Container services

Module overview

- Part 1: Introduction to containers
- Part 2: Containers vs. hardware virtualization
- Part 3: Microservices: Use case for containers
- Part 4: Amazon container orchestration services

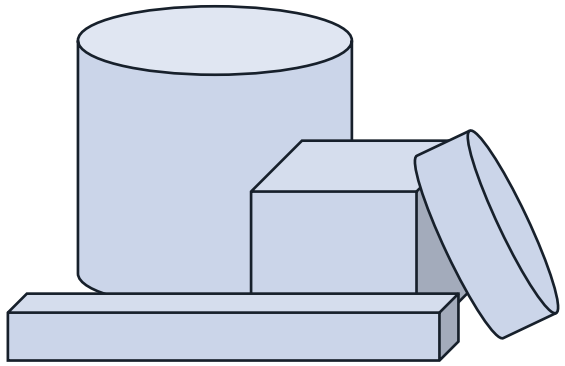
Lab

- Working with Docker containers

Shipping with containers

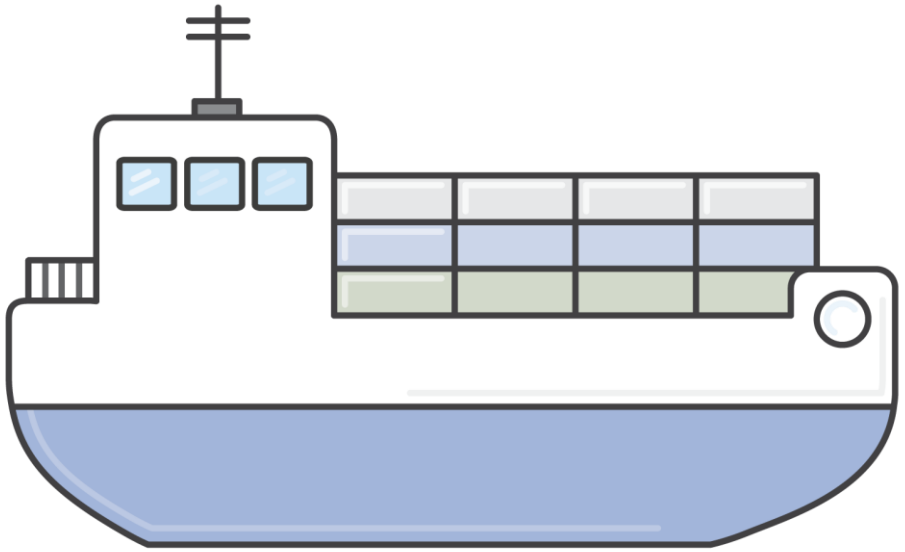
Before shipping containers:

- Goods were shipped in a variety of vessels with no standardized weight, shape, or size.
- Transporting goods was slow, inefficient, and costly.



After shipping containers:

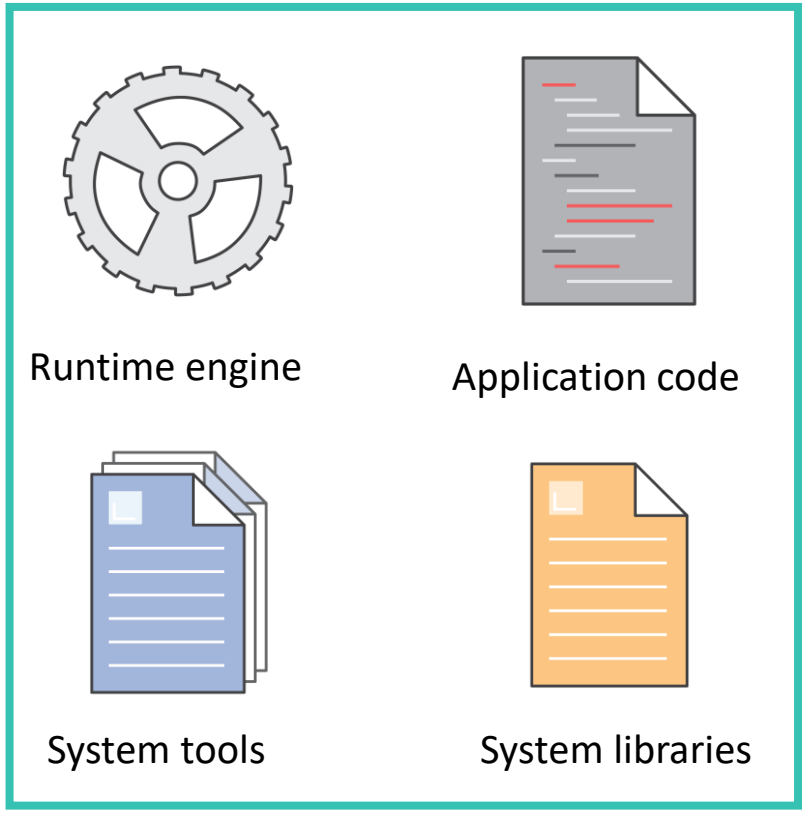
- Uniform size of shipping containers made it more efficient to load, unload, and stack.
- Containers improved efficiency, increased productivity, and reduced costs.



Container ship

What is a container?

A standardized unit of software

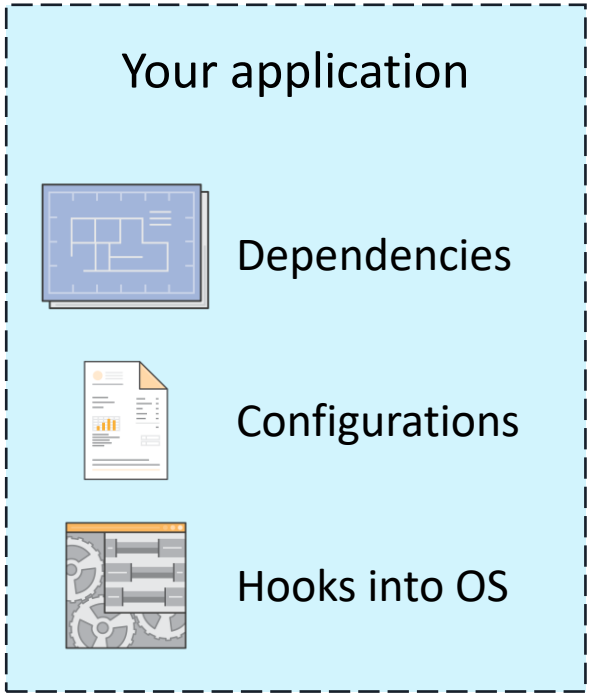


Container basics

- **Containers** are a method of operating system virtualization.

- Benefits –
  - Repeatable.
  - Self-contained execution environments.
  - Software runs the same in different environments.
    - Developer's laptop, test, production.
  - Faster to launch and stop or terminate than virtual machines

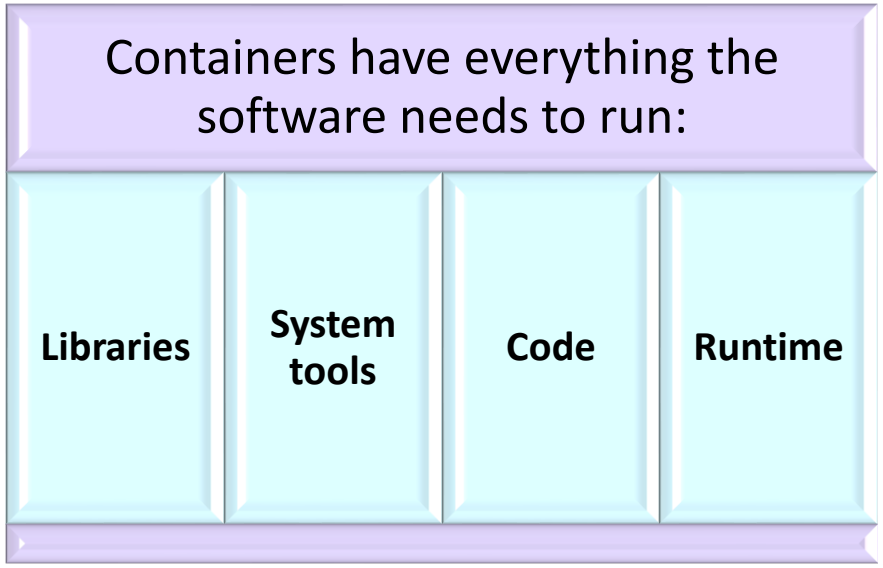
Your Container



What is Docker?

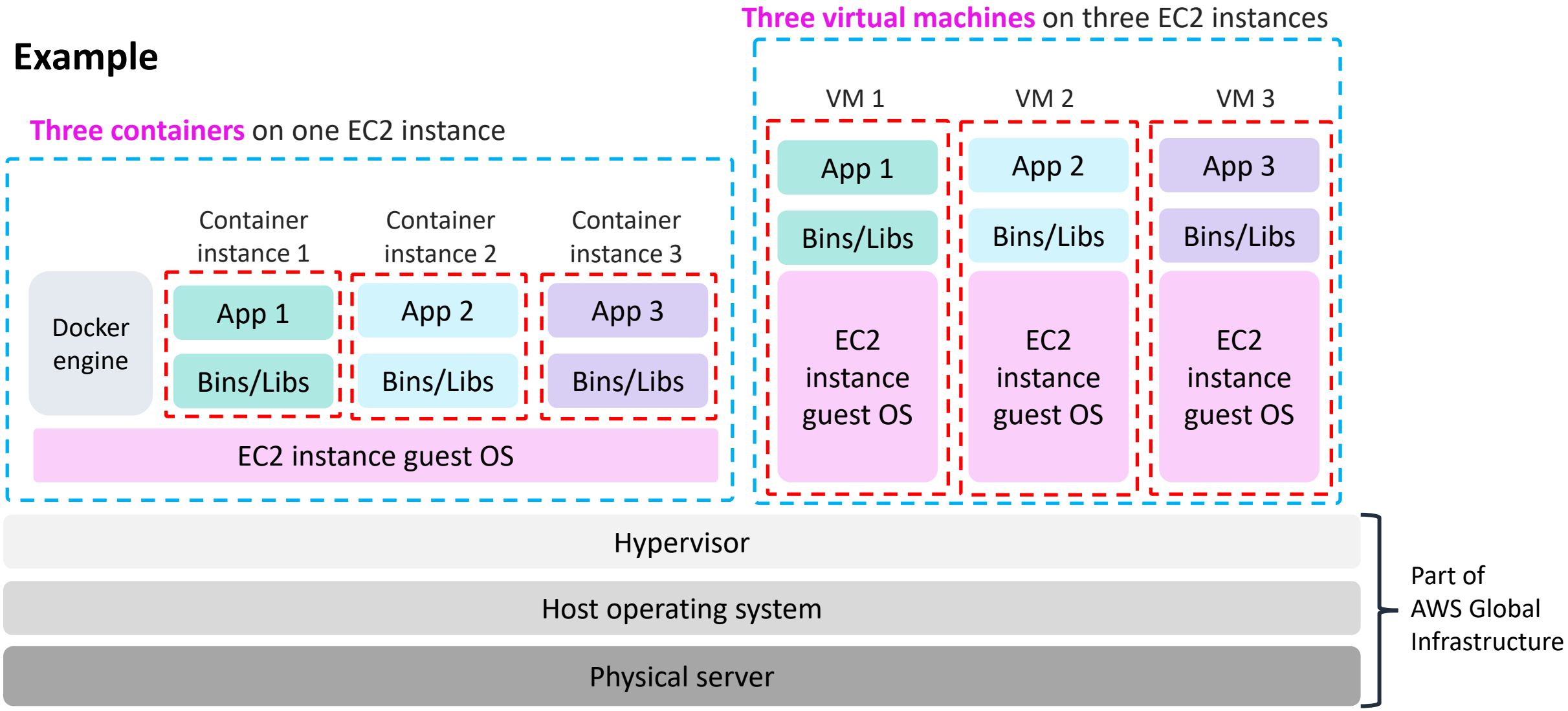
- **Docker** is a software platform that enables you to build, test, and deploy applications quickly.
- You run containers on Docker.
  - Containers are created from a template called an *image*.
- A **container** has everything a software application needs to run.

Container



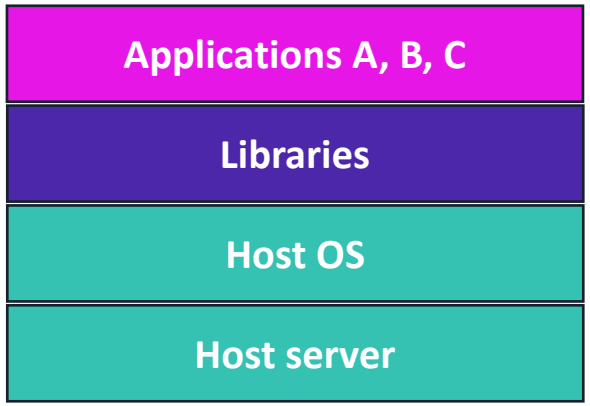
Containers versus virtual machines

Example



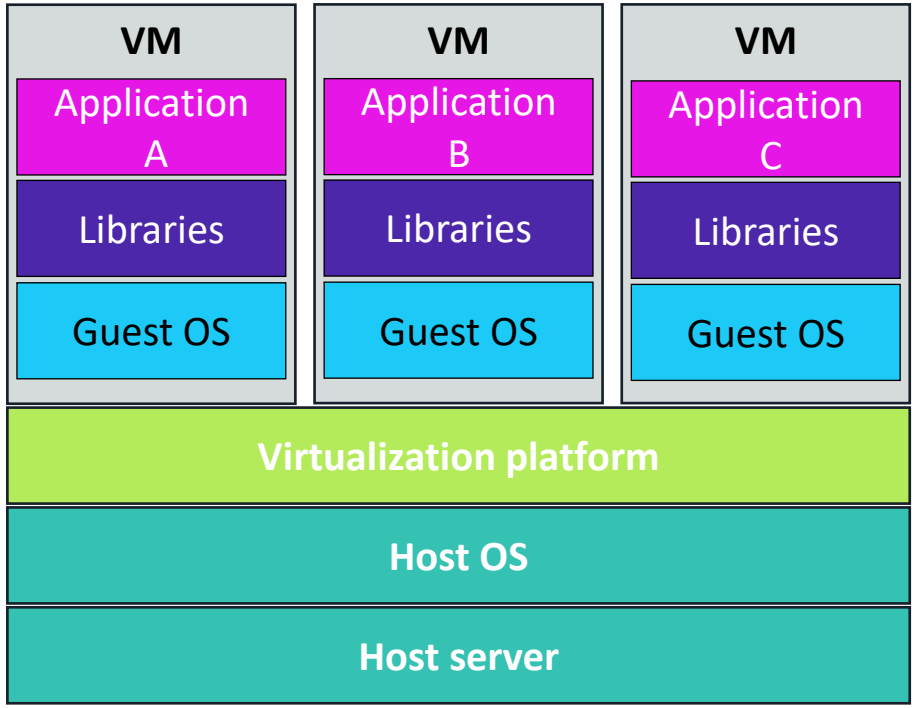
Evolution of application deployment models

1960 – 1998



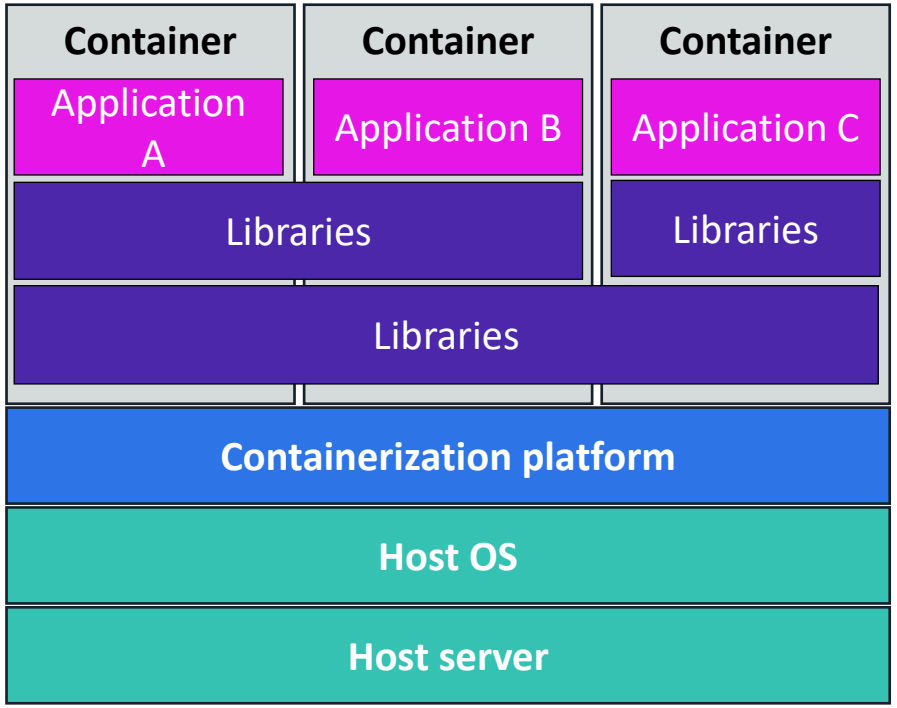
Bare-Metal Servers

1998 – 2010



Virtual Machines

2010 – current



Containers

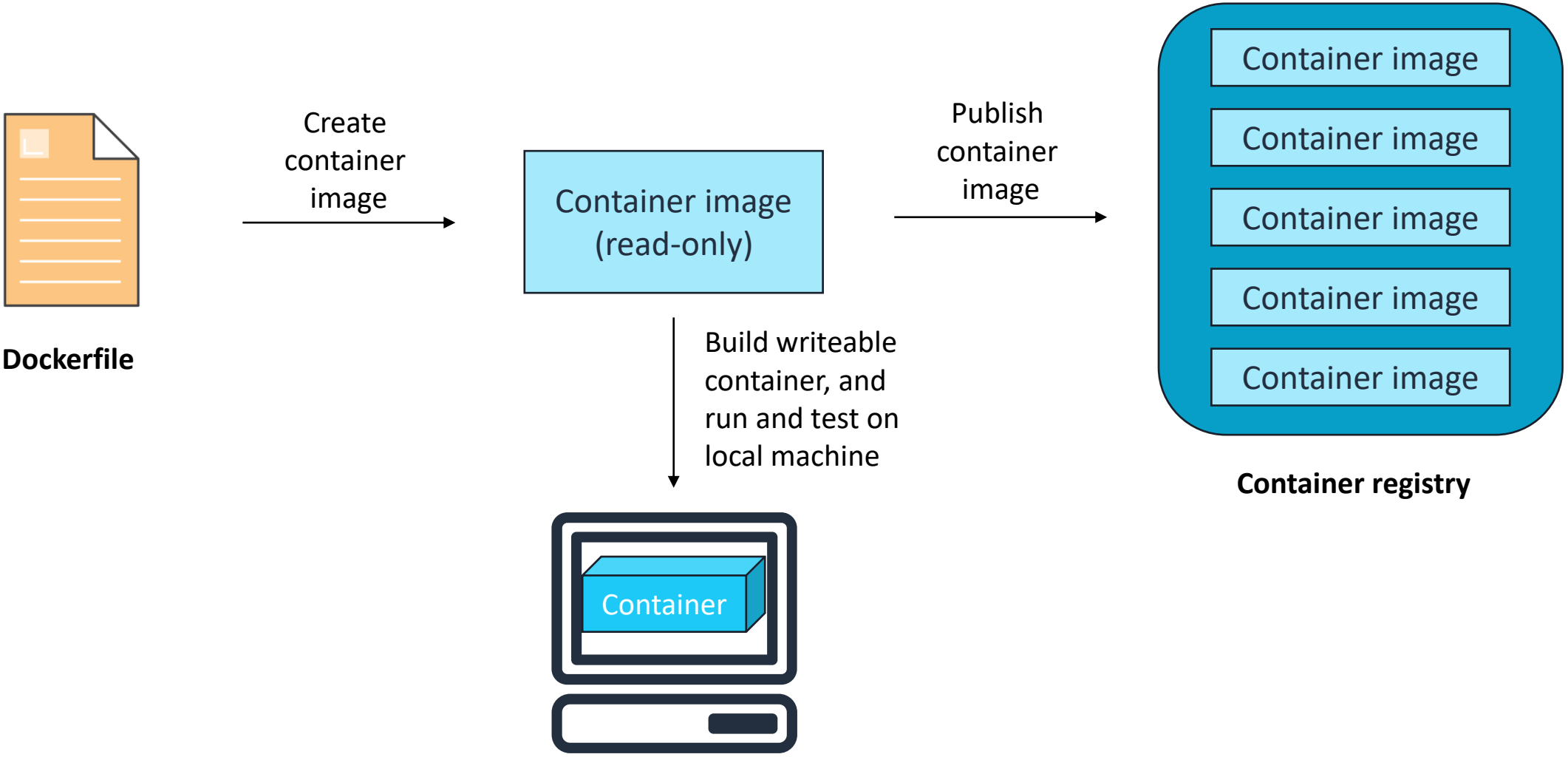
Docker as a virtualization platform

- Is a lightweight container virtualization platform
- Provides tools to create, store, manage, and run containers
- Integrates with automated build, test, and deployment pipelines

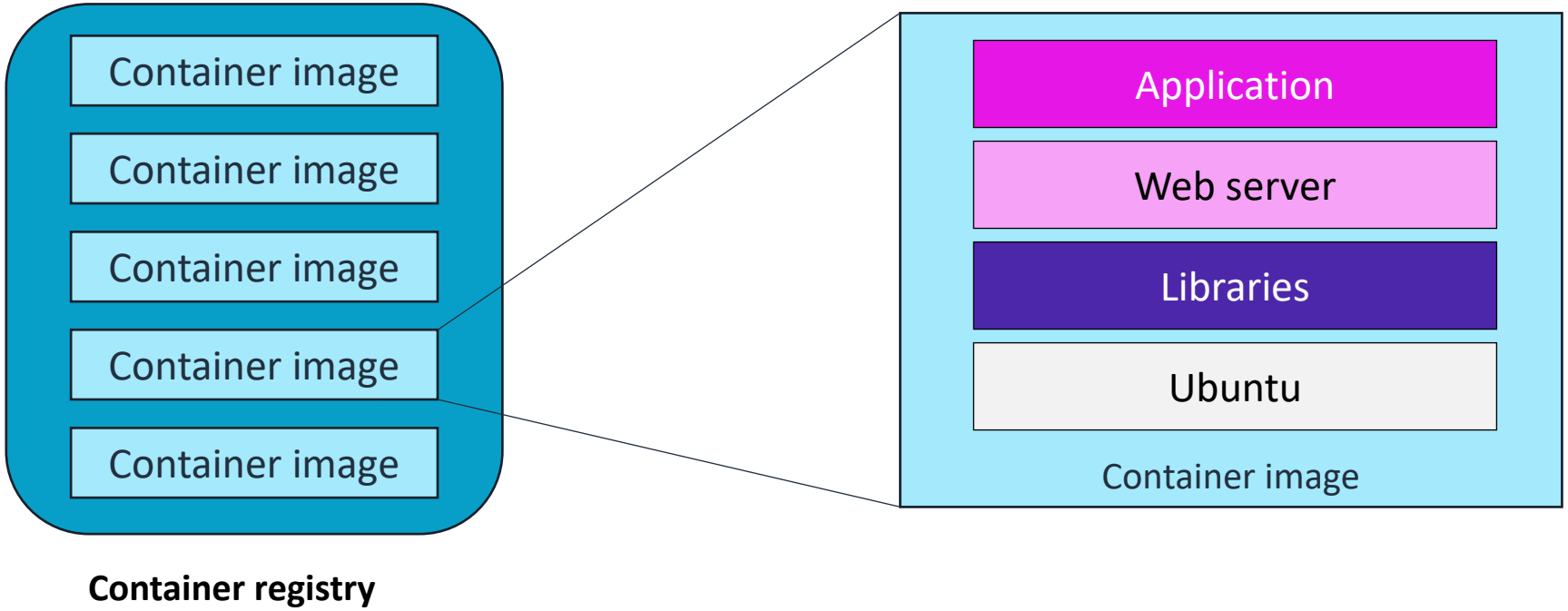
Docker container benefits

- Portable runtime application environment
- Application and dependencies can be packaged in a single, immutable artifact
- Ability to run different application versions with different dependencies simultaneously
- Faster development and deployment cycles
- Better resource utilization and efficiency

Docker container terminology



Images: Templates for containers



Dockerfile example 1

```
# Start with the Ubuntu latest image
FROM ubuntu:latest

# Output hello world message
CMD echo "Hello World!"
```

Dockerfile example 2

```
# Start with open JDK version 8 image
FROM openjdk:8

# Copy the jar file that contains your code from your
system to the container
COPY /hello.jar /usr/src/hello.jar

# Call Java to run your code
CMD java -cp /usr/src/hello.jar
Org.example.App
```

Dockerfile example 3

```
# Start with CentOS7 image
FROM centos:7

# Update the OS and install Apache
RUN yum -y update && yum -y install httpd

# Expose Port 80—the port that the web server "listens to"
EXPOSE Port 80

# Copy shell script and give it executable permissions
ADD run-httpd.sh /run-httpd.sh
RUN chmod -v +x /run-httpd.sh

# Run shell script
CMD ["/run-httpd.sh"]
```

Dockerfile example 3

```
# Start with CentOS7 image
FROM centos:7

# Update the OS and install Apache
RUN yum -y update && yum -y install httpd

# Expose Port 80
EXPOSE Port 80

# Copy shell script and give it executable permissions
ADD run-httpd.sh /run-httpd.sh
RUN chmod -v +x /run-httpd.sh

CMD [“/run-httpd.sh”]
```

Image layers (read-only)

Each line of the Dockerfile adds a layer to the image.

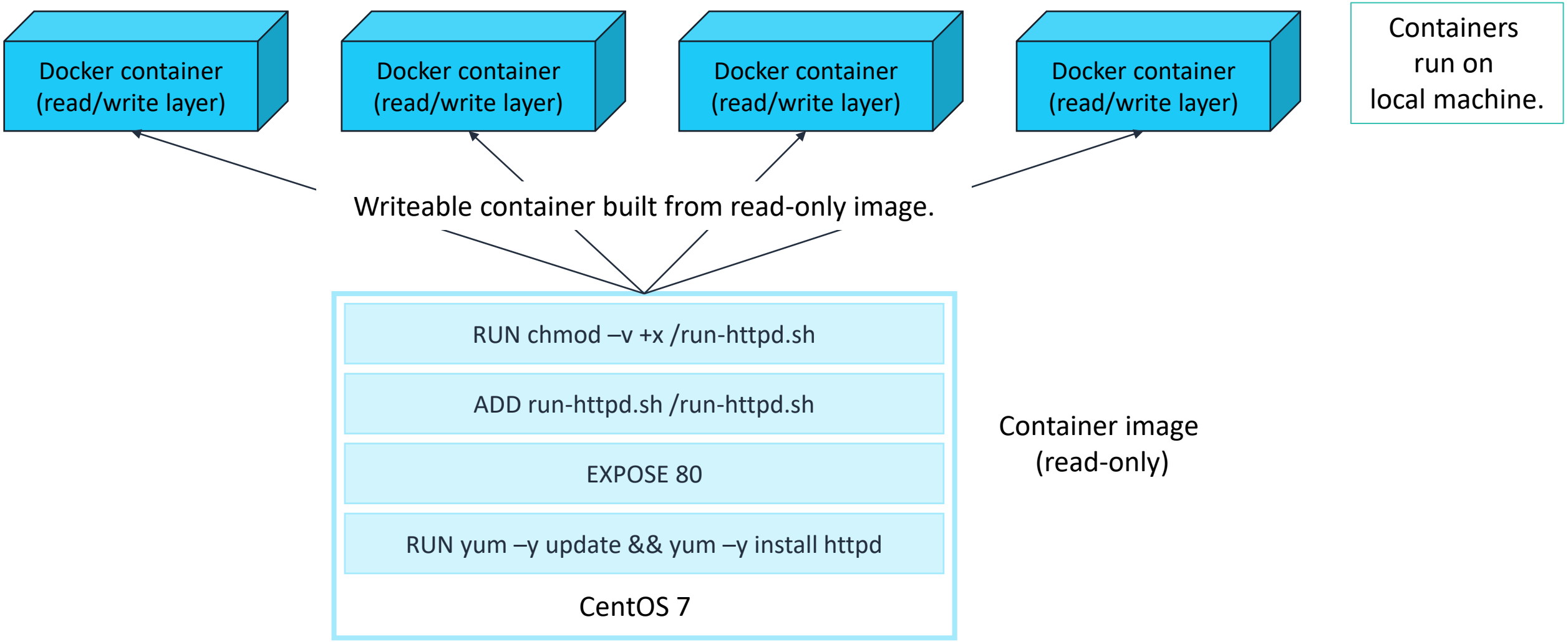
Docker CLI commands

Command	Information
`docker build` *	Build an image from a Dockerfile.
`docker images`	List images on Docker host.
`docker run`	Run an image.
`docker ps` *	List running containers.
`docker exec`	Run a command in a container.
`docker stop`	Stop a running container.

Command	Information
`docker start`	Start a container.
`docker logs`	View container log output.
`docker port`	List container port mappings.
`docker tag` *	Tag an image.
`docker push` *	Push image to a registry.
`docker inspect`	Inspect container information.

* *indicates most common Docker commands.*

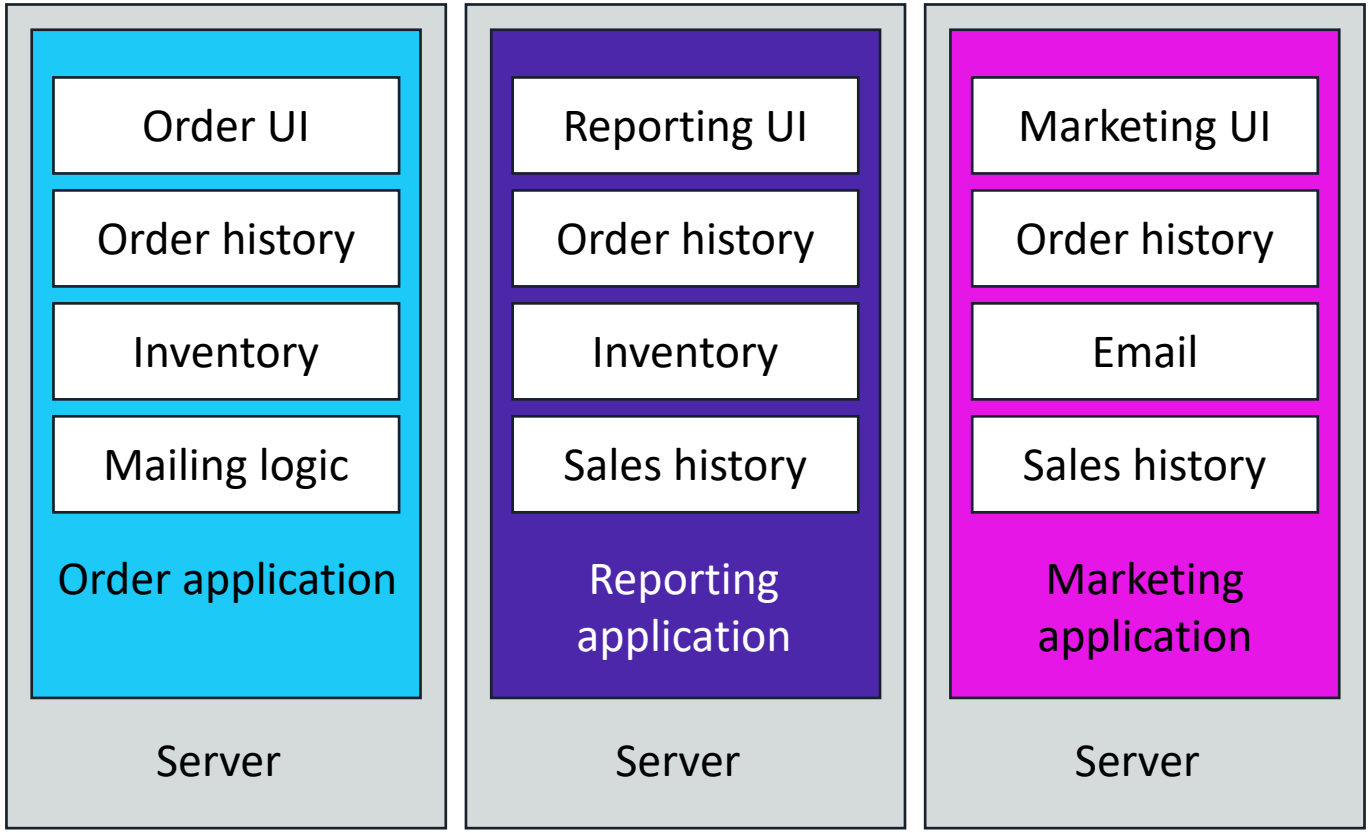
Summary: Docker images vs. containers



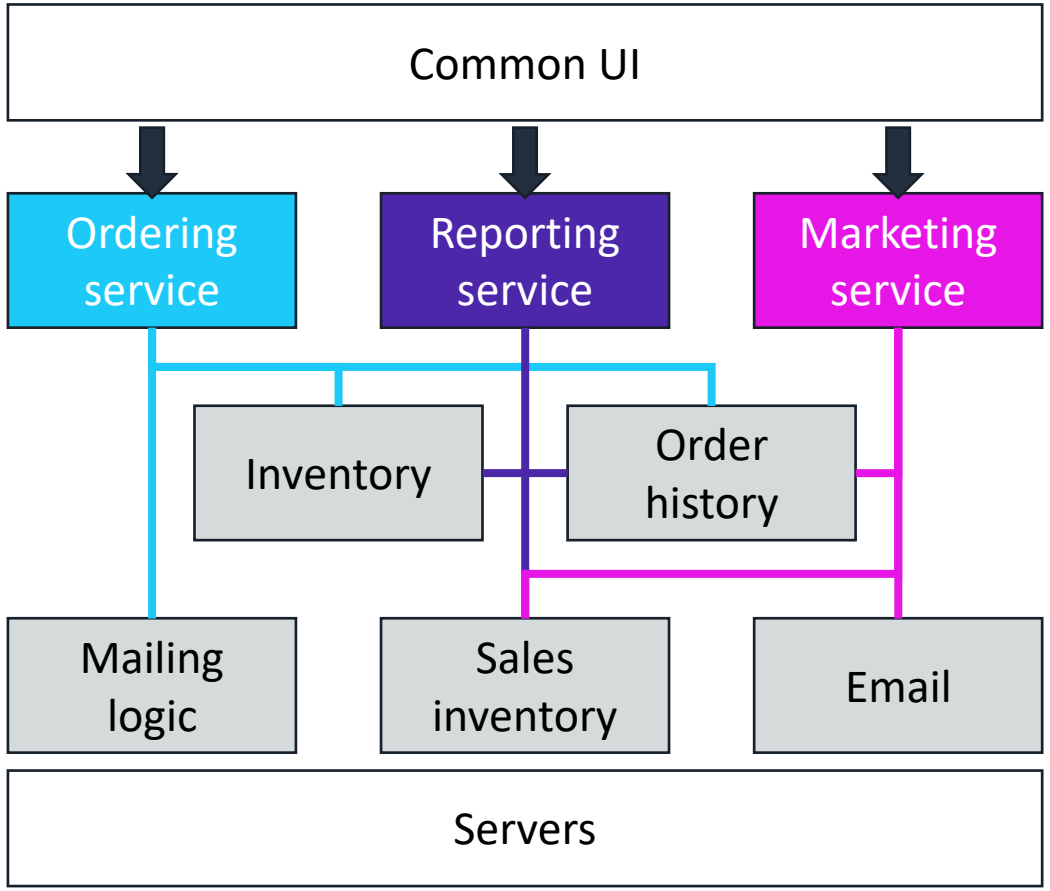
Part 3: Microservices – Use case for containers

Monolithic vs. microservice architecture

Monolithic Architecture



Microservice Architecture

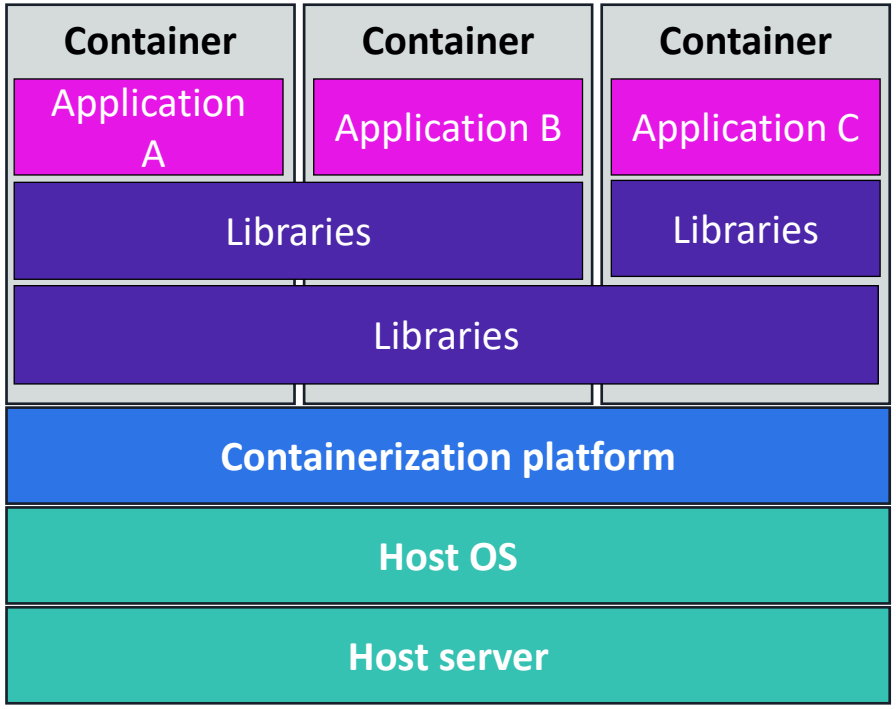


Characteristics of microservices

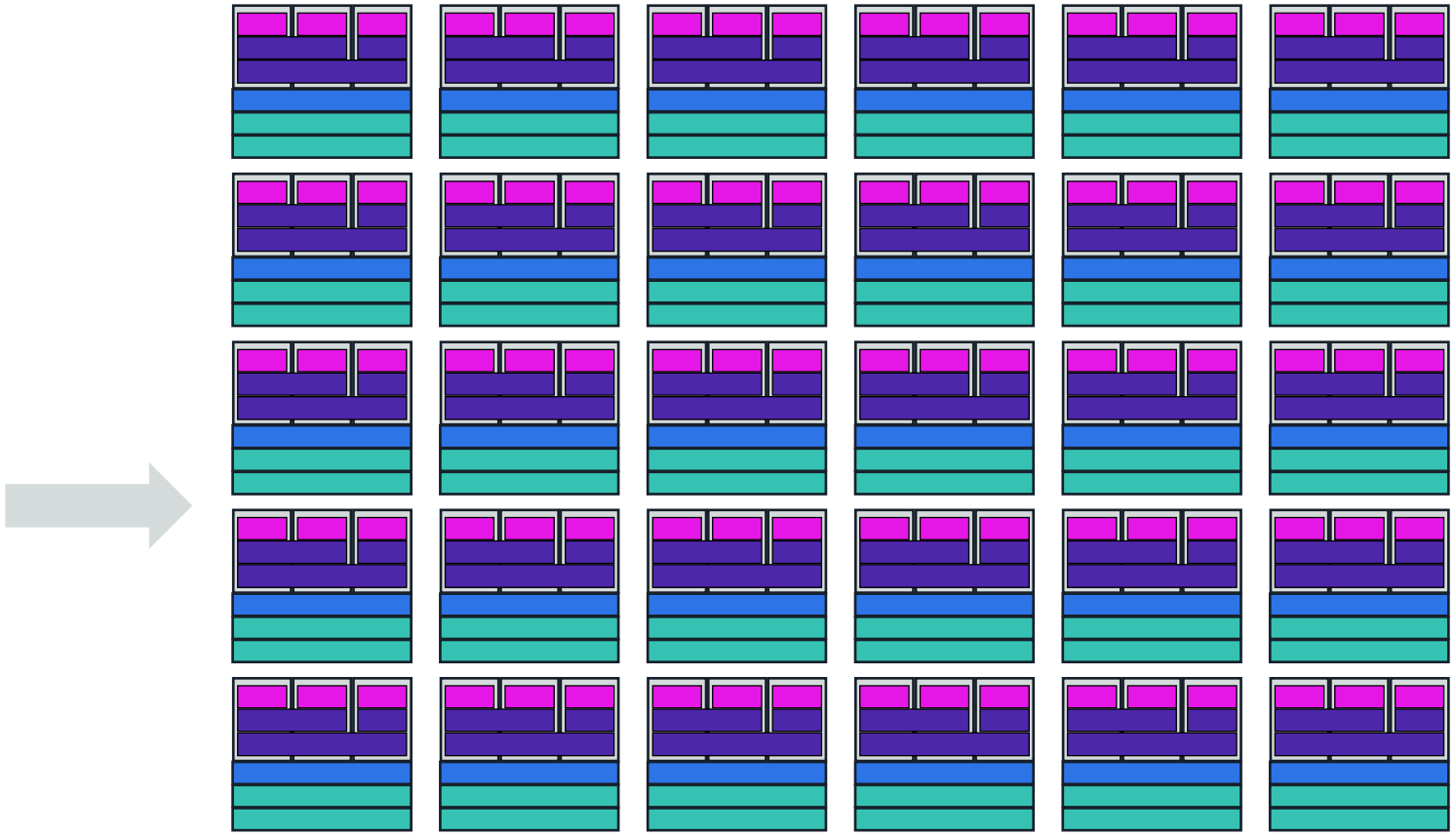
- Decentralized, evolutionary design
- Smart endpoints, dumb pipes
- Independent products, not projects
- Designed for failure
- Disposable
- Development and production parity

Part 4: Amazon container services

The challenge with managing containers

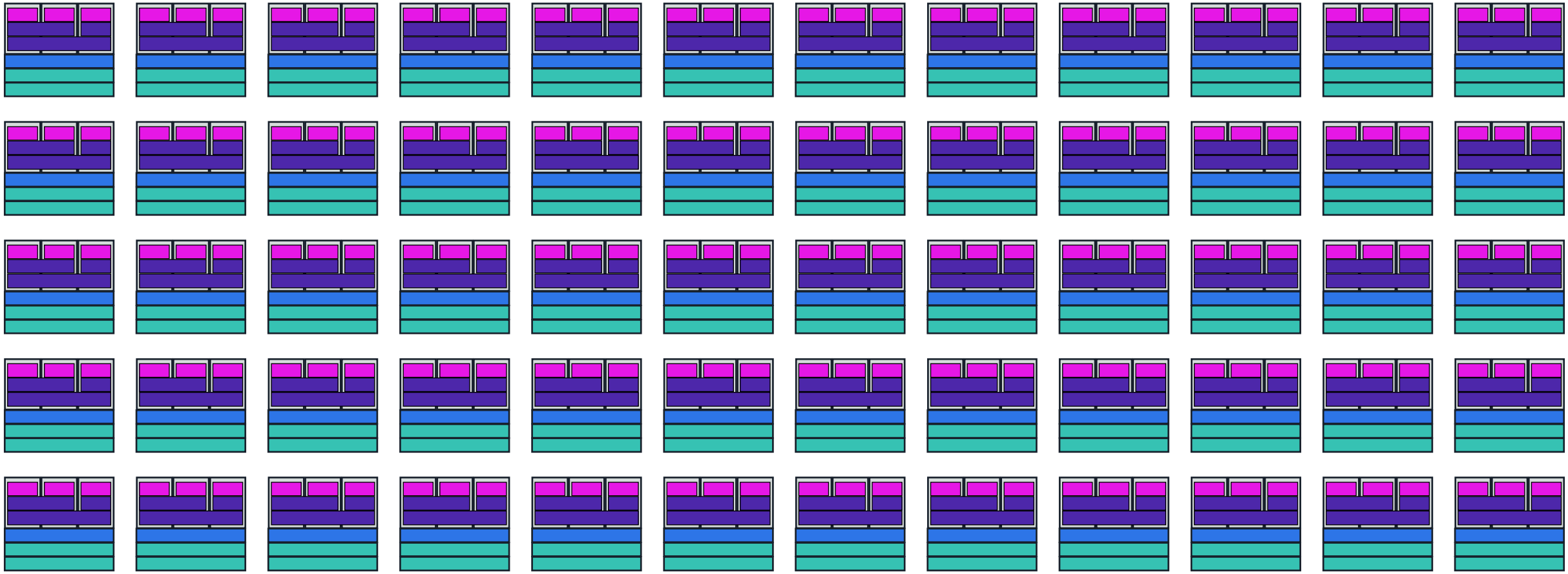


One host with multiple containers



Tens of hosts with hundreds of containers

The challenge with managing containers



Hundreds of hosts with thousands of containers

Container management platforms

- Scheduling and placement
- Service integration

Amazon ECS

Docker swarm

Kubernetes

Amazon Elastic Container Service

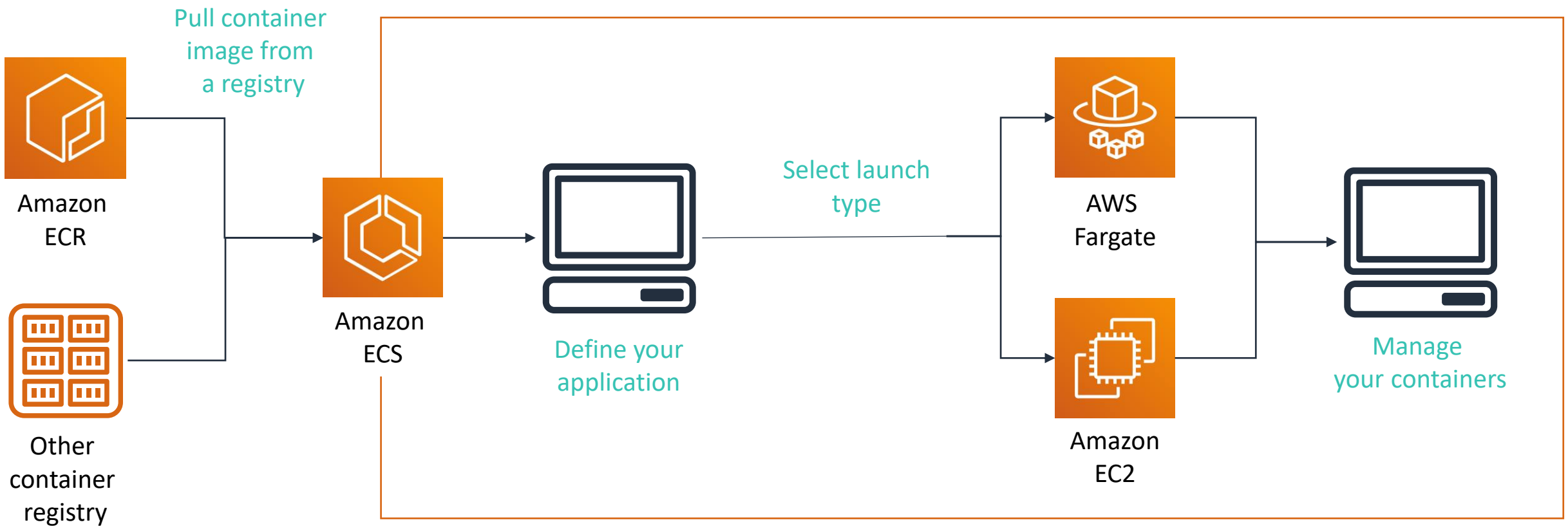
Highly scalable, highly performing container orchestration service

Amazon ECS

Schedules placement across managed clusters

Integrates with third-party schedulers and other AWS services

Amazon ECS solution architecture



Amazon Elastic Container Service for Kubernetes

Managed service that runs Kubernetes on the AWS Cloud

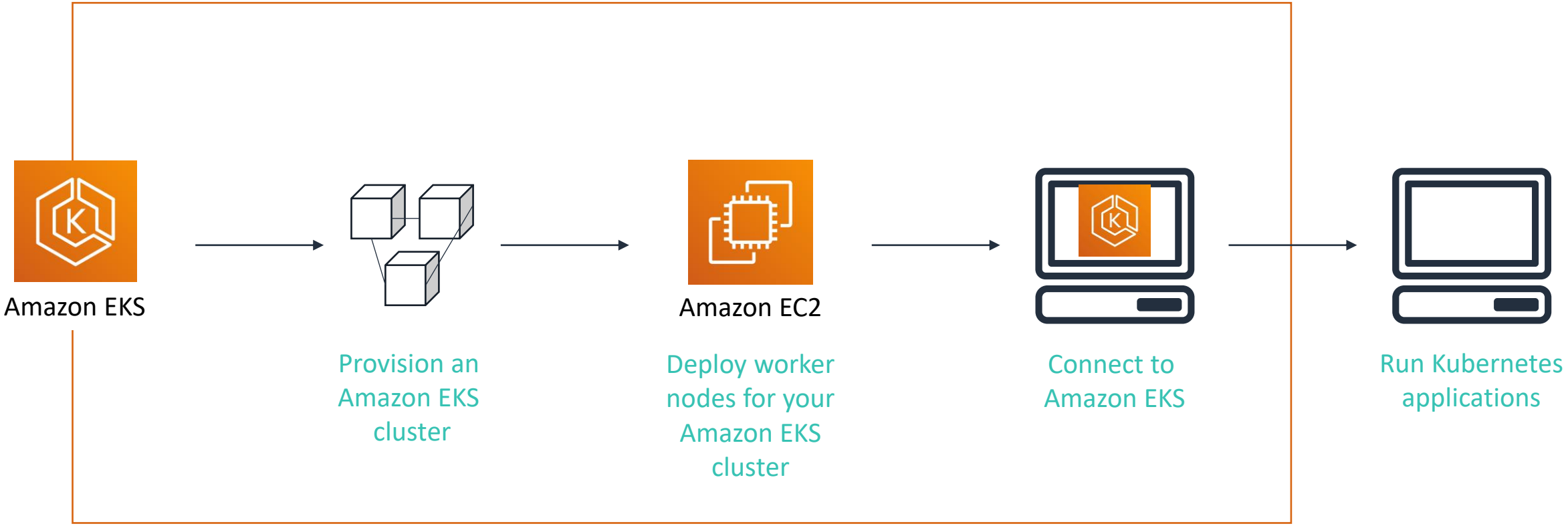
Conformant and compatible

Amazon EKS

Secure by default

Built with the Kubernetes community

Amazon Elastic Container Service for Kubernetes



Fully managed,
cloud-based Docker image registry

Integrated with
Amazon ECS and
Docker CLI

Amazon ECR

Scalable and highly available

Secure:
- Encryption at rest
- Integration with IAM

Creating an Amazon ECR repository and pushing an image

```
# Create a repository called hello-world
> aws ecr create-repository --repository-name hello-world

# Build and tag an image
> docker build -t hello-world .
> docker tag hello-world aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world

# Authenticate Docker to your Amazon ECR registry
> aws ecr get-login
> docker login -u AWS -p <password> -e none aws_account_id.dkr.ecr.us-east-
1.amazonaws.com

# You can skip the `docker login` step if you have amazon-ecr-credential-helper
setup.

# Push an image to your repository
> docker push aws_account_id.dkr.ecr.us-east-1.amazonaws.com/hello-world
```

Lab: Working with Docker containers