Introduction to Microservices, Docker & Kubernetes

Somnuk Keretho, PhD

Department of Computer Engineering

Kasetsart University

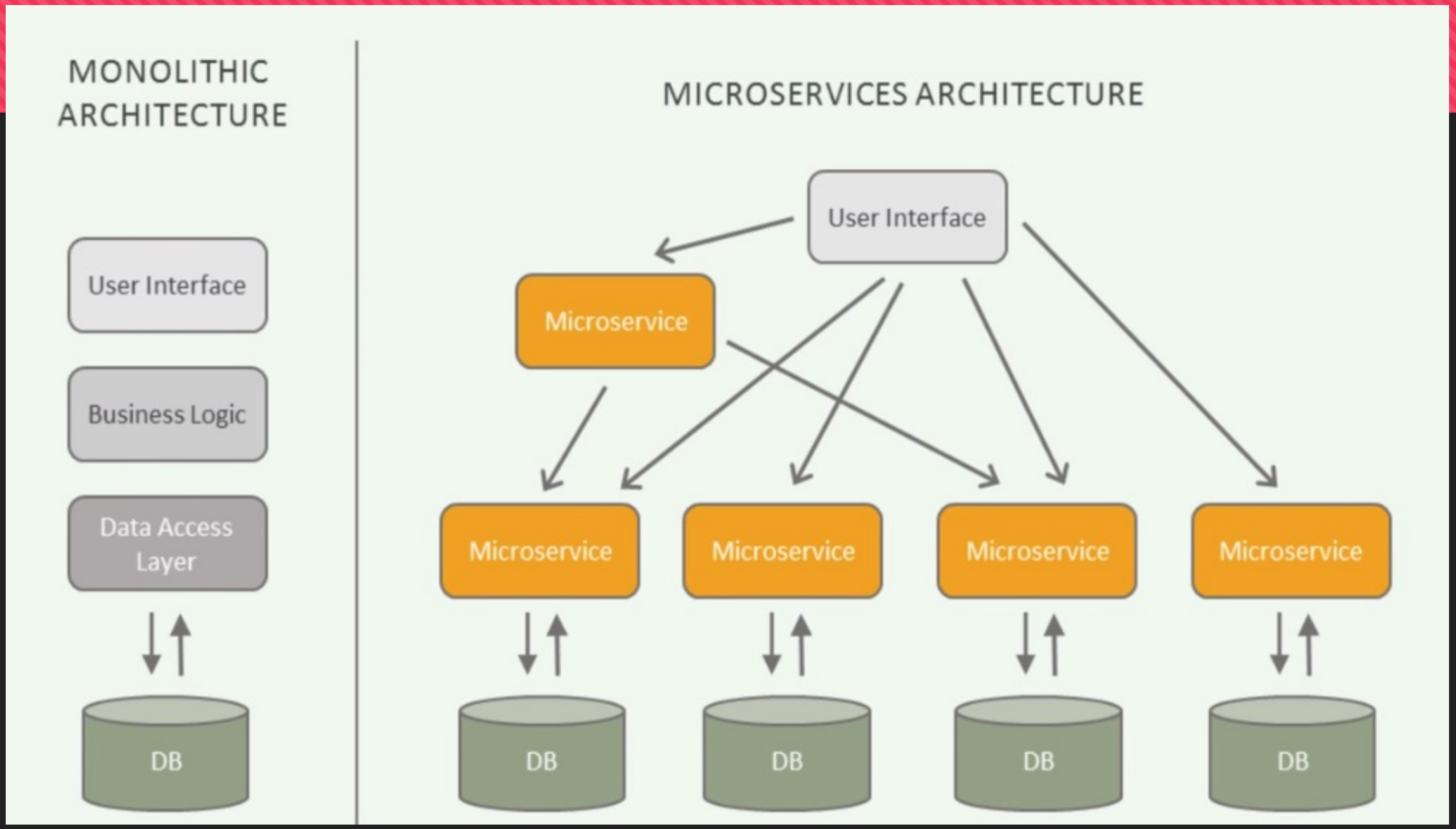
References

- Introduction to Microservices, Docker, and Kubernetes (with a hand-on tutorial)
  https://www.youtube.com/watch?v=1xo-0gCVhTU&feature=youtu.be
- Docker Tutorial for Beginners - Part 1 (What is Docker? | Docker Training | DevOps Tools)
  https://youtu.be/h0NCZbHjlpY
- Microservices • A lecture by Martin Fowler
  https://youtu.be/wgdBVIX9ifA
- The State of the Art in Microservices by Adrian Cockcroft (2015)
  https://www.youtube.com/watch?v=pwpxq9-uw_0&t=2840s

What is Microservice?

Microservices

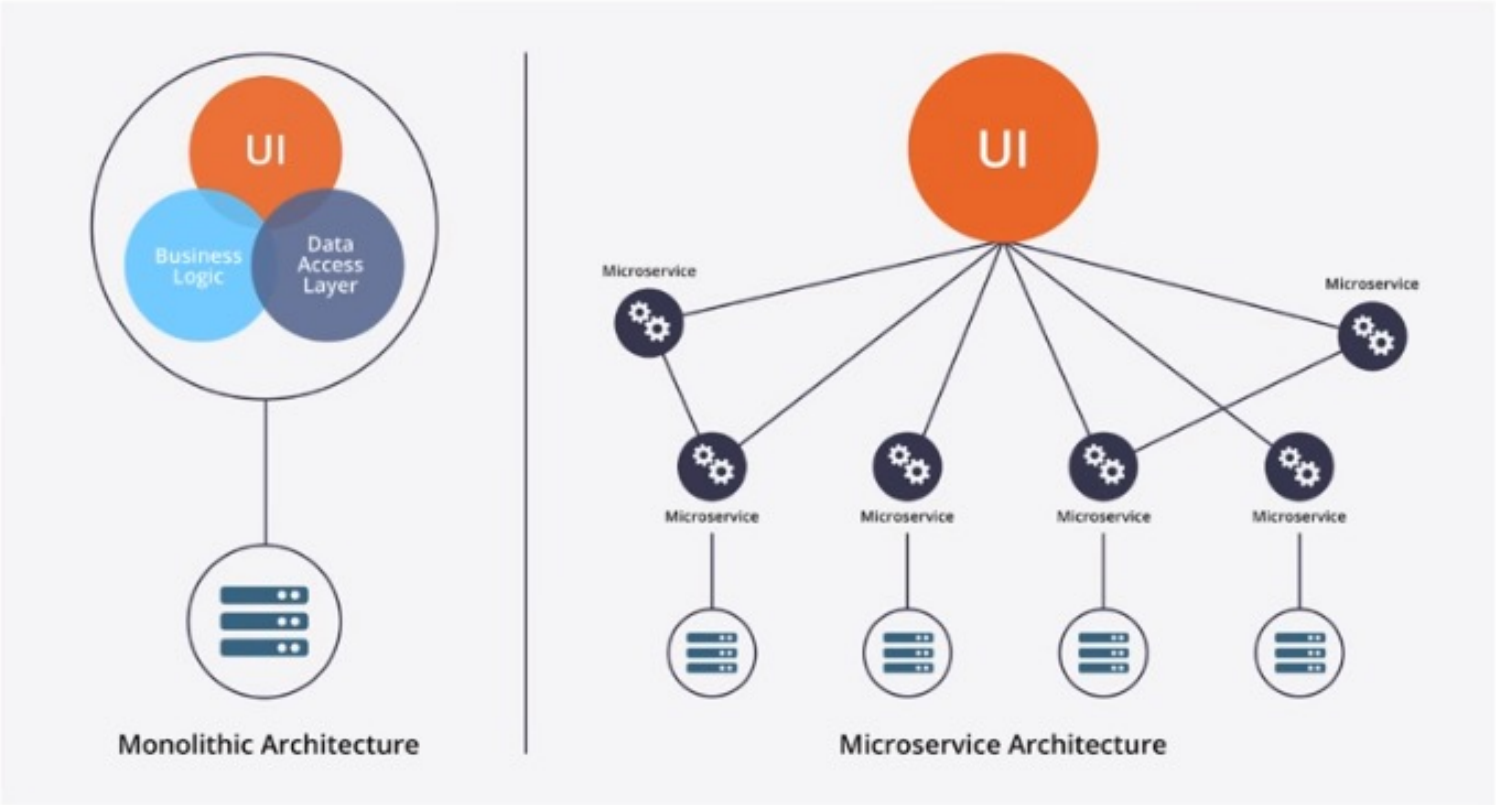
- Separate business logic functions
- Instead of one big program, several smaller applications
- Communicate via well defined APIs - usually HTTP
- In demand


MONOLITHIC ARCHITECTURE
User Interface
Business Logic
Data Access Layer
DB
MICROSERVICES ARCHITECTURE
User Interface
Microservice
Microservice
Microservice
Microservice
Microservice
DB
DB
DB
DB

Microservices - Advantages

Microservices - Advantages

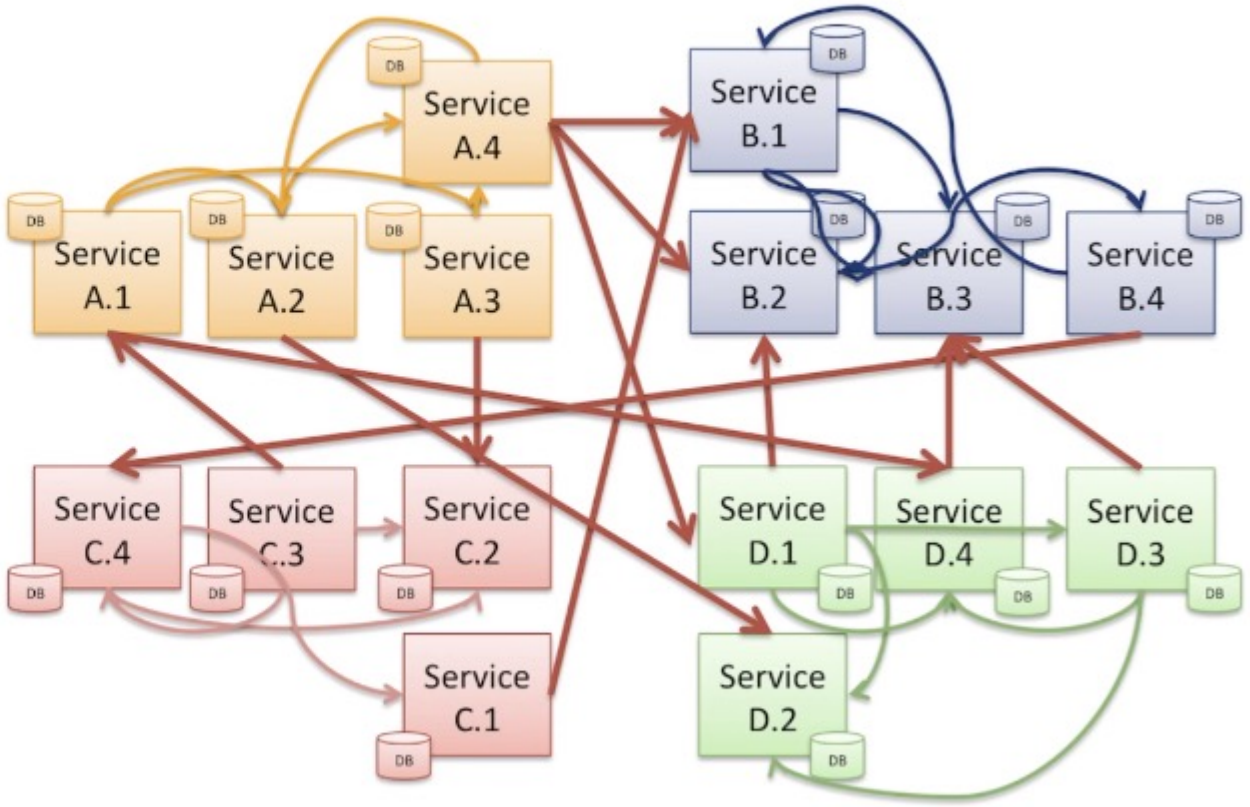
- Language independent
- Fast iterations
- Small teams
- Fault Isolation
- Pair well with containers
- SCALABLE
  - Big plus



Microservice - Disadvantages

Microservices - Disadvantages

- Complex networking
- Overhead
  - Databases
  - Servers

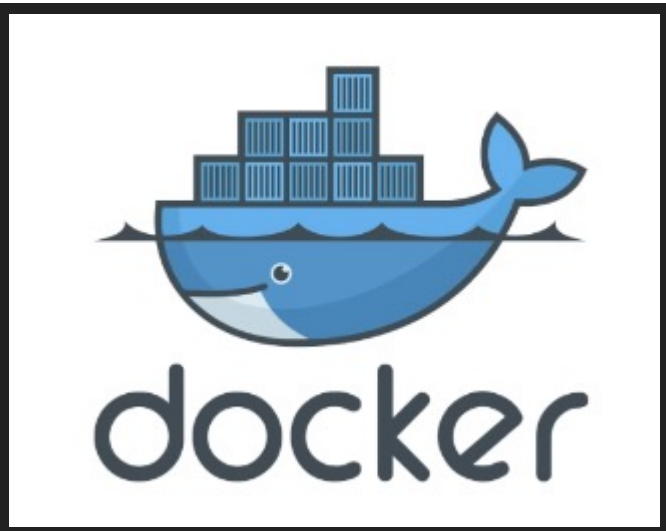


Docker

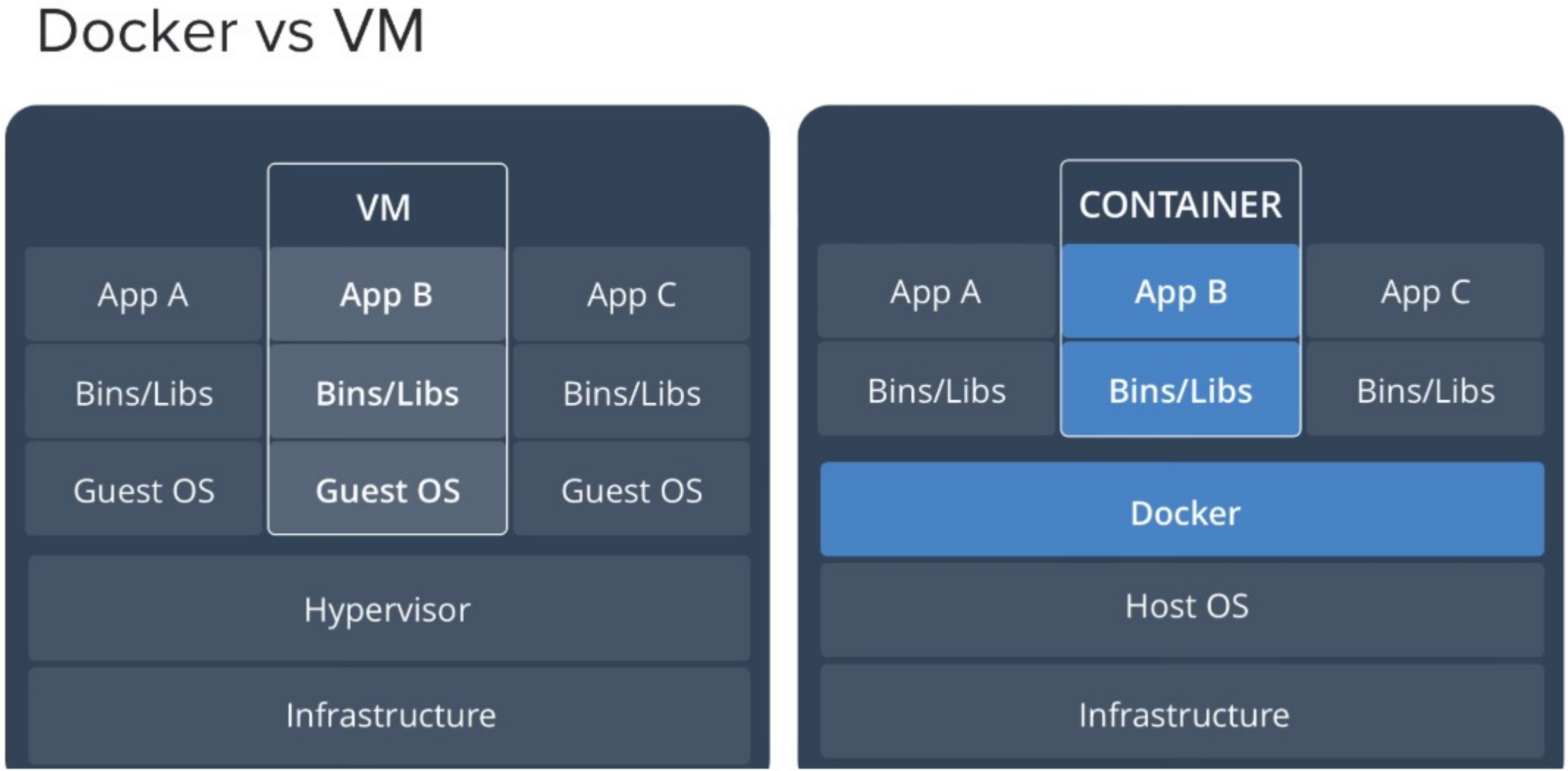
Docker is an open platform for developers and sysadmins to build, ship, and run distributed applications, whether on laptops, data center VMs, or the cloud.

What is Docker?

"Containers are a way to package software in a format that can run isolated on a shared operating system. Unlike VMs, containers do not bundle a full operating system - only libraries and settings required to make the software work are needed. This makes for efficient, lightweight, self-contained systems and guarantees that software will always run the same, regardless of where it's deployed."



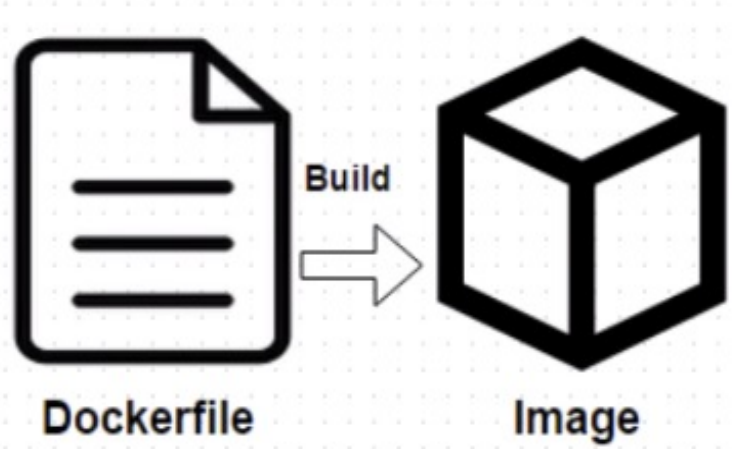
Docker vs VM

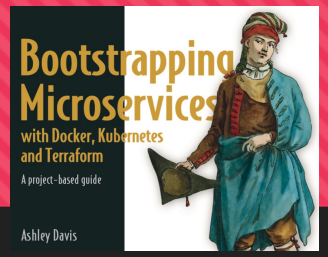

Docker vs VM
VM
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Guest OS
Guest OS
Guest OS
Hypervisor
Infrastructure
CONTAINER
App A
App B
App C
Bins/Libs
Bins/Libs
Bins/Libs
Docker
Host OS
Infrastructure

Dockerfile

Dockerfile

- Describes the build process for an image
- Can be run to automatically create an image
- Contains all the commands necessary to build the image and run your application





A Dockerfile

Listing 3.1 A Dockerfile for our video-streaming microservice (chapter-3/example-1/Dockerfile)

```
FROM node:12.18.1-alpine

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install --only=production

COPY ./src ./src

COPY ./videos ./videos

CMD npm start
```

Sets the base image for our new image. This allows us to produce new images based on existing images.

Sets the directory in our image. Other paths are relative to this.

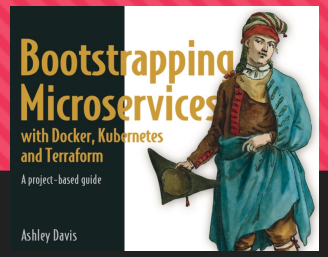
Copies the Node.js package.json file into the image

Installs only the production dependencies using npm

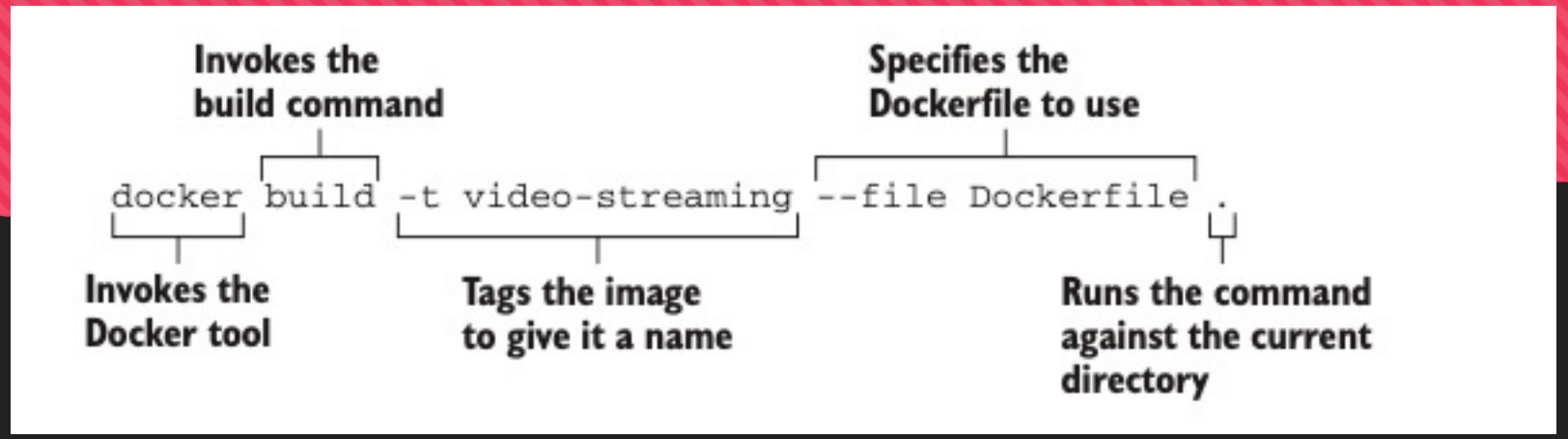
Copies the source code for our microservices

Copies our sample video

Starts the microservice using the "npm start" convention (see the previous chapter)



Docker build command



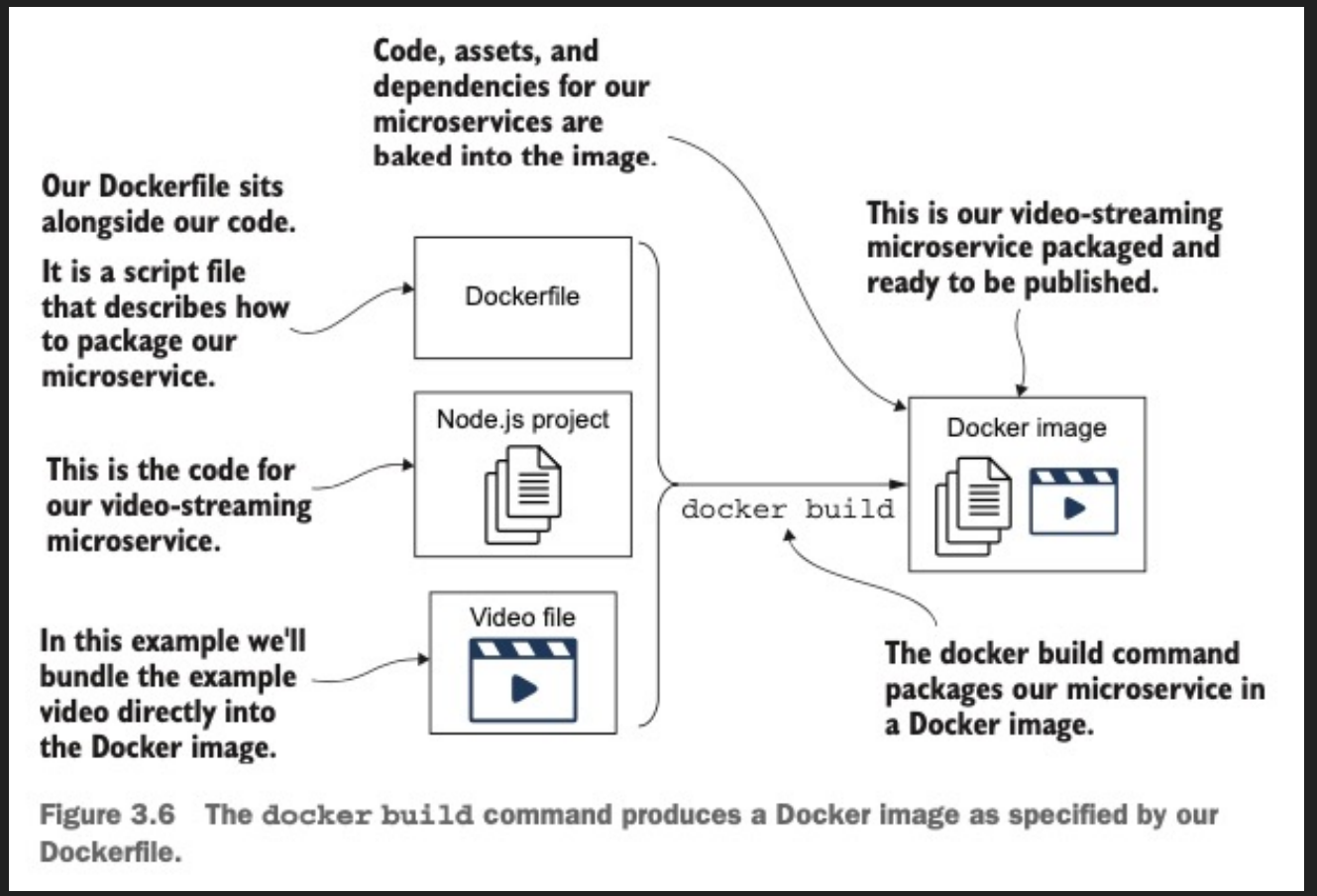
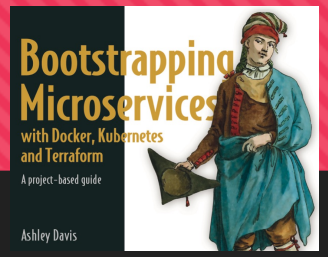


Figure 3.6 The `docker build` command produces a Docker image as specified by our Dockerfile.



Docker run command

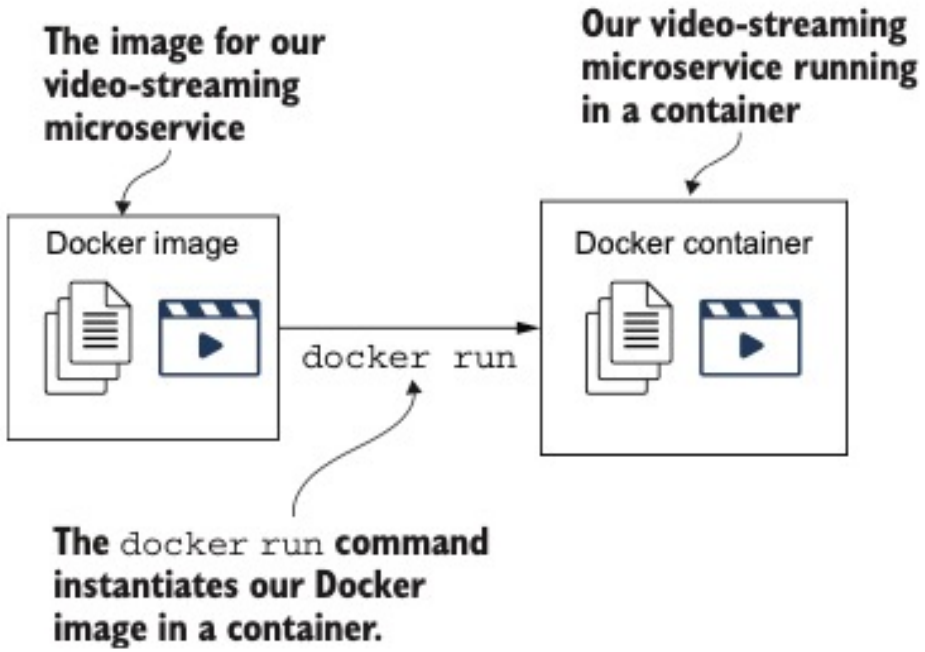
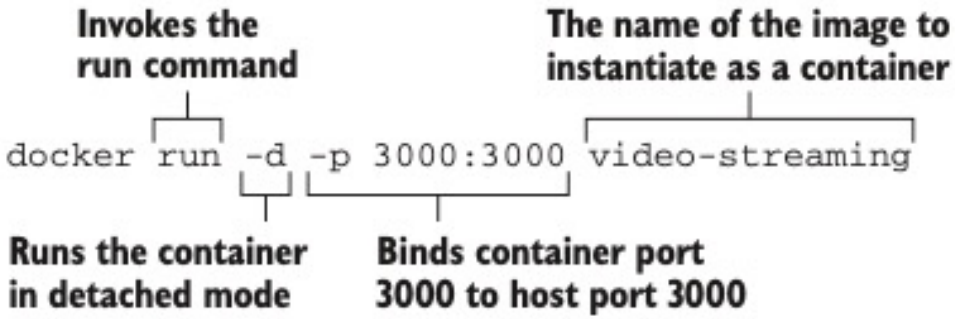
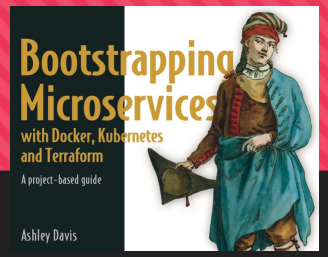


Figure 3.7 The `docker run` command produces an instance of our microservice running in a container.

When you are ready, open a terminal and invoke the following command to instantiate your microservice from the image:





Kubernetes

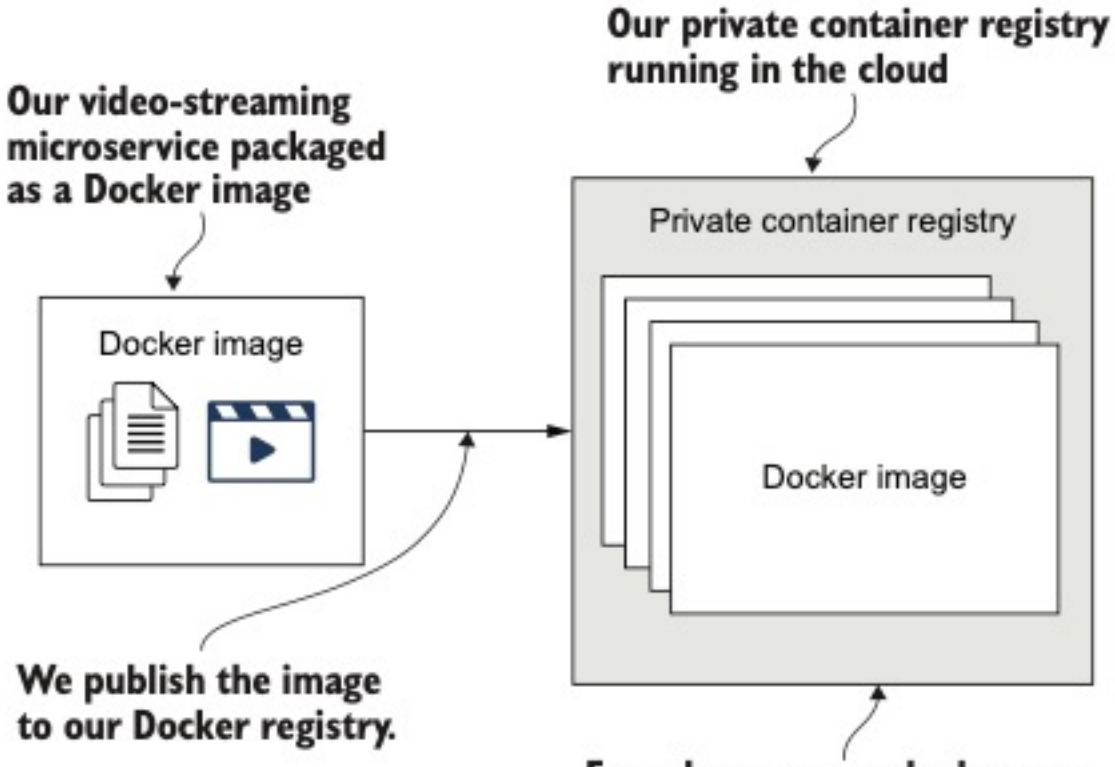
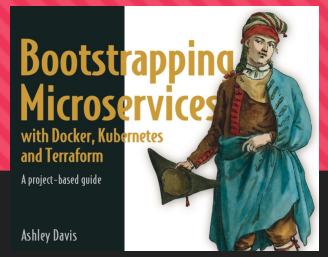


Figure 3.8 Publishing our Docker image to a private container registry in the cloud

it to our Kubernetes cluster. Figure 3.8 illustrates how we will now publish our image to a private container registry hosted in the cloud.



Docker build pipeline

We built our microservice on our development workstation.

We packaged our microservice in a Docker image.

We published our microservice to a private container registry in the cloud.

Later, we'll run our container in a production environment. In this chapter, though, we only simulated it on our development workstation.

Development workstation

Dockerfile

Node.js project

`docker build`

Docker image

`docker push`

Private container registry

Docker image

`docker run`

Development workstation

Docker container

We finished with a microservice running in a container on our development workstation.

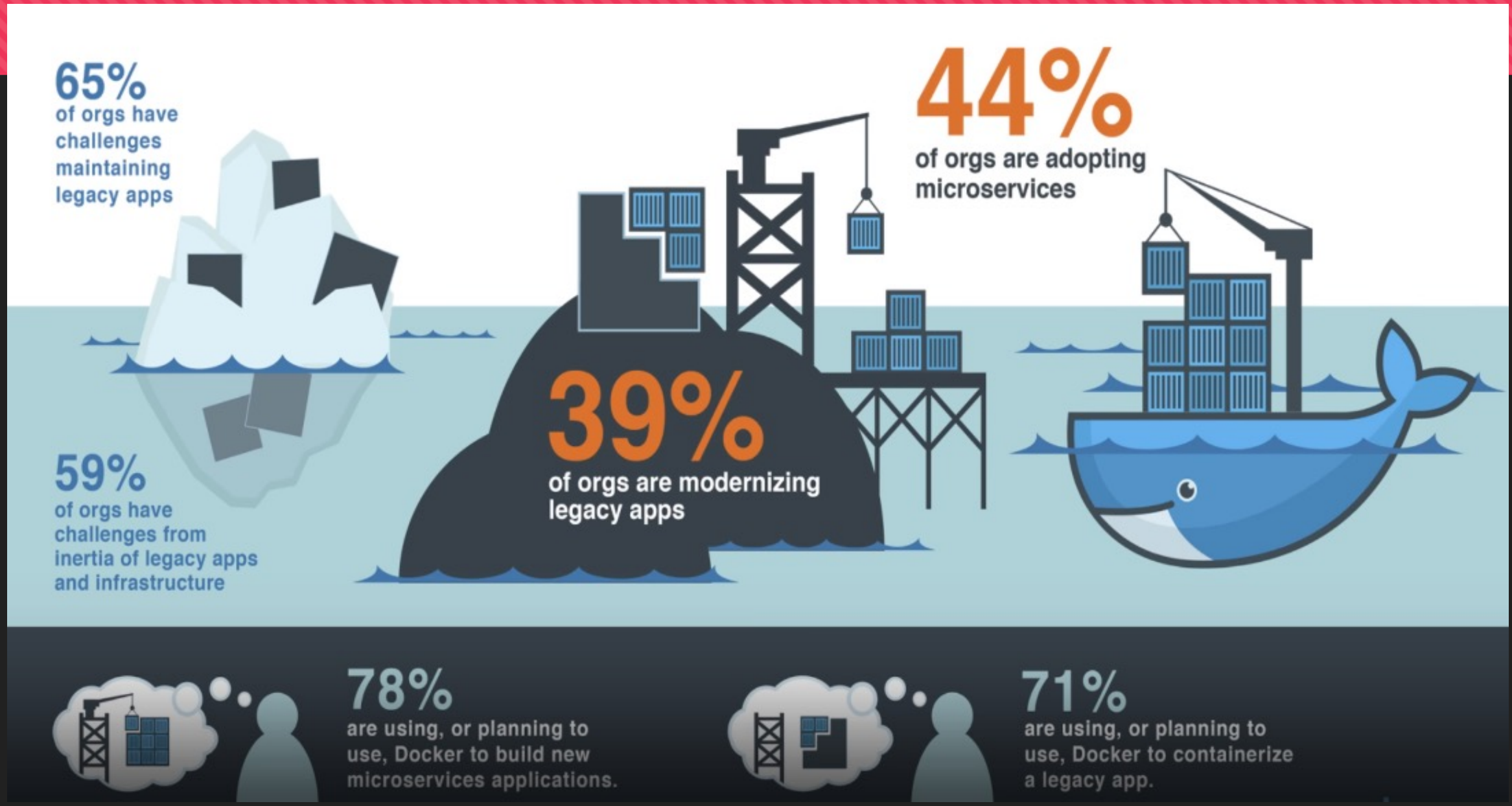
`docker build` packages our microservice in an image.

`docker push` uploads our image to our registry.

`docker run` instantiates our microservice in a container.

Figure 3.18 A complete Docker build pipeline showing where *build*, *push*, and *run* fit within the process.

A Docker Adoption Report



Container Orchestration

Deploying and scaling containers

Docker Compose

Another tool from the developers of Docker, builds on top of Docker to more easily manage a multiple-container application.

Docker Compose

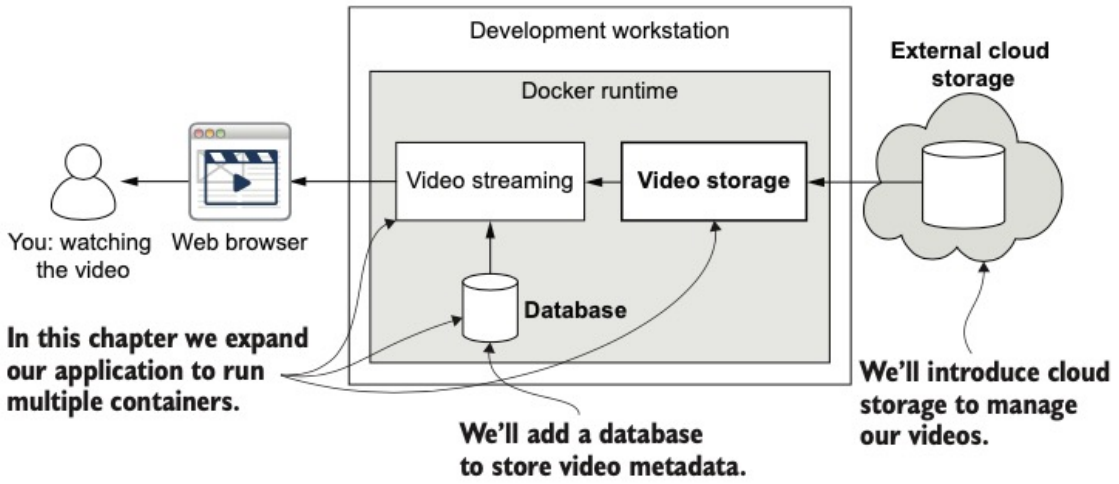


Figure 4.2 We expand our application to multiple containers.

To build, run, and manage our growing application, we could get by running the various Docker commands multiple times (repeated for each image or container). But this quickly becomes tedious during development because we'll need to stop and restart our application many times during our working day. And this only gets worse! As our application continues to grow, we'll add even more containers to it. We need a better tool.

4.3.1 Why Docker Compose?

Managing multiple containers in development can be painstaking; in chapter 6 you'll see how we'll use Kubernetes to manage containers in production. However Kubernetes is a big and complex system designed to run on multiple computers (you need at least one master and one node). It's not easy to "simulate" Kubernetes on a development workstation. You could use Minikube to do this, that's like a cut-down version of Kubernetes. But there's an easier way, and you might even already have it installed—Docker Compose.

Why Docker Compose? In the same way that Docker allows us to build, run, and manage a single microservice, Docker Compose gives us a convenient way to build, run, and manage multiple microservices in development.

Docker Compose, another tool from the developers of Docker, builds on top of Docker to more easily manage a multi-container application. During development and testing, we must boot and reboot our entire application frequently. And after

Creating our Docker Compose file

DEFINITION The *Docker Compose file* is a script that specifies how to compose an application from multiple Docker containers.

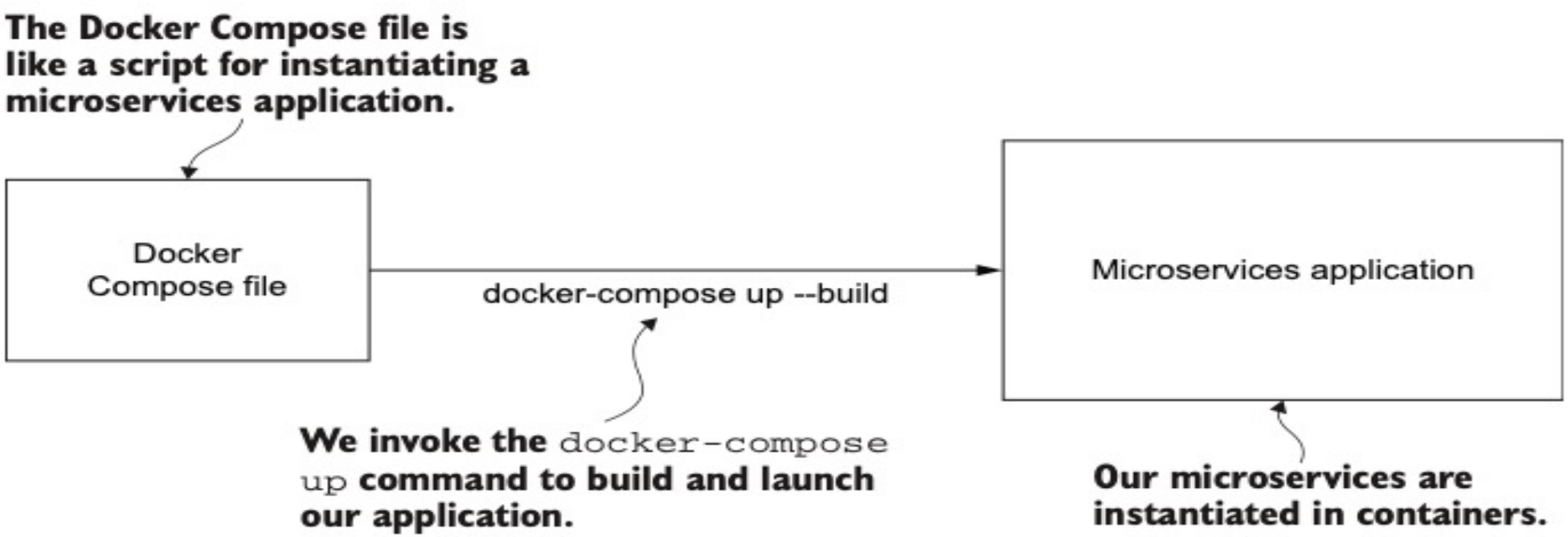
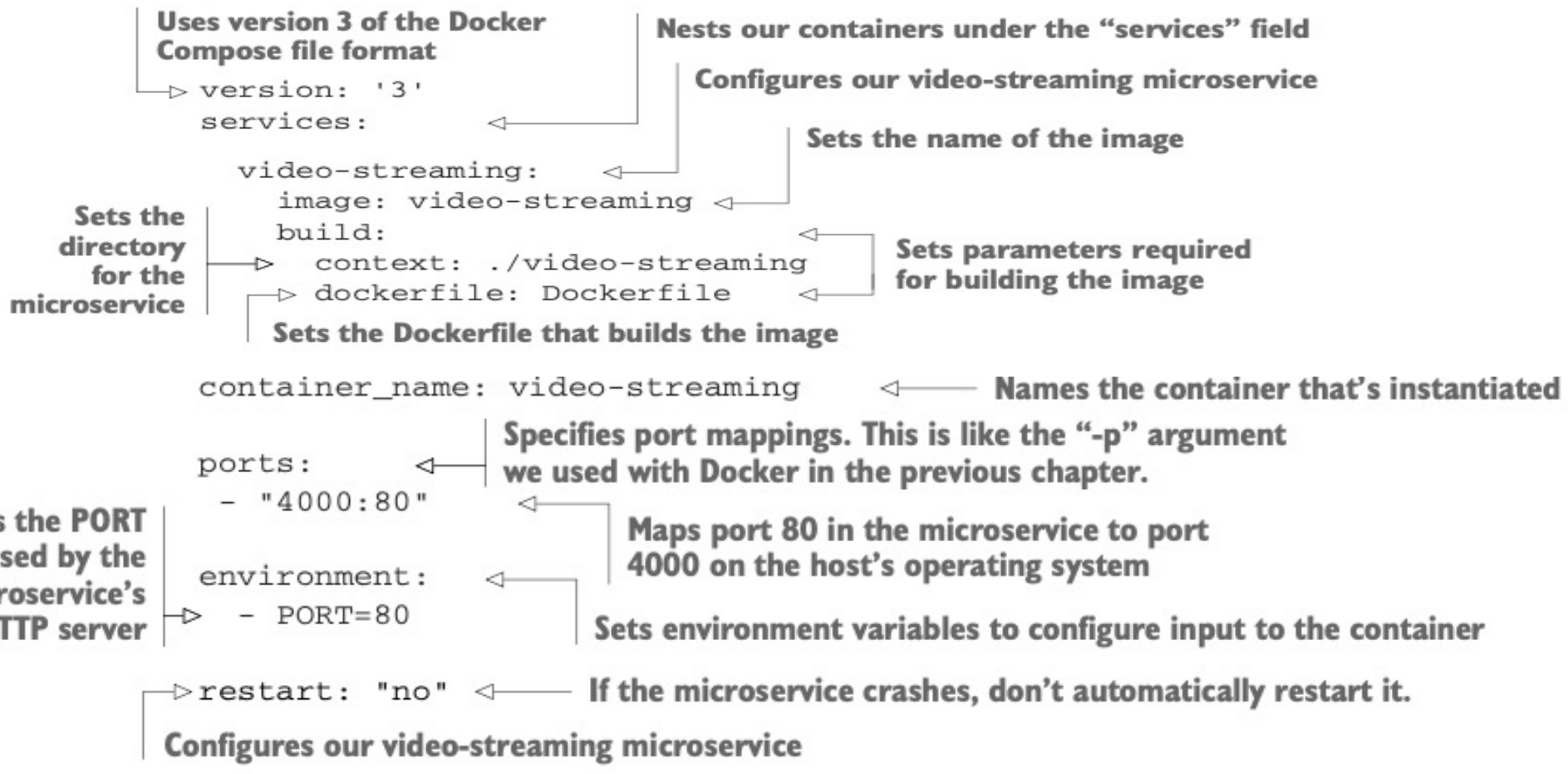


Figure 4.3 The Docker Compose file is like a script for building and launching a microservices application.

Docker Compose file

Listing 4.1 Docker Compose file for our microservice (chapter-4/example-1/docker-compose.yml)



Kubernetes

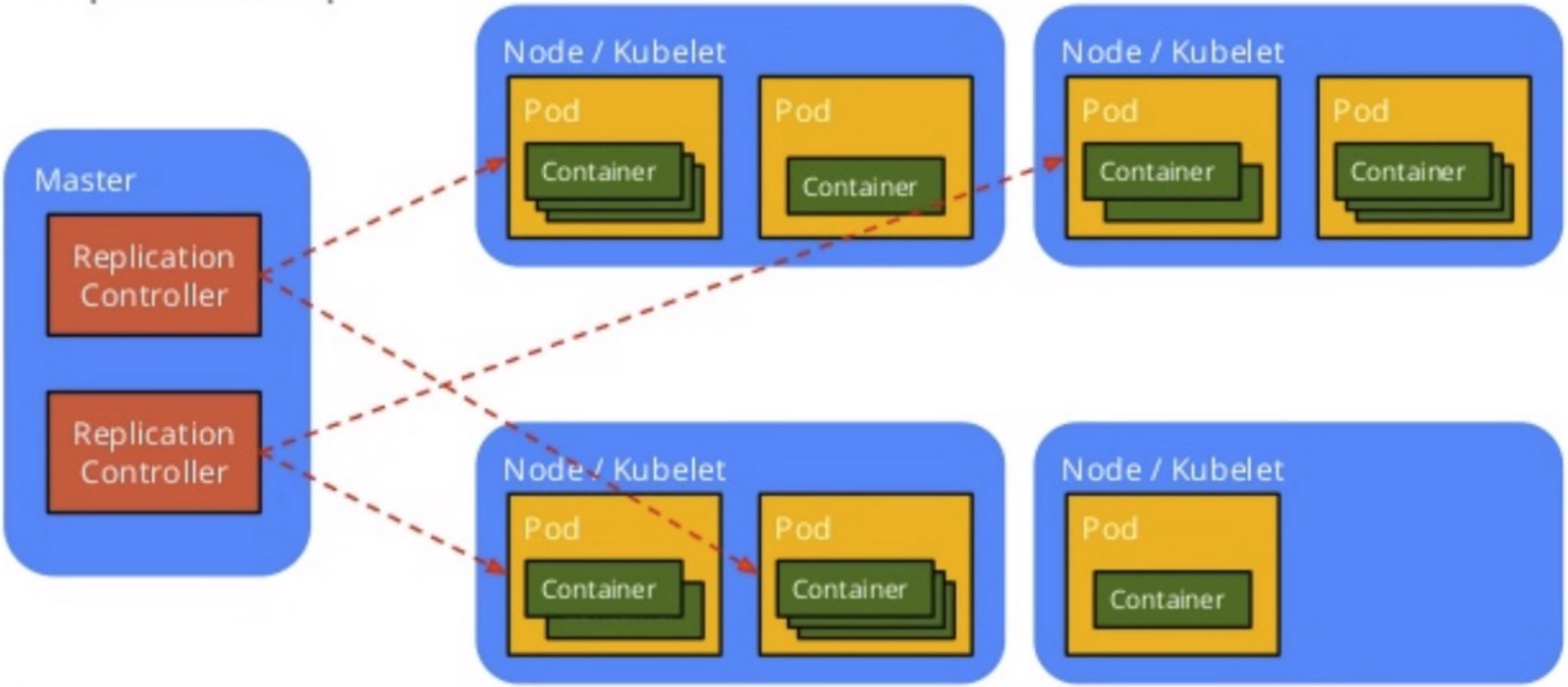
Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.

Kubernetes Vocab

Kubernetes Vocab

- Node
  - Kubelet
  - Communicates with master
  - Runs pods
- Pod
  - Runs 1+ containers
  - Exists on a node
- Service
  - Handles requests
  - Usually a load balancer
- Deployment
  - Defines desired state - kubernetes handles the rest

A quick recap



Deployment Yaml

Deployment Yaml

```
apiVersion: apps/v1beta1 # for versions before 1.6.0 use extensions/v1beta1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 3
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```