COMP5349– Cloud Computing Week 10: Container and Microservices Architecture

Dr. Ying Zhou

The University of Sydney

Table of Contents

Container

Containerization

- **“Operating-system-level virtualization**, also known as **containerization**, refers to an **operating system feature** in which the kernel allows the existence of multiple *isolated* user-space instances.”
- “Such instances, called **containers**, partitions, virtualization engines (VEs) or jails (FreeBSD jail or chroot jail), may look like real computers from the point of view of programs running in them."

https://en.wikipedia.org/wiki/Operatingsystem-level_virtualization#Implementations

Linux Kernel, System and Distribution

- Linux kernel is the most basic component of a Linux Operating System. It does the four basic jobs
  - Memory Management
  - Process Management
  - Device Drivers
  - System Calls and Security
- Linux System
  - Kernel + system libraries and tools
    - E.g. GNU tools like gcc,
- Linux distribution
  - Pre-packaged Linux system + more applications
    - E.g. news servers, web browsers, text-processing and editing tools, etc.
  - Redhat, Debian, Amazon Linux, Android(?)

Linux Kernel Feature: Namespace

- To create an illusion of full computer system for each container, we need to give each a “copy” of the kernel resources
  - An independent file system starting with a root directory: /
  - An independent set of process ids, with id 1 assigned to *init* process
  - An independent set of user uids, with id *0* assigned to root user
  - Etc..
- To avoid conflict, OS provides a feature called **namespace**
- The namespace provides containers their own view of the system

Namespace Kinds

Early Linux has a single namespace for each resource type

Gradually, namespaces are added to different resources

- Mount (mnt)
  - This deals with file system
  - Each container can have its own rootfs
  - Each container manages its own mount points
- Process ID (pid)
  - Each container has its own numbering starting at 1
  - When PID 1 goes away, all other processes exit immediately
  - PID name spaces are nested. The same process may have different PIDs in different namespaces
- User ID
  - Provide user segregation and privilege isolation
  - There is a mapping between container UID to host OS UID
  - UID 0 (root) in container may have a different UID in the host

Linux Kernel Feature: Cgroup

- Control Groups (cgroups) is another kernel feature to enable isolated container
- It is used to control kernel resource allocated to each container/process
  - Metering and limiting
- The resources include:
  - Memory
  - CPU
  - I/O (File and Network)

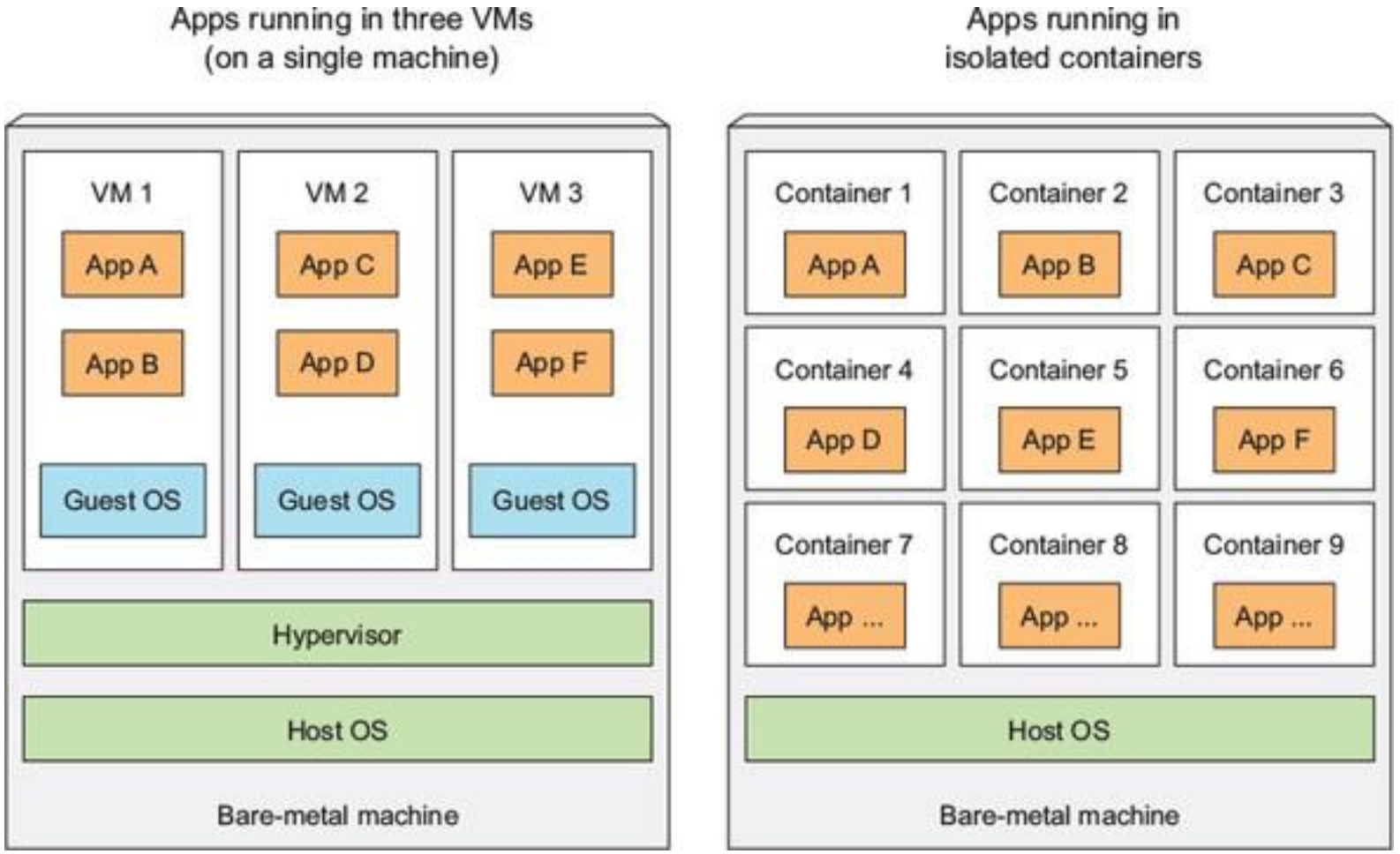
Container Runtimes

- Runtime
  - Life cycle of a program, e.g runtime error
  - Language specific environment to support its execution, e.g. JRE
- Container runtime has similar responsibilities as a Java Runtime Environment (JRE)
  - It enables containers to run
  - by setting up namespaces and cgroups
- It can also be viewed as the counterpart of hypervisor

Container Runtimes

- Low level container runtimes, focus on just running containers
  - LXC, Systemd-nspawn, OpenVZ, Sandboxie, etc…
- High level container runtimes contain lot of other features like defining image formats, managing images, etc
  - Docker

Container vs. Virtual Machine



All containers use the kernel of the host machine

Each VM contains a full OS

Kubernetes in Action, 1st Edition Figure 1.4

Container vs. Virtual Machine (cont'd)

Operating Systems and Resources

- Running full fledged OS inside VM takes up certain amount of resources even if no app is running in the VM
- Starting OS takes some time
- Container exits when the process inside it finishes unless we specify interactive container, or run a server in it.
- Container is much faster to start, similar to starting an application

Isolation for performance and security

- VMs have very good isolation and security
- They enjoy hardware support and mature technology
- Containers offer reasonable level of isolation using *kernel techniques like namespace and control groups*

Docker Overview

Overview

- Docker is the most “famous” container runtime
- It is not just a container runtime, it is a packaging and deployment system build on container technology
- There is a large ecosystem of various components
- The container part is usually presented as a black box where docker users do not need to know a lot about how it works
- For most users, the dependency management and deploy everywhere are the most prominent features

Main concepts

Images

- A Docker-based container image is something you package your application and its environment into.

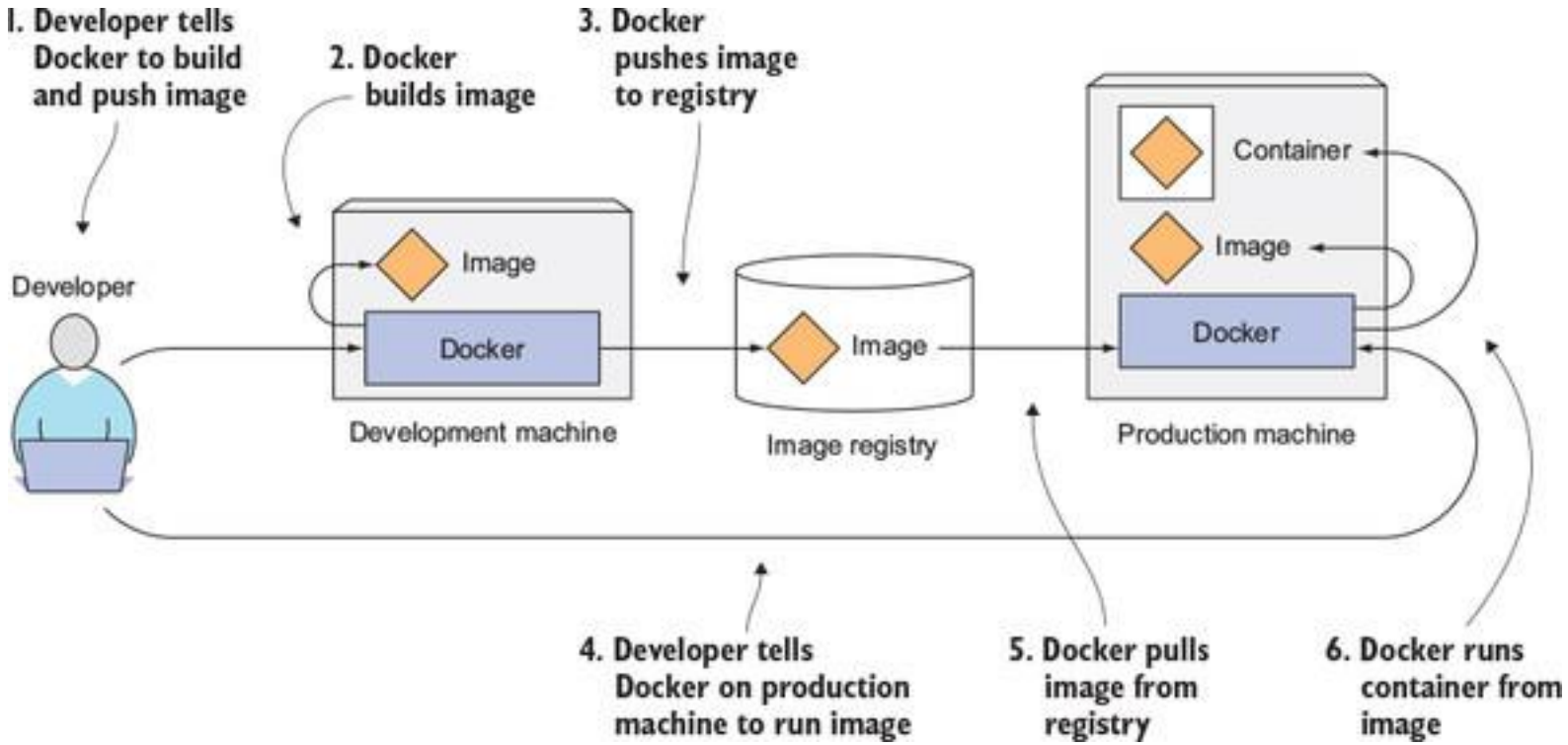
Registries

- Docker Registry is a repository that stores your Docker images and facilitates easy sharing of those images between different people and computers.

containers

- A Docker-based container is a regular ~~Linux~~ container created from a Docker-based container image. A running container is a process running on the host running Docker.

Images, registries and containers



Docker image

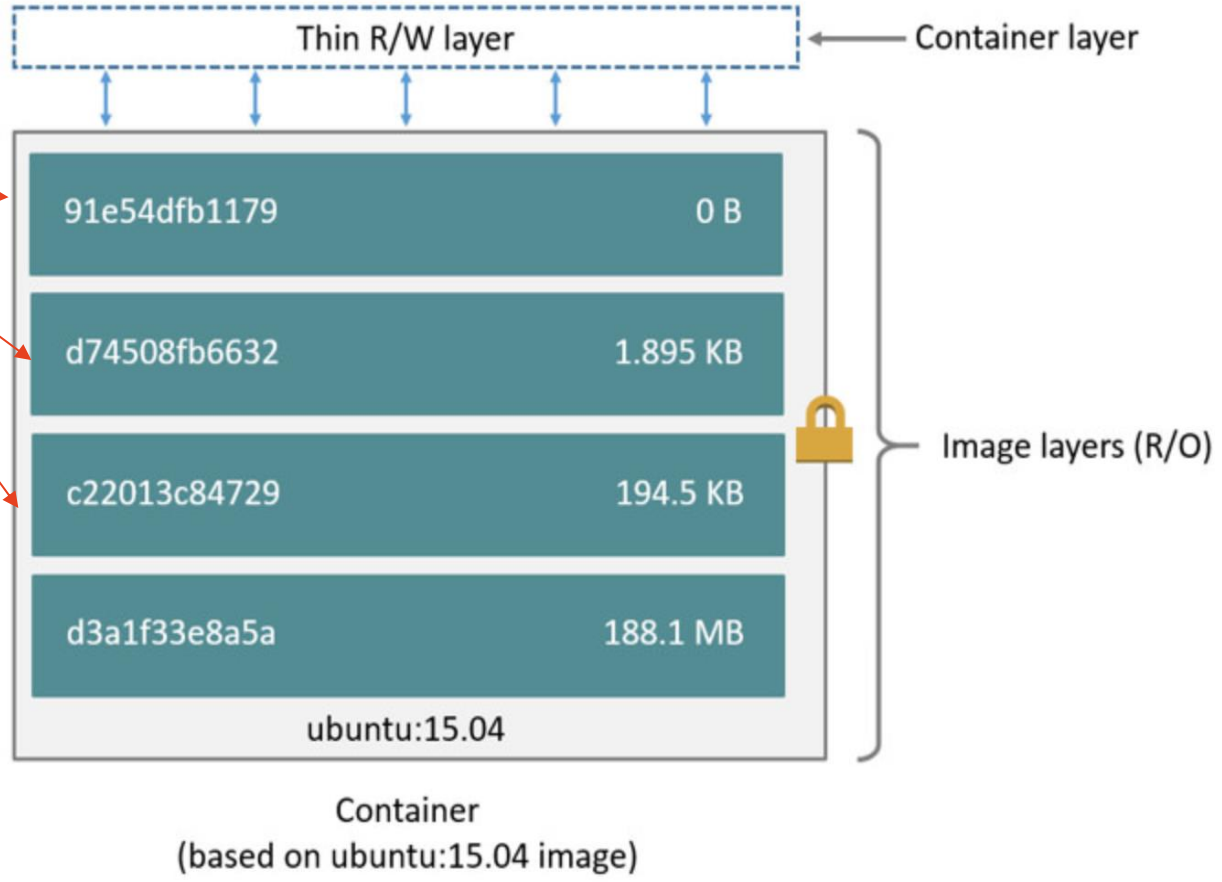
- Docker images are defined in **`Dockerfile`**
  - A text document containing a sequence of instructions/commands to build and run your application
  - If you ever written something like a *make* file, *ant* build file, Maven *POM* file, etc. etc. the `Dockerfile` is designed for similar purpose
  - You declare dependencies, set environment variables and other configurations in it
- Docker images are composed of read-only layers
- Base layers can be shared by many images

Image Layers

```
FROM ubuntu:15.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Obtain some layers from a parent image

Create a layer each



https://docs.docker.com/v17.09/engine/userguide/storagedriver/imagesandcontainers/

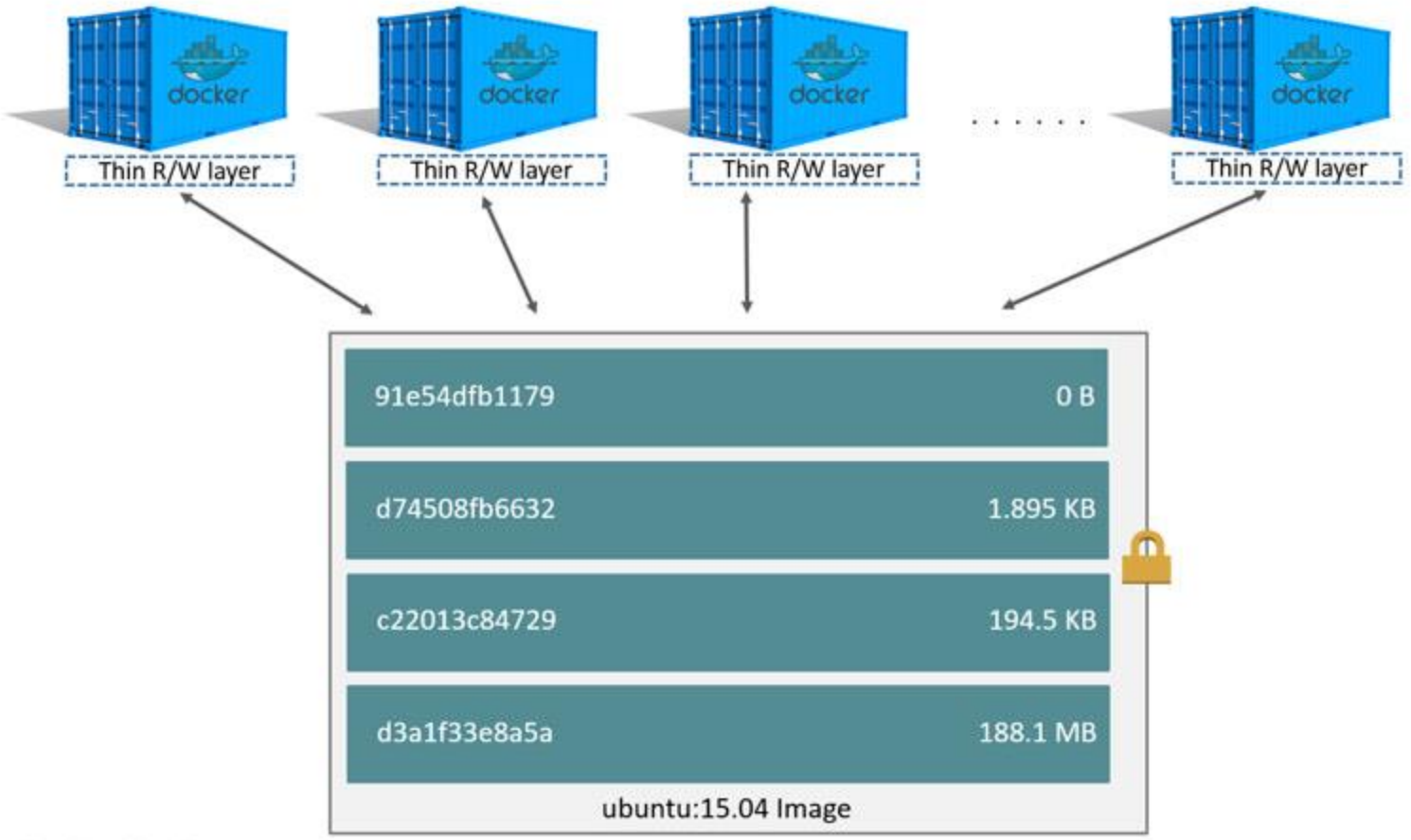
Container and image layers

- Images are read-only templates, all image layers are read-only
- When an image is loaded into a container to run, the container add a thin writable layer on top of it.
  - All writes to the container are stored in this layer
  - It is deleted when container exits (is deleted)
  - Should be used as temporary storage

Container and image layers

- If multiple images use a same base image, only one copy of the base image is required
  - Small space
- When container starts, only a new writable layer is added on top of the existing image layers
  - Fast start up
- Application persistence should be handled differently

Multiple containers sharing the same image

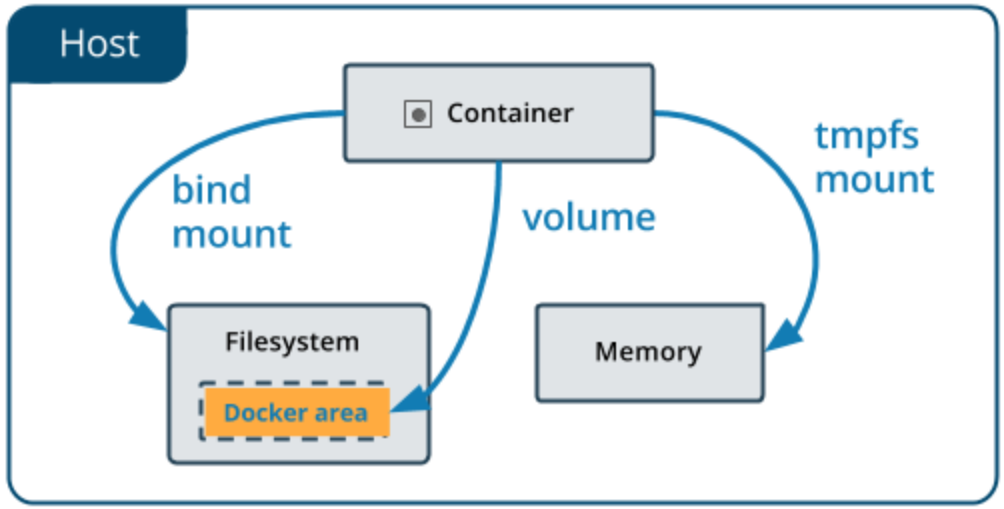


Docker Container

- The actual container uses many OS technologies to provide isolation and to allocation resources: cpu, memory, i/o
- This is achieved by utilizing various features provided by Linux kernel
  - Namespaces
  - Cgroups
  - Others
- **Linux Containers (LXC)** was used as the execution driver
- The current driver **Libcontainer** is more general, allowing Docker to run on platform other than Linux

Data Volume

- Apart from storing data within the writable layer of a container, there are other preferred ways to persist data
  - Bind mount is an early version storage option, it allows a client to mount any file or directory on the host machine into a container
  - Volumes uses a designated location in the host machine as container storage, it is completely managed by Docker
  - tmpfs is a rarely used option, which uses host memory to simulate storage



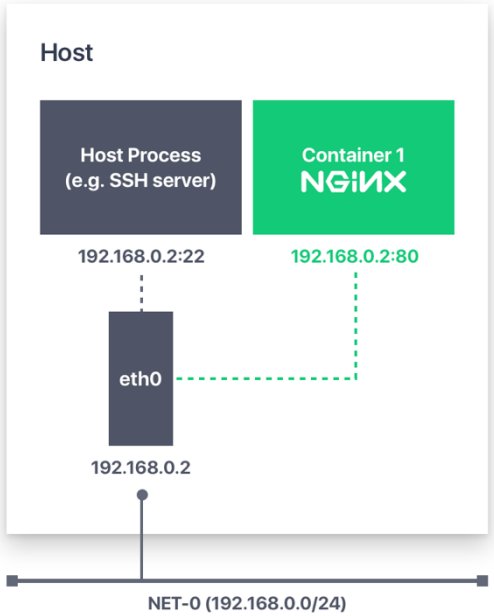
Networking

- Docker containers have networking enabled by default and they can make outgoing connections
- Docker provides a few options for container to decide how it wants to connect with outside
  - **Host**
  - **Bridge**: default/user defined
  - **None**
  - Overlay
  - Others…
- These are specified during the start of the container, if nothing is specified, the default **bridge** driver is used.

https://docs.docker.com/network/

Networking: Host

- Host driver is the simplest option,
  - It removes the isolation between container and host
  - Container is treated in the same way as a process in the host
  - It connects directly to the host NIC (host networking namepace)

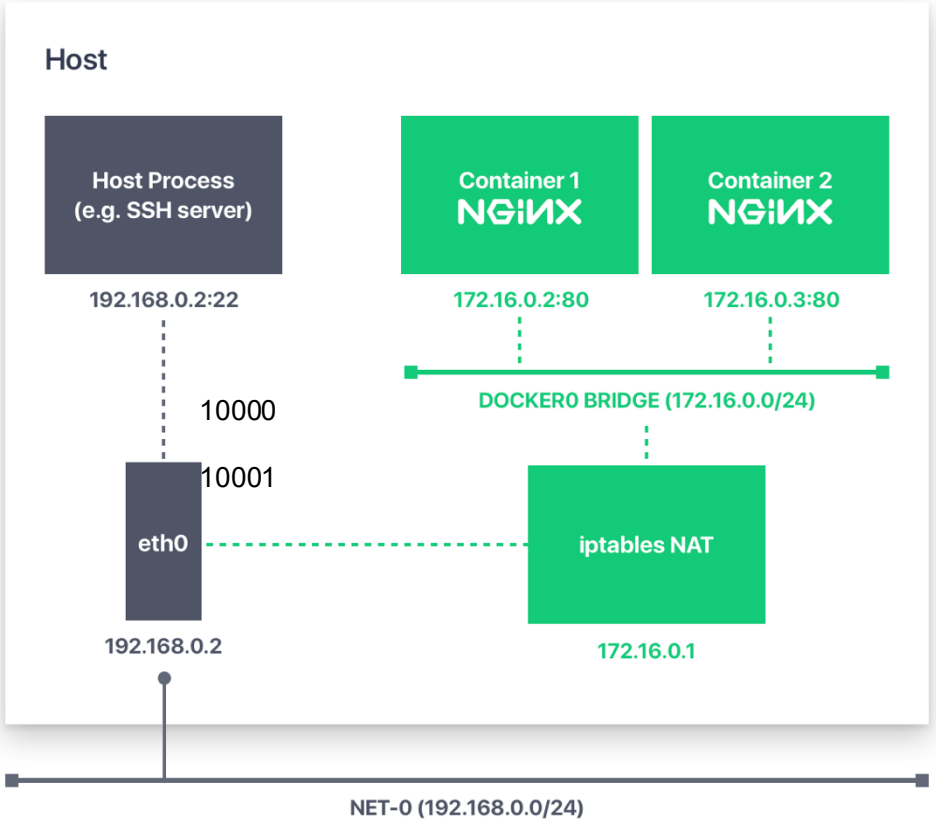



```
docker run -d --name nginx-1 -net=host nginx
```

https://mesosphere.com/blog/networking-docker-containers/

Networking: Bridge

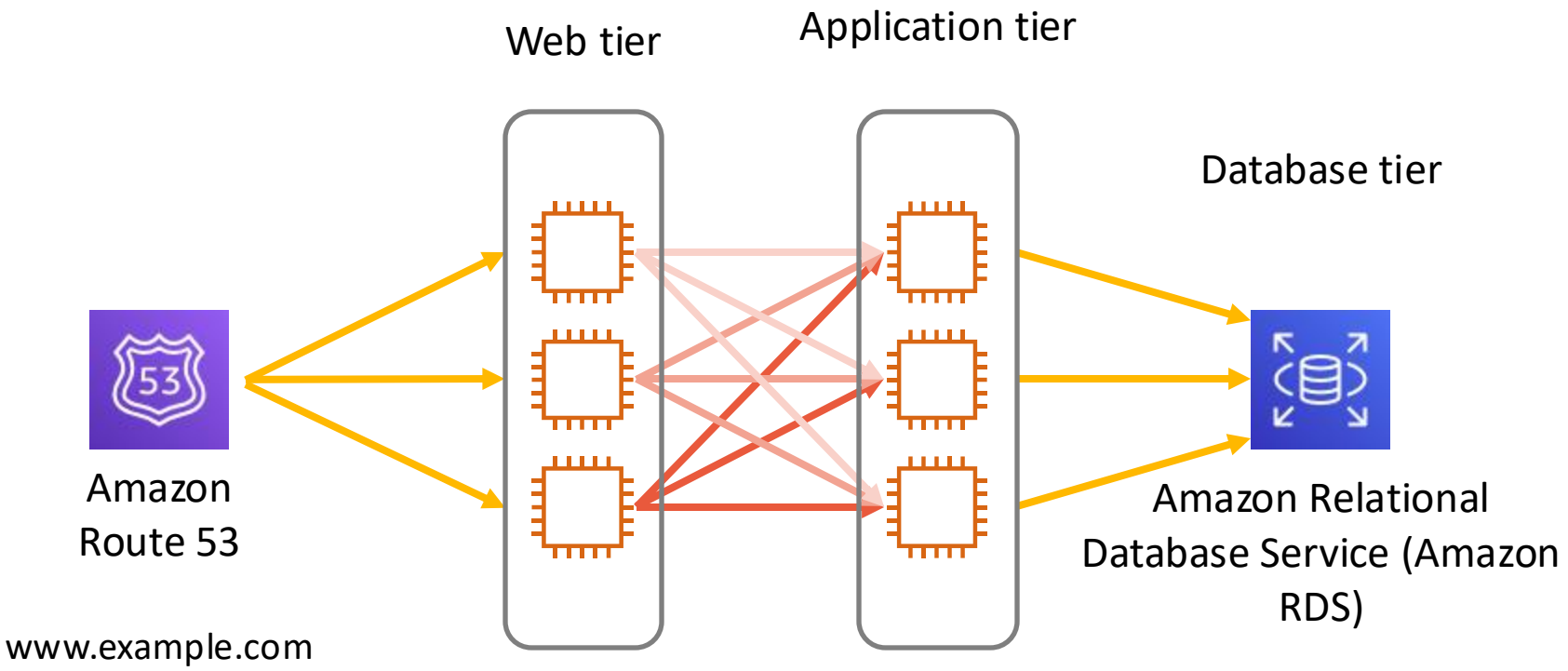
- Docker manages its own private network
  - Adding a software bridge to the host
  - The mapping from internal/private address to public/host based address is done through Network Address Translation (NAT)




```
docker run –d –-name nginx-1 -p 10000:80 nginx
docker run –d –-name nginx-2 -p 10001:80 nginx
```

Microservices Architecture

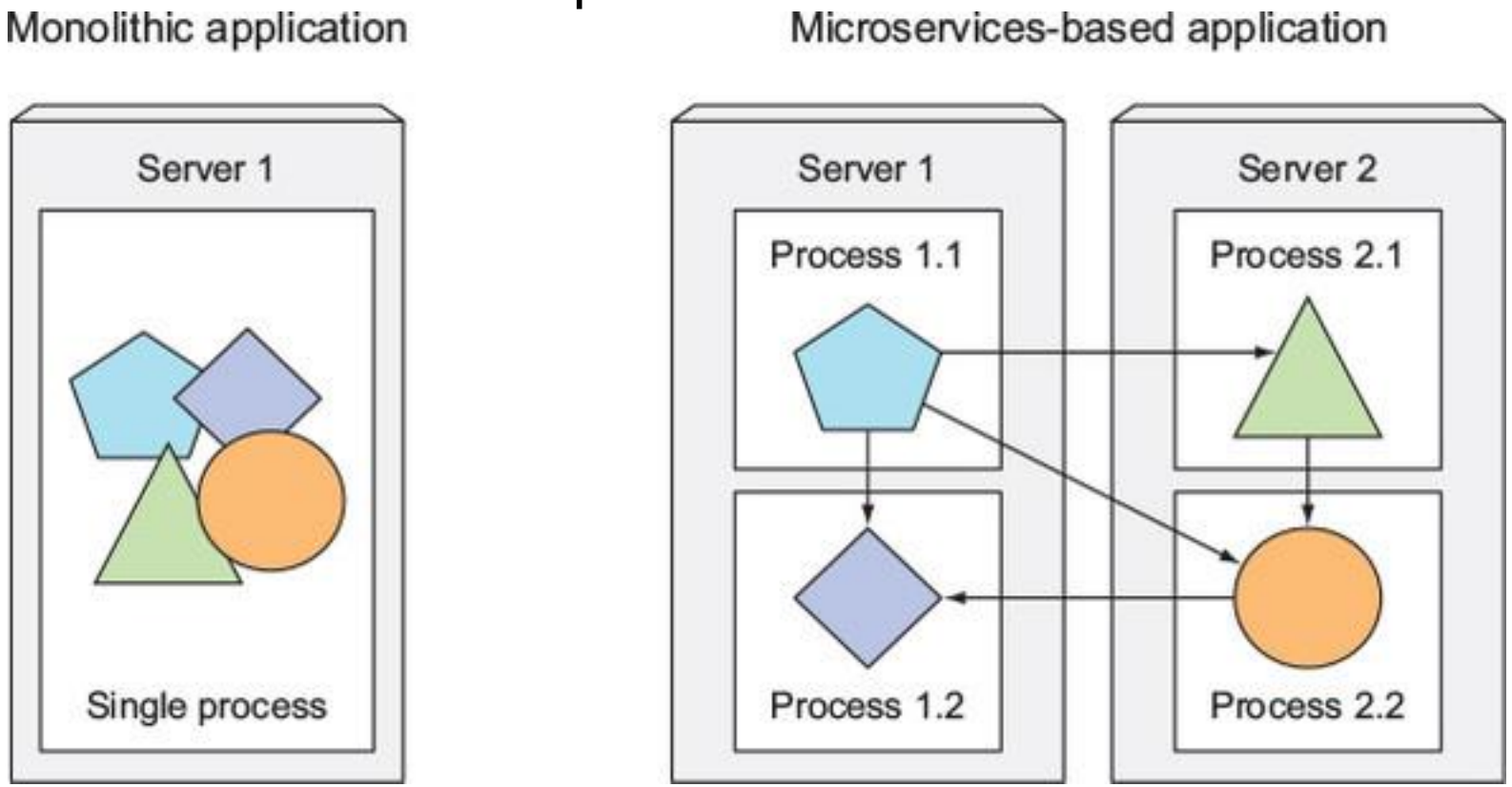
MVC Architecture



Components are strongly connected to each other.

Microservices Architecture

- Decouple services into individual services that can be deployed separately from one another on separate hosts.

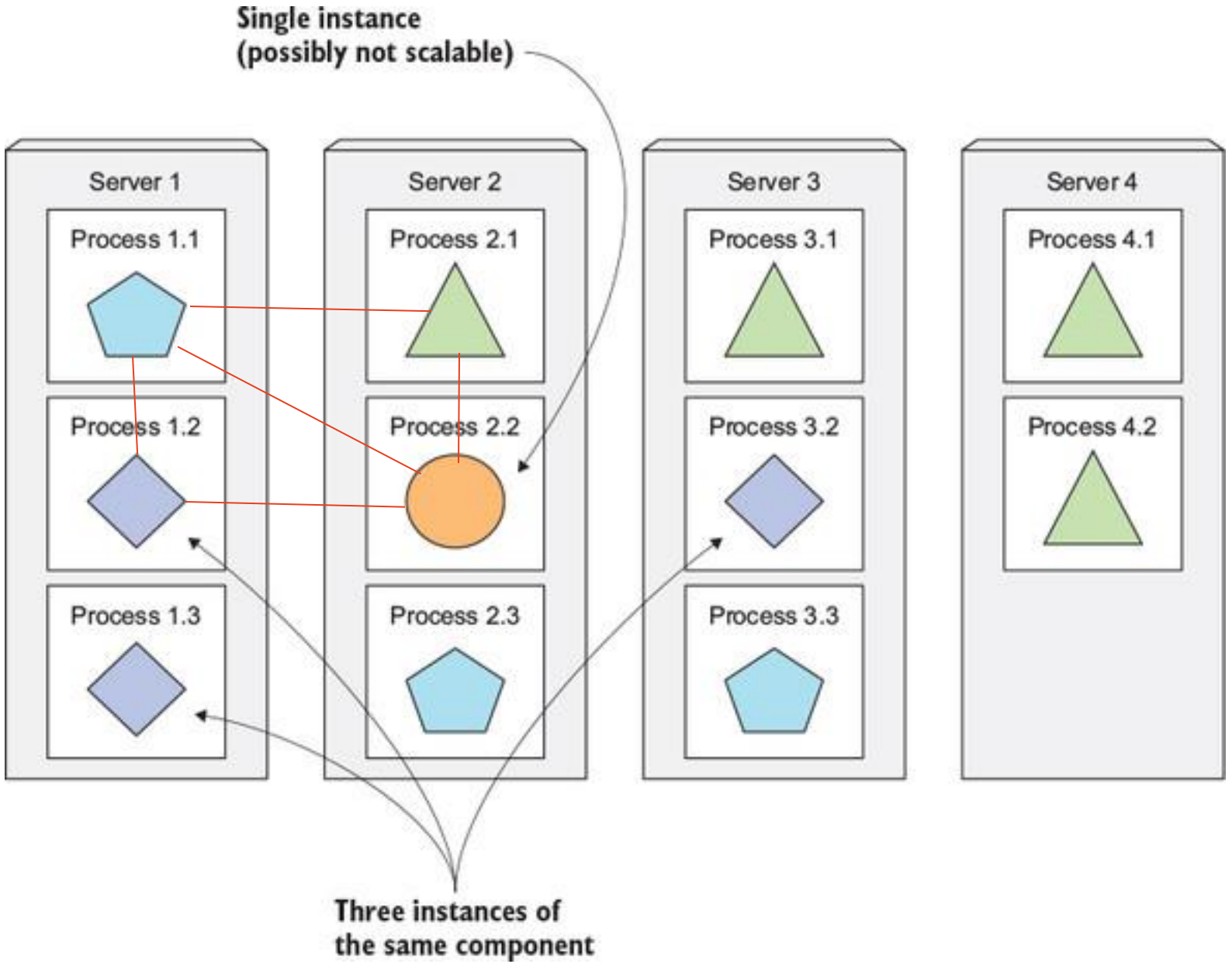


RESTful API over HTTP

Stable interfaces among microservices

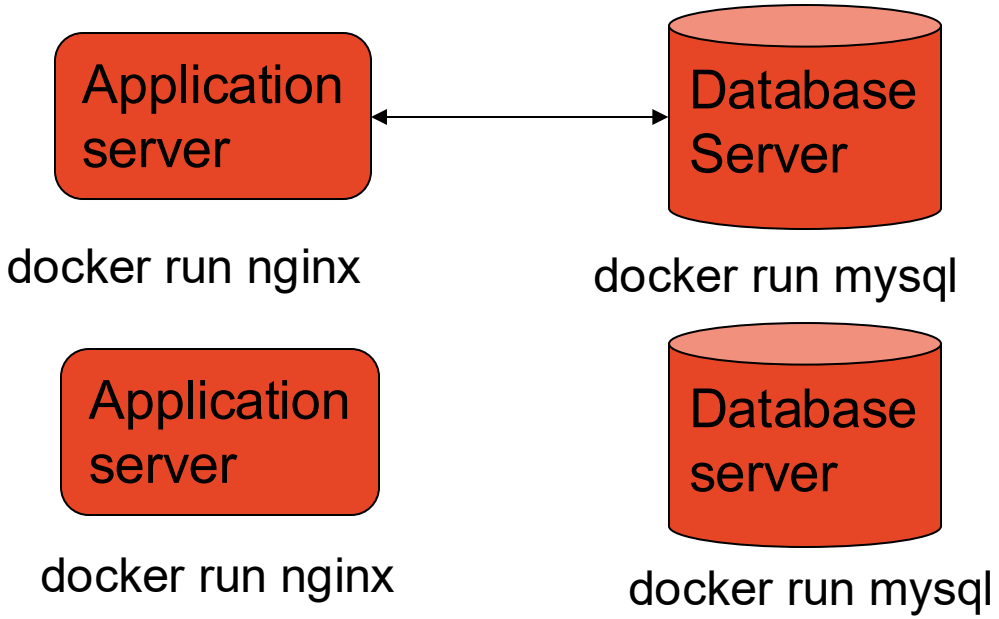
Kubernetes in Action, 1st Edition Figure 1.1

Scaling microservices

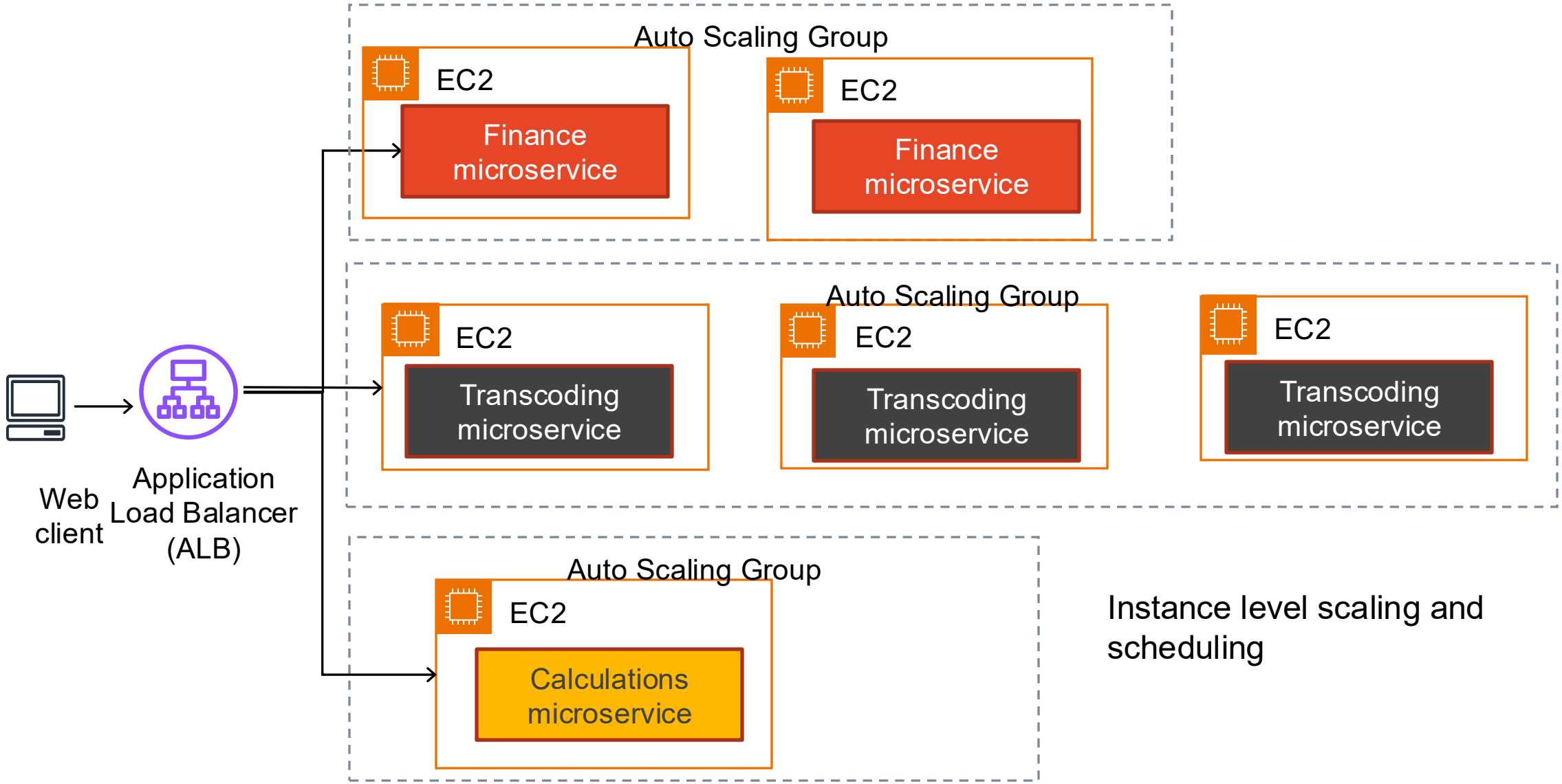


Container and microservices

- Each container is supposed to run a single service
  - Deployment is simplified since all dependencies are included in the container
  - More resource efficient than running a service on a VM
- Docker focuses on single service deployment
  - Start a single Jupyter notebook server in a container
  - Start a single web server in a container
- An orchestrating service is needed for distributed application with multiple containers
  - ECS or EKS(based on Kubernetes)

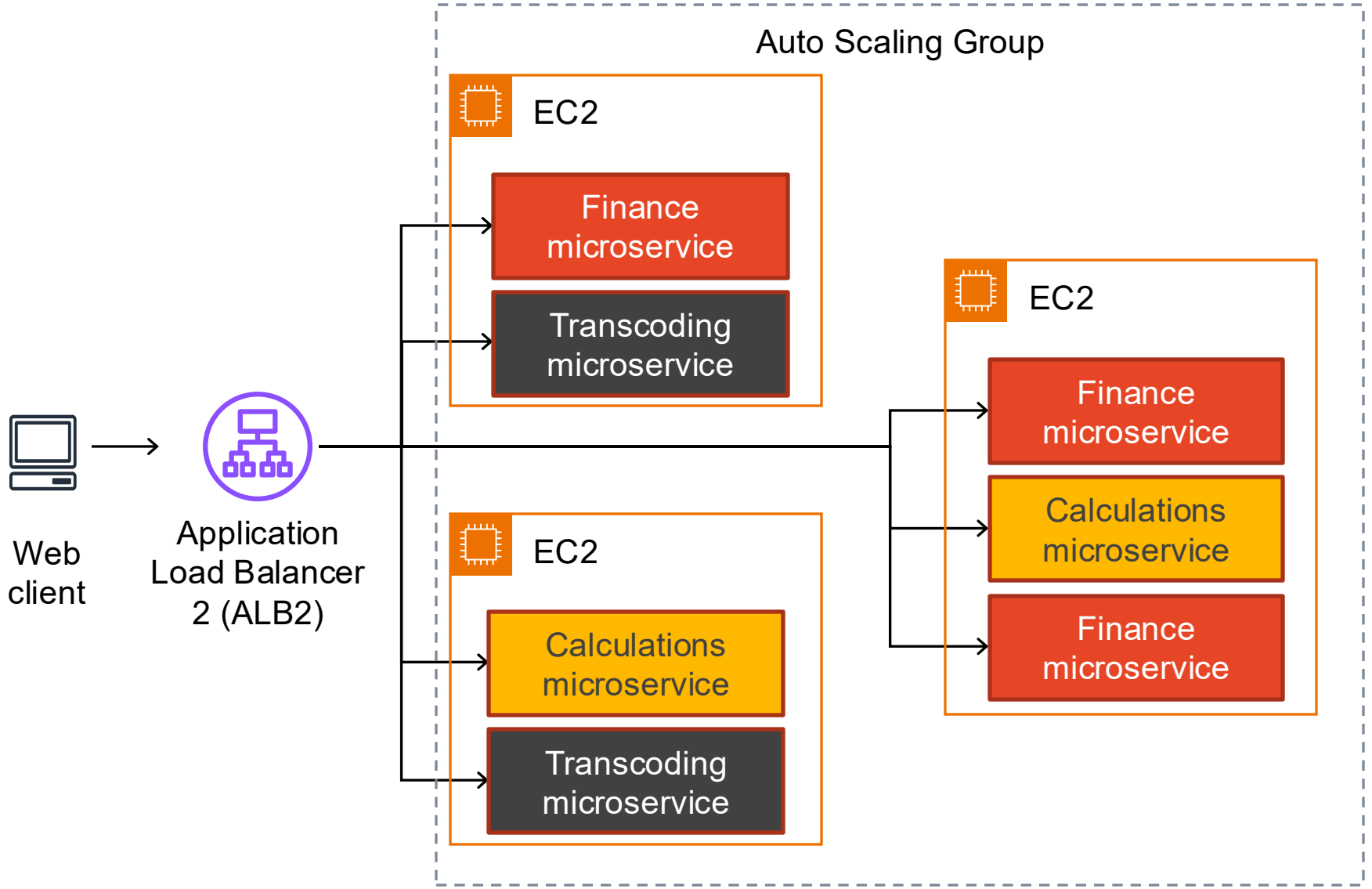


A not very useful microservices architecture



Instance level scaling and scheduling

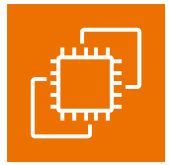
A more desirable architecture



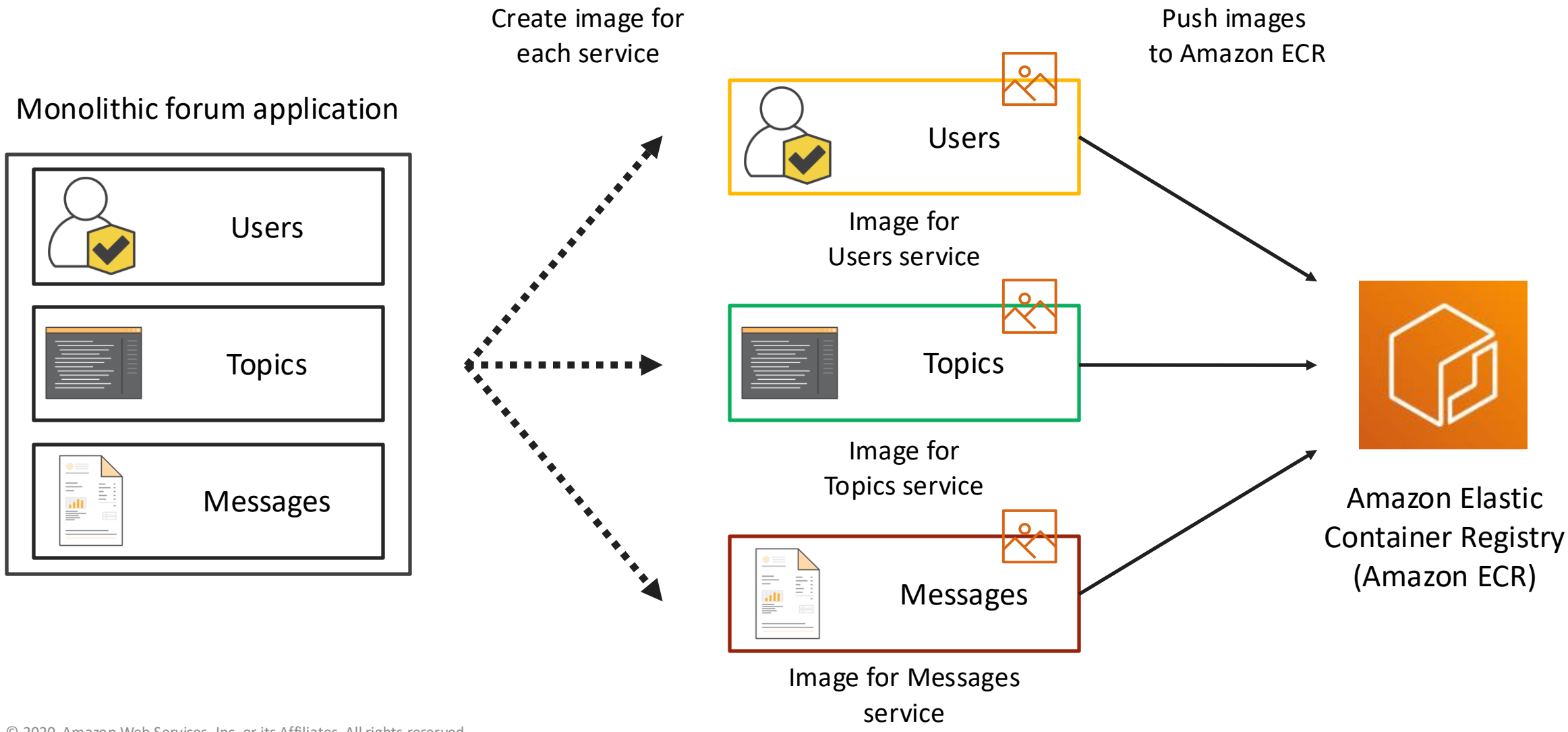
container level scaling and scheduling

Amazon Container Service

AWS container services

Registry	Orchestration	Compute options
		
Amazon Elastic Container Registry (Amazon ECR)	Amazon Elastic Container Service (Amazon ECS)	Amazon Elastic Compute Cloud (Amazon EC2)
		
	Amazon Elastic Kubernetes Service (Amazon EKS)	AWS Fargate
		
		AWS Lambda

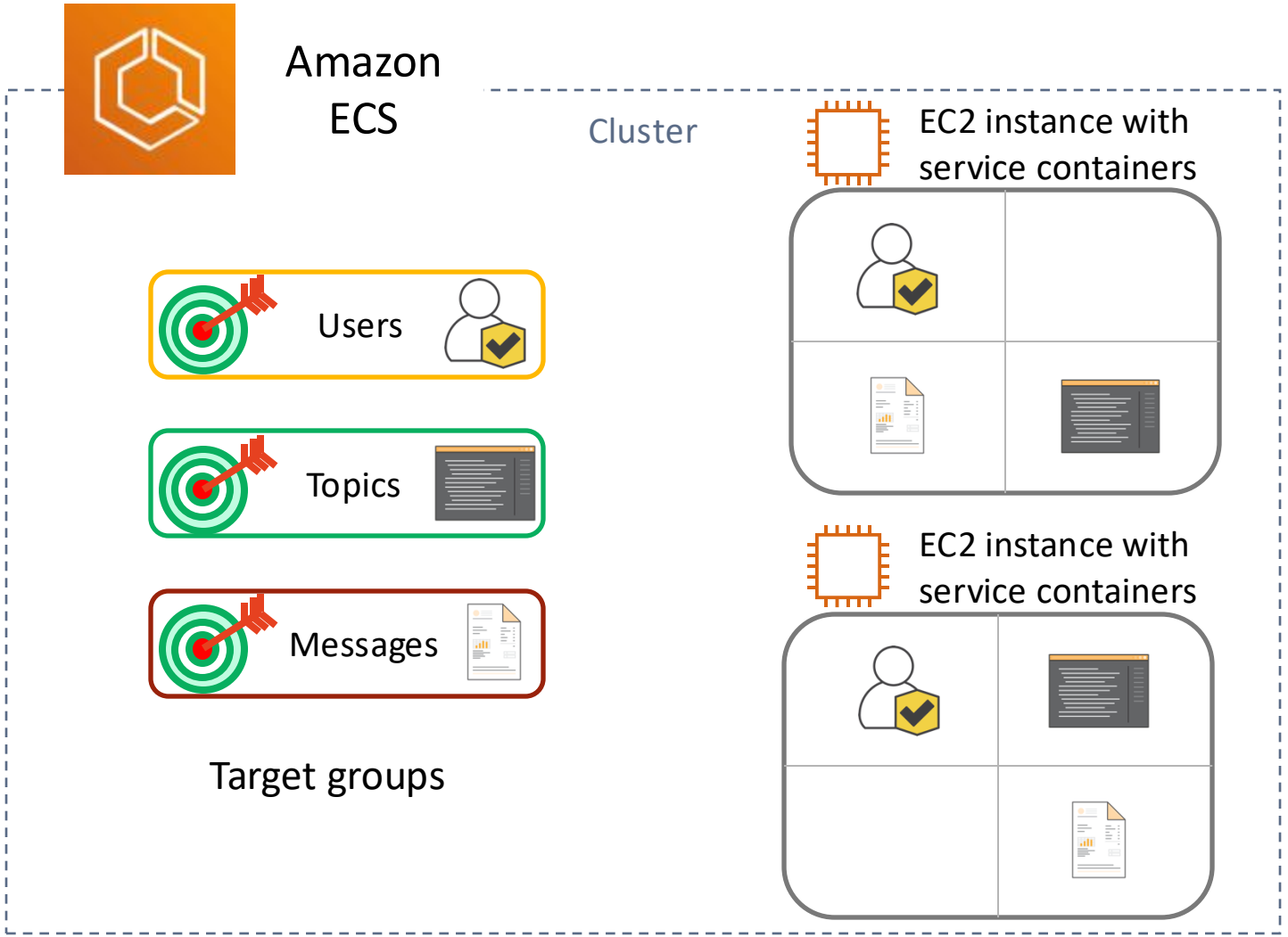
Decomposing monoliths – Step 1: Create container images



Decomposing monoliths – Step 2: Create service task definition and target groups

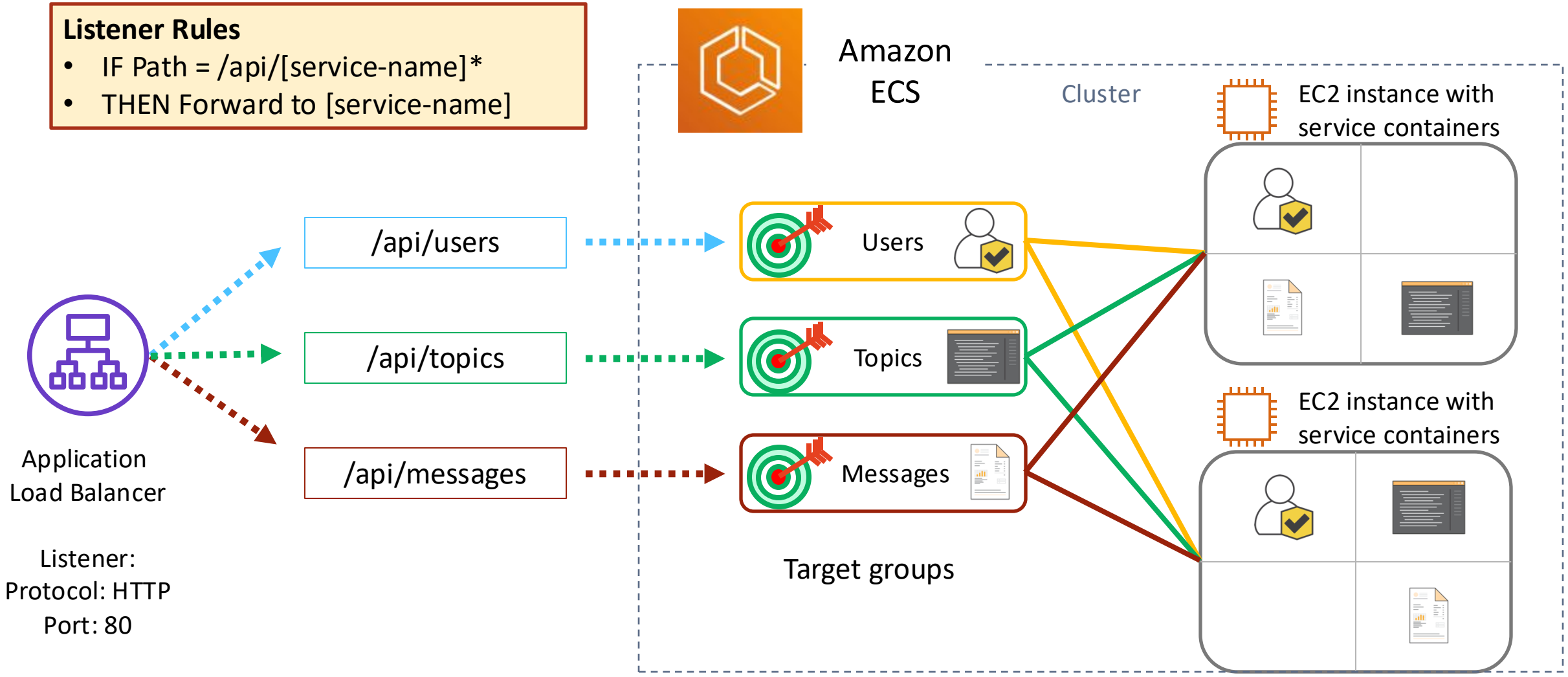
Service Task Definition
- Launch type = [EC2 or Fargate]
- Name = [service-name]
- Image = [service ECR repo URL]:version
- CPU = [256]
- Memory = [256]
- Container port = [3000]
- Host port = [0]

Service Target Group
- Name = [service-name]
- Protocol = [HTTP]
- Port = [80]
- VPC = [vpc-name]

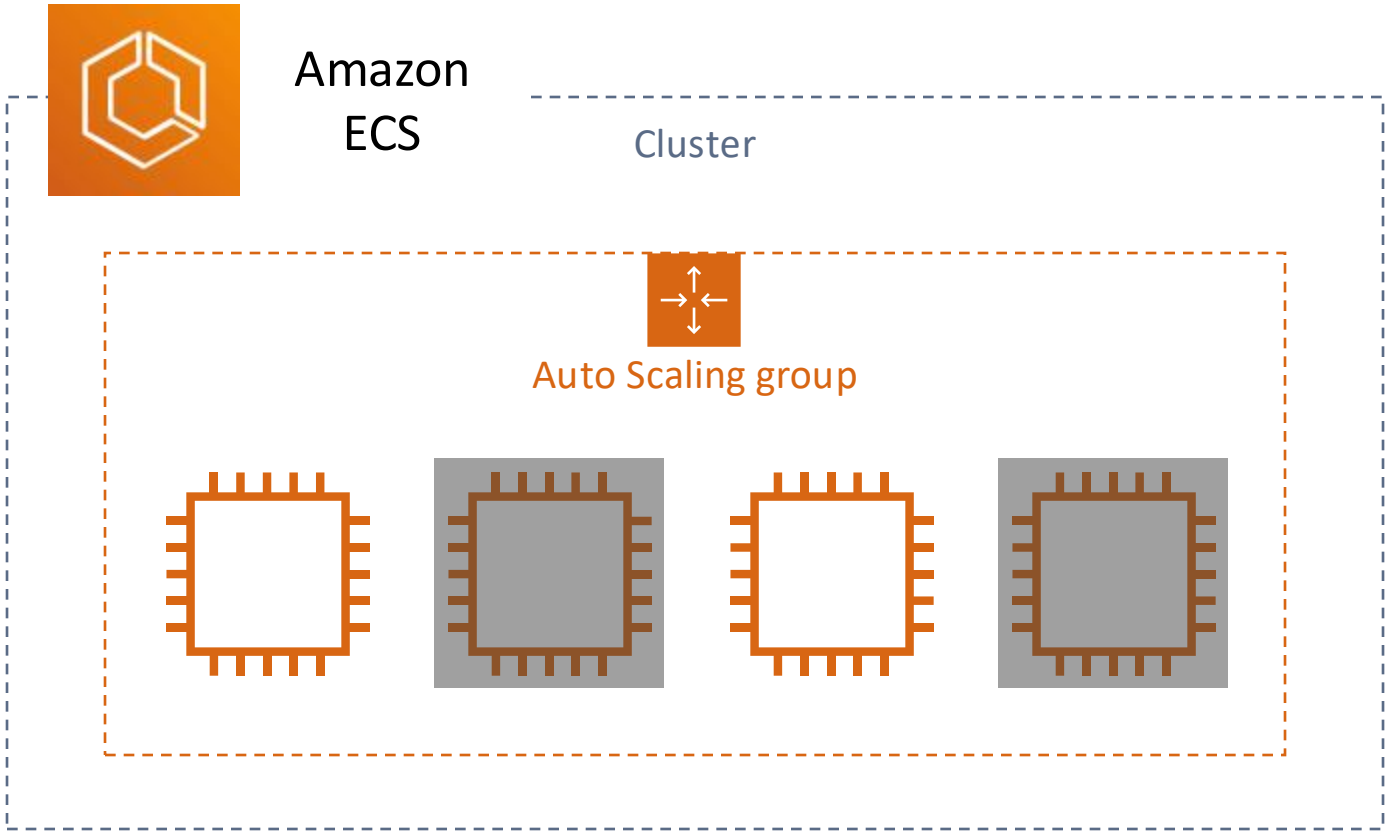


Decomposing monoliths – Step 3: Connect load balancer to services

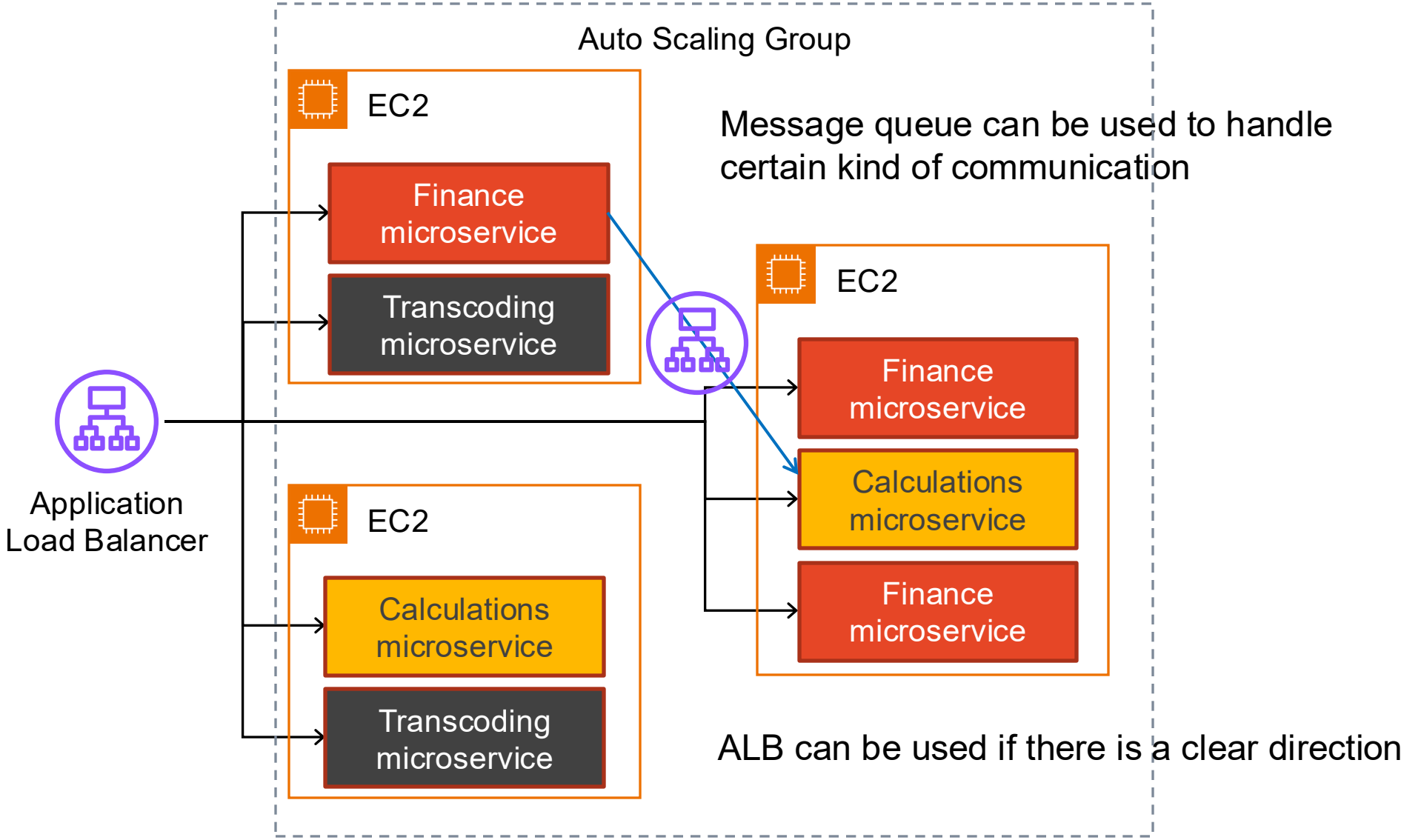
Listener Rules
- IF Path = /api/[service-name]*
- THEN Forward to [service-name]



Amazon ECS cluster auto scaling



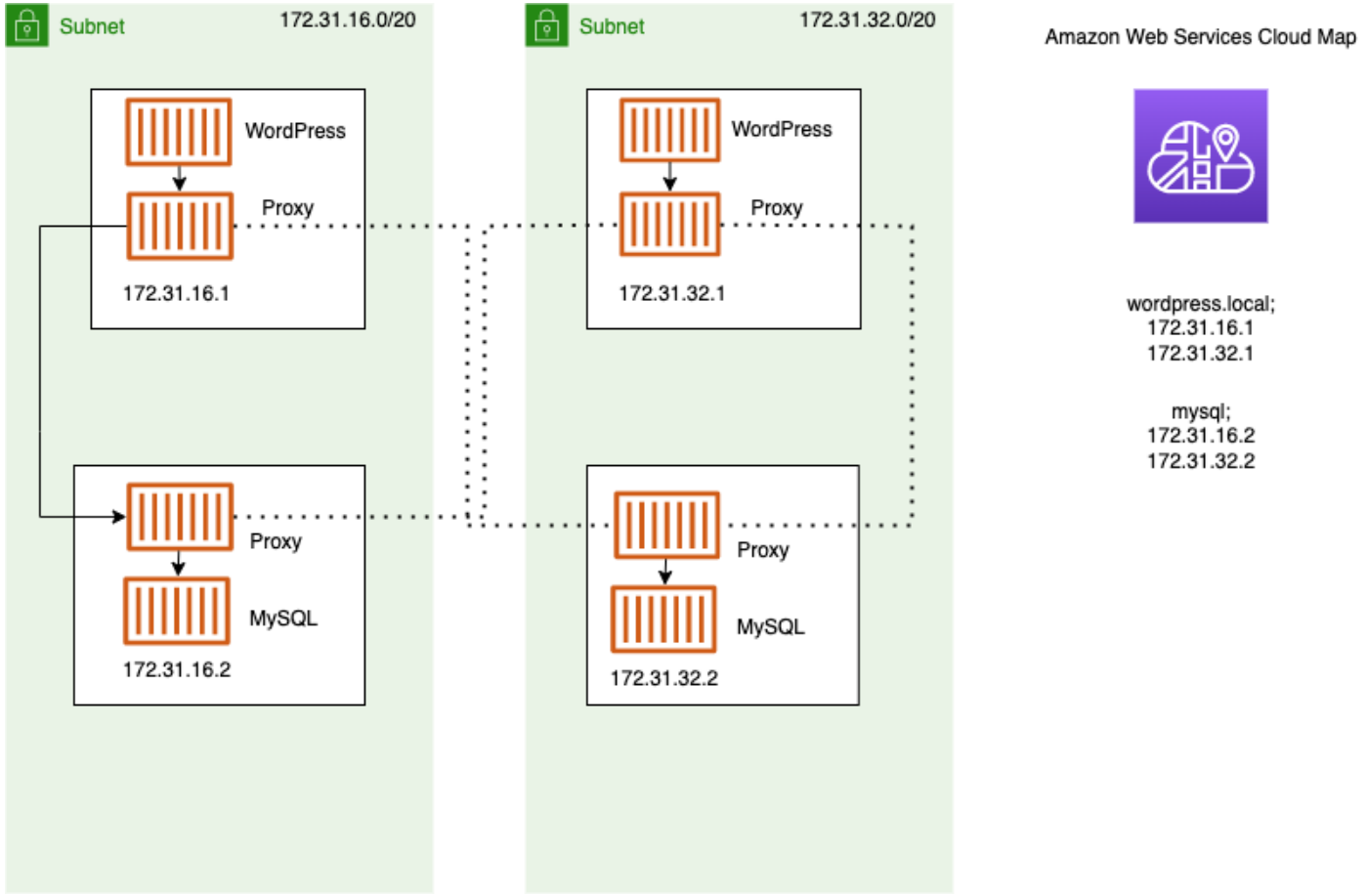
Inter-service communication



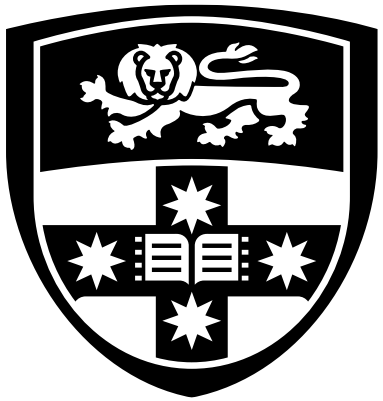
Inter-service communication

- ECS Service Discovery
  - Each service can automatically register itself with a DNS name in a CloudMap Namespace
  - Services referring to each other using the DNS name when they need to talk to each other, Route 53 helps to resolve the location
- ECS Service Connect
  - Each each service has a logical name registered with CloudMap
  - ECS Service Connect handles the routing
- AWS App Mesh
  - More advanced approach
  - App Mesh provides a dedicated infrastructure layer for service-to-service communication.

Service Connect Example



https://docs.aws.amazon.com/AmazonECS/latest/developerguide/service-connect.html



THE UNIVERSITY OF
SYDNEY