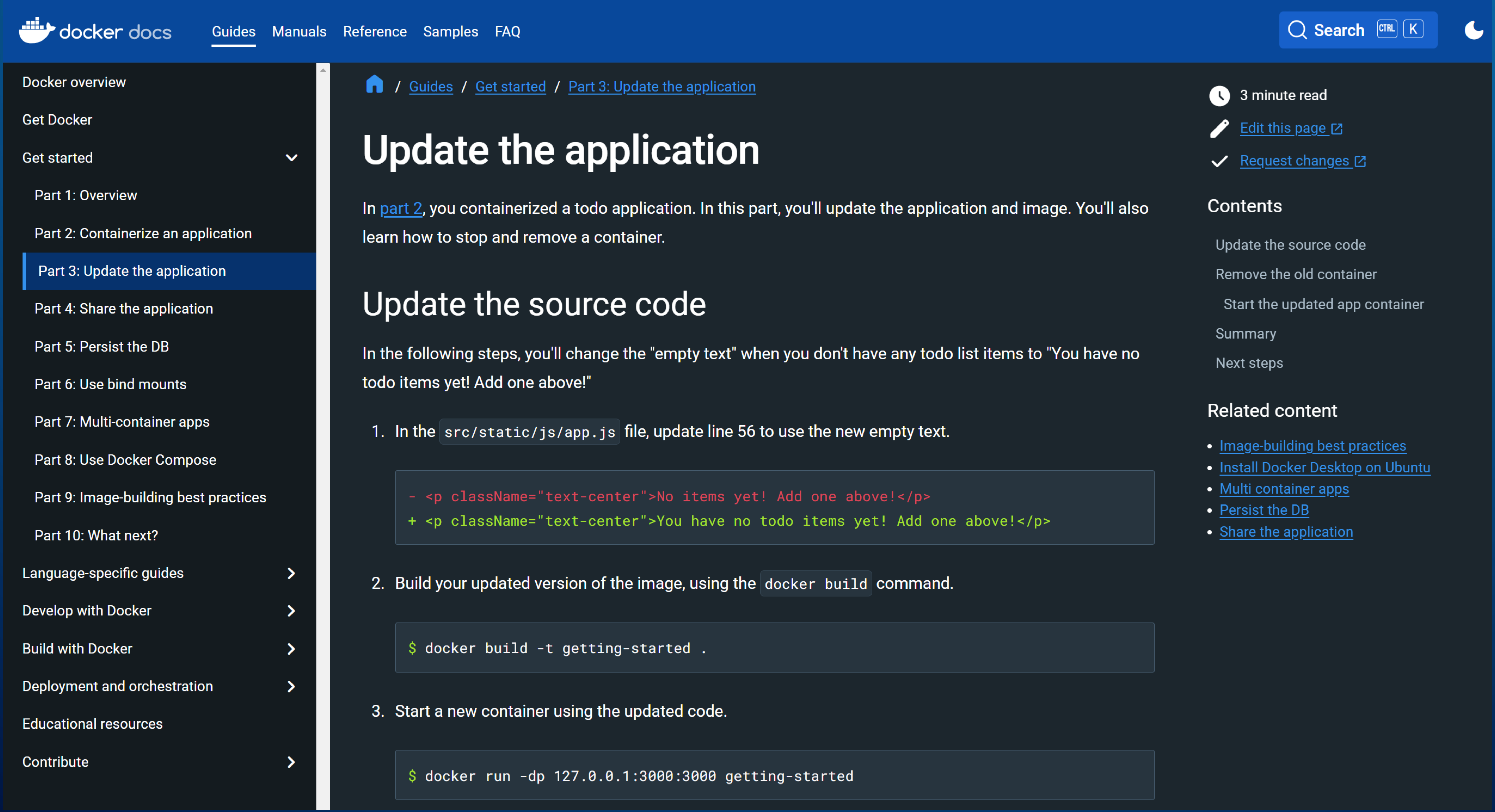
1. Clone the getting-started-app repository using the following command:

```
$ git clone https://github.com/docker/getting-started-app.git
```

2. View the contents of the cloned repository. You should see the following files and sub-directories.

STU FIIT

https://docs.docker.com/get-started/02_our_app/



Update the application

In part 2, you containerized a todo application. In this part, you'll update the application and image. You'll also learn how to stop and remove a container.

Update the source code

In the following steps, you'll change the "empty text" when you don't have any todo list items to "You have no todo items yet! Add one above!"

1. In the `src/static/js/app.js` file, update line 56 to use the new empty text.

```
- <p className="text-center">No items yet! Add one above!</p>
+ <p className="text-center">You have no todo items yet! Add one above!</p>
```

2. Build your updated version of the image, using the `docker build` command.

```
$ docker build -t getting-started .
```

3. Start a new container using the updated code.

```
$ docker run -dp 127.0.0.1:3000:3000 getting-started
```

https://docs.docker.com/get-started/03_updating_app/

Portainer

```
docker pull portainer/portainer-ce
docker volume create portainer_data


docker run -d -p 8000:8000 -p 9443:9443 --name portainer
portainer/portainer-ce:latest


docker run -d -p 8000:8000 -p 9443:9443 --name portainer --
restart always -v
\\.\pipe\docker_engine:\\.\pipe\docker_engine -v
portainer_data:C:\data portainer/portainer-ce:latest
```

LiteSpeed

```
docker pull litespeedtech/litespeed:latest
docker pull litespeedtech/openlitespeed:latest

docker run --name litespeed -p 7080:7080 -p 80:80 -p 443:443 -it litespeedtech/litespeed:latest

docker run --name openlitespeed -p 7080:7080 -p 80:80 -p 443:443 -it litespeedtech/openlitespeed:latest

docker ps
```

LiteSpeed #2

```
docker exec -it openlitespeed bash
cat /usr/local/lsws/adminpasswd
/usr/local/lsws/admin/misc/admpass.sh
/usr/local/lsws/conf/vhosts/
```

LiteSpeed #3

```
## Create new vhost
cd /usr/local/lsws
mkdir Example2
mkdir Example2/{conf,html,logs}
chown lsadm:lsadm Example2/conf
```

MySQL

```
docker pull mysql:latest

docker run --name mysql-1 -e MYSQL_ALLOW_EMPTY_PASSWORD=1 -d -p 3306:3306  -p 33060:33060 mysql:latest

docker ps
```

Postgres

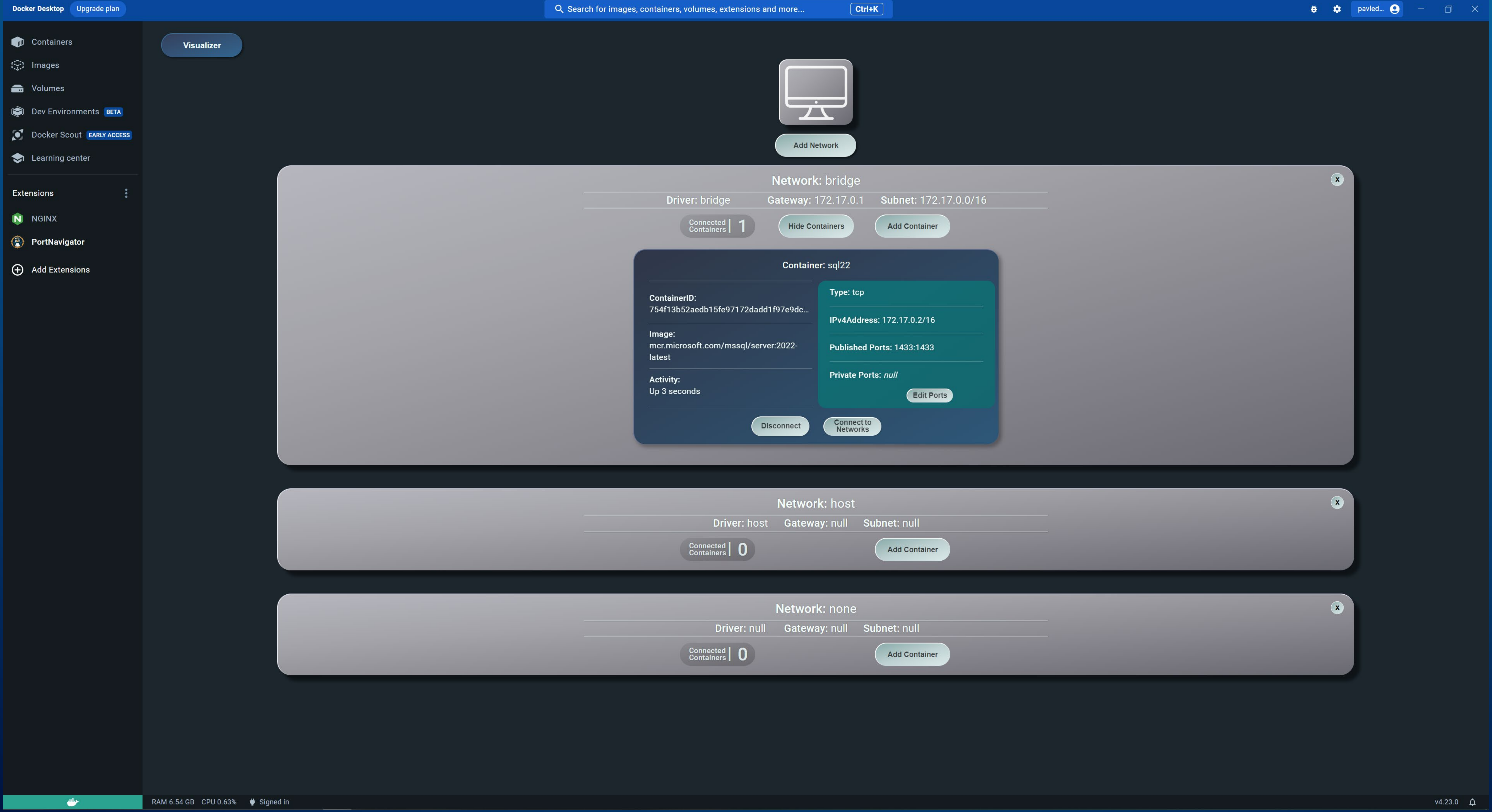
```
docker pull postgres:latest


docker run --name postgres-1 -e
POSTGRES_HOST_AUTH_METHOD=trust -d -p 5432:5432
postgres:latest


docker ps
docker stop
```

MSSQL 2022

```
docker pull mcr.microsoft.com/mssql/server:2022-latest
docker pull mcr.microsoft.com/mssql/server:2019-latest


## SQL22
docker run --name='sql22' --hostname='sql22' -p 1433:1433 --memory='5g' --shm-size='2g' --add-host=host.docker.internal:host-gateway -v sql22data:/var/opt/mssql -e 'MSSQL_AGENT_ENABLED=True' -e 'TZ=Europe/Bratislava' -e 'MSSQL_COLLATION=SQL_Slovenian_CP1250_CI_AS' -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=yourStrong(!)Password' -d mcr.microsoft.com/mssql/server:2022-latest
```

MSSQL 2022 #2

```
## Connect to network
docker network connect --alias sql22 mynet sql22
docker exec -it --add-host=host.docker.internal:host-gateway
sql22 bash
## Disconnect from network
docker network disconnect mynet sql22

## Read Config for Docker container
docker inspect sql22

## Root Bash access
sudo docker exec -it --user root sql22 bash
```



Network: bridge

Driver: bridge Gateway: 172.17.0.1 Subnet: 172.17.0.0/16

Connected Containers | 1

Container: sql22

ContainerID: 754f13b52aedb15fe97172dadd1f97e9dc...

Image: mcr.microsoft.com/mssql/server:2022-latest

Activity: Up 3 seconds

Type: tcp

IPv4Address: 172.17.0.2/16

Published Ports: 1433:1433

Private Ports: *null*

Network: host

Driver: host Gateway: null Subnet: null

Connected Containers | 0

Network: none

Driver: null Gateway: null Subnet: null

Connected Containers | 0

MSSQL 2019

```
## SQL19
docker run --name sql19 --hostname sql19 -p 1433:1433 --memory='5g' --shm-size='5g' -e 'MSSQL_AGENT_ENABLED=True' -e 'TZ=Europe/Bratislava' -e 'MSSQL_COLLATION=SQL_Slovenian_CP1250_CI_AS' -e 'ACCEPT_EULA=Y' -e 'MSSQL_SA_PASSWORD=yourStrong(!)Password' -d mcr.microsoft.com/mssql/server:2019-latest
```