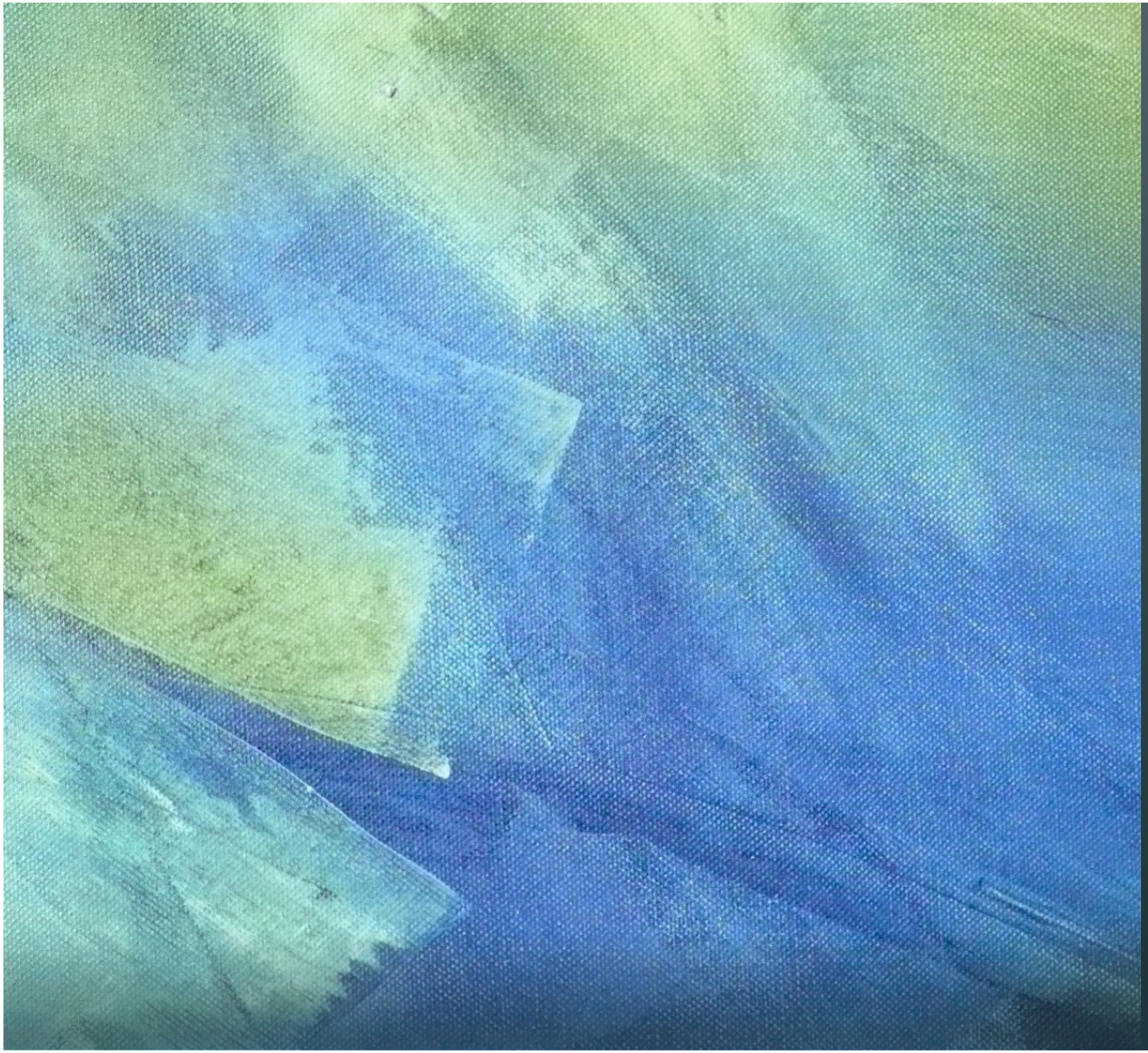


Deploying Microservices

Wiwat Vatanawood

Duangdao Wichadakul

Nuengwong Tuaycharoen

Pittipol Kantavat

The Chapter covers

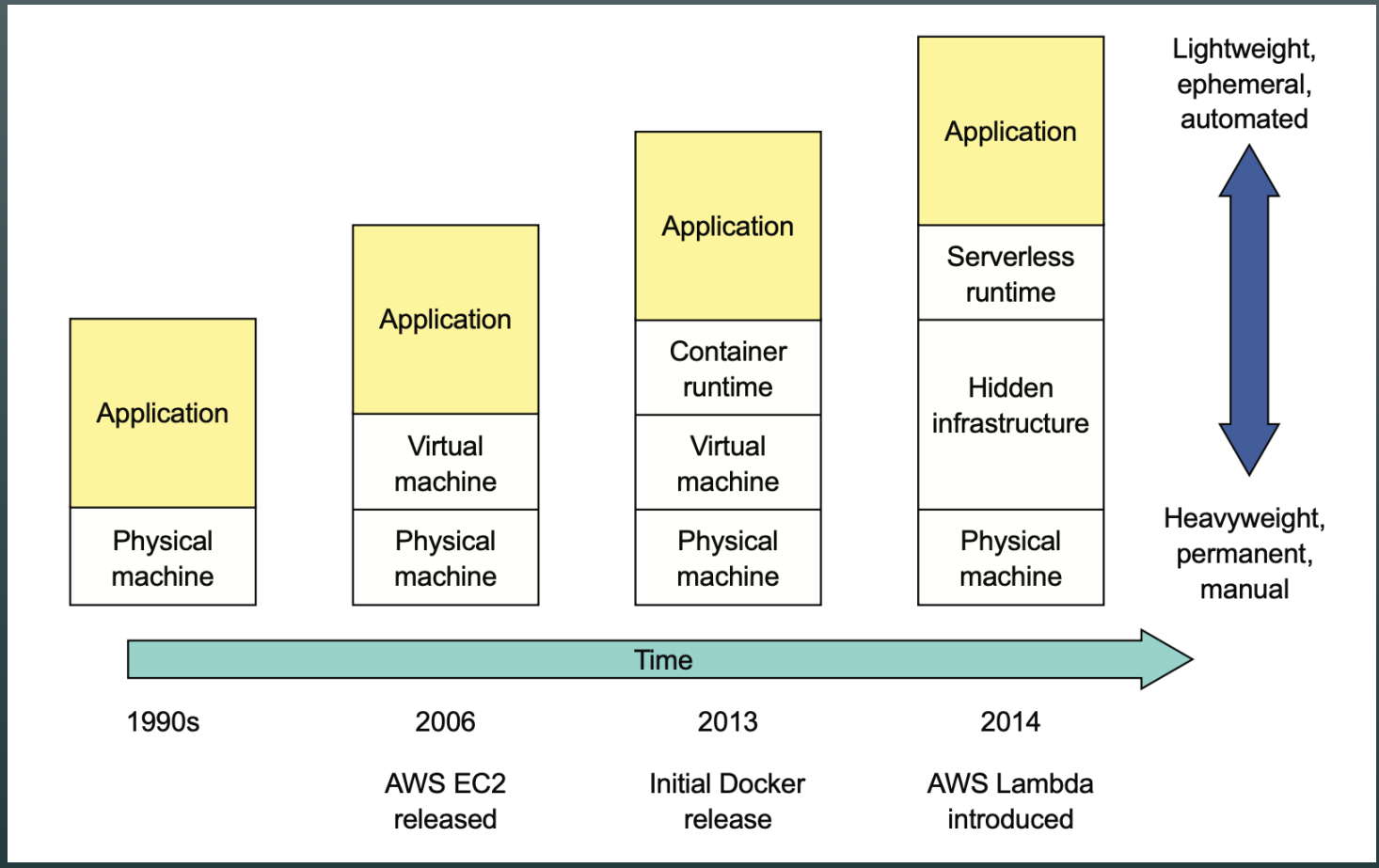
- The four key deployment patterns, benefits and drawbacks:
  - Language-specific packaging format
  - Deploying a service as a VM
  - Deploying a service as a container
  - Serverless deployment
- Deploying services with Kubernetes
- Using a service mesh to separate deployment
- Deploying services with AWS Lambda
- Picking a deployment pattern

Agenda

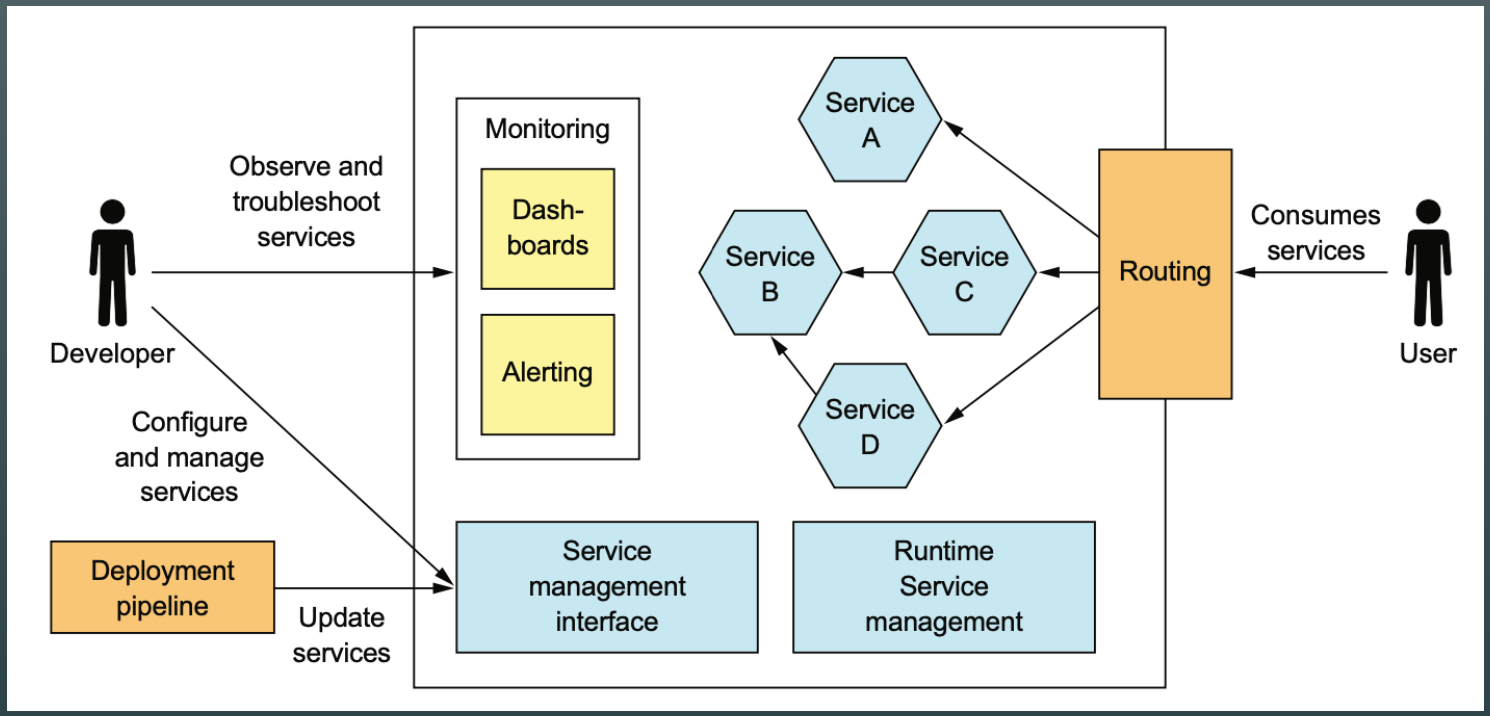
- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

An Overview of Deploying Microservice

- Deployment is a combination of two interrelated concepts: process and architecture.
  - **Process** consists of the steps that must be performed by people— developers and operations—in order to get software into production.
  - **Architecture** defines the structure of the environment in which that software runs.
- Both aspects of deployment have changed radically since 1990s.



- Heavyweight and long-lived physical machines has been abstracted away by increasingly lightweight and ephemeral technologies.



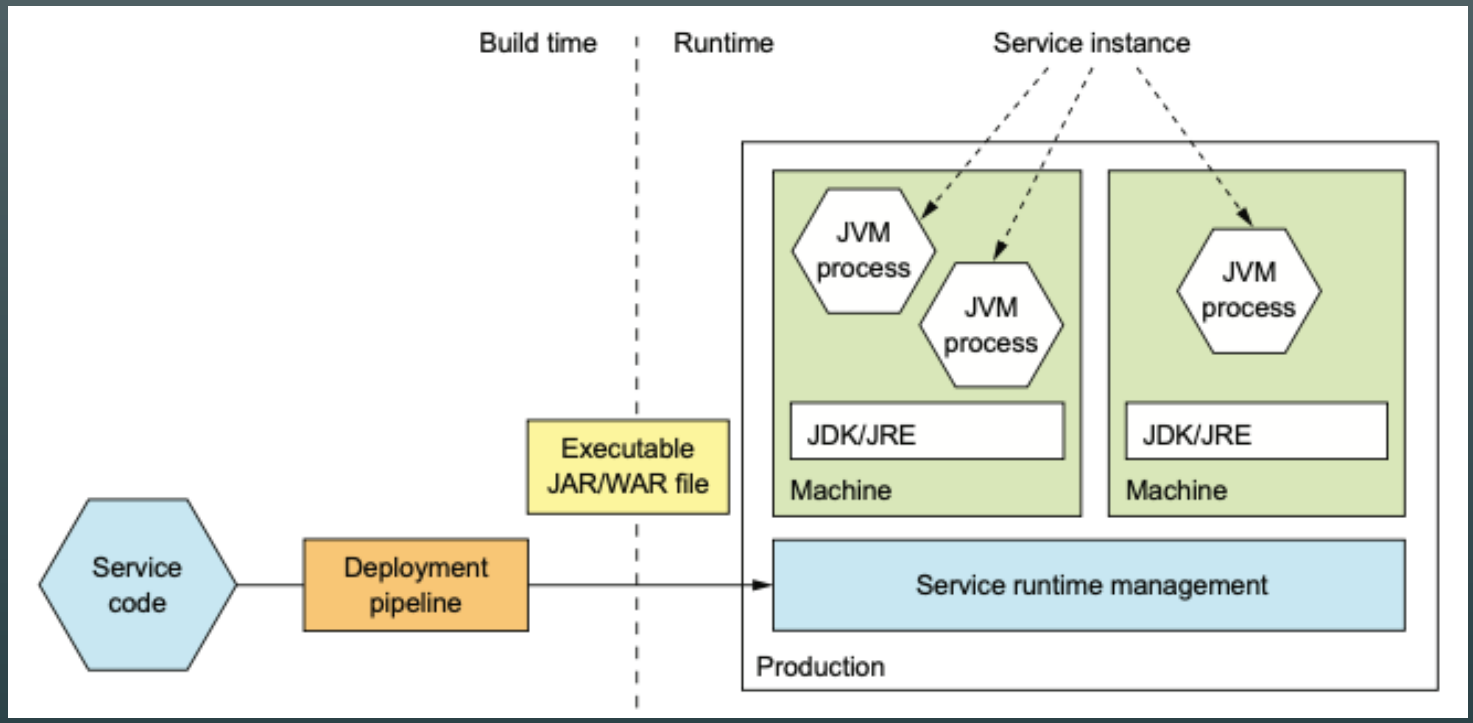
- A simplified view of the production environment.
  1. **Service management interface** enables developers to deploy and manage their services
  2. **Runtime service management** ensures that the services are running
  3. **Monitoring** visualizes service behavior and generates alerts
  4. **Request routing** routes requests from users to the services

Agenda

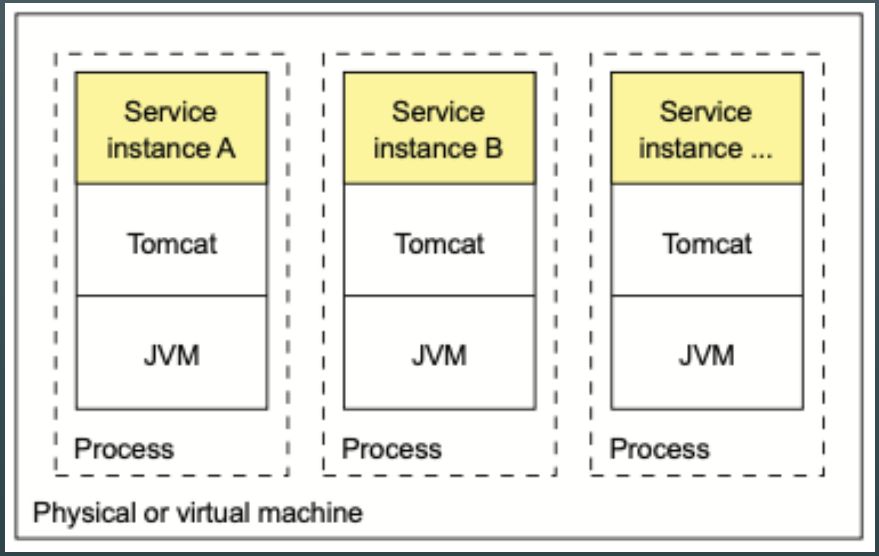
- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

Deploying services using the Language-specific packaging format pattern

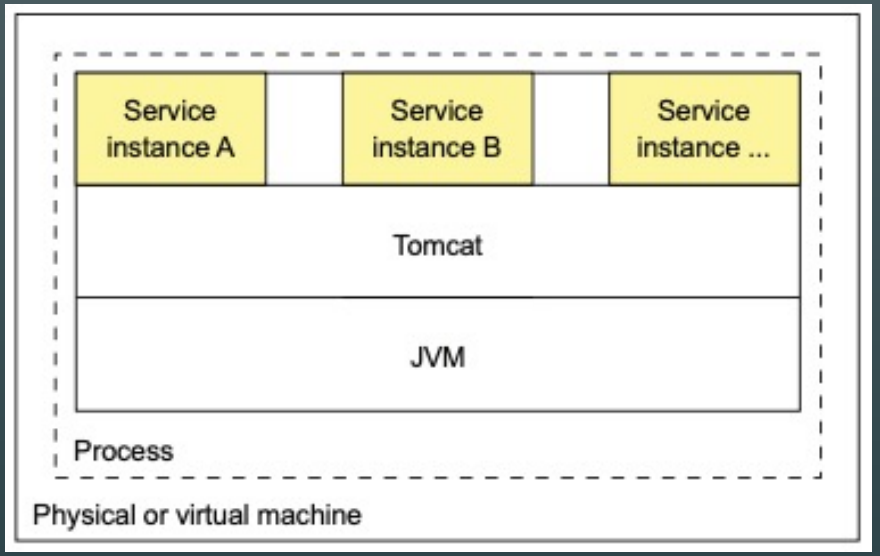
- What's deployed and what's managed by the service runtime is a service in its language-specific package
  - Spring Boot-based Java - executable JAR file or a WAR file
  - NodeJS - directory of source code and modules
  - GoLang - operating system-specific executable.



- For FTGO's Restaurant Service, which is a Spring Boot-based Java application.
  - The deployment pipeline builds an executable JAR file and deploys it into production.
  - In production, each service instance is a JVM running on a machine that has the JDK or JRE installed.



- Deploying multiple service instances on the same machine.



- Deploying multiple services instances on the same web container or application server.

Benefits and Drawbacks

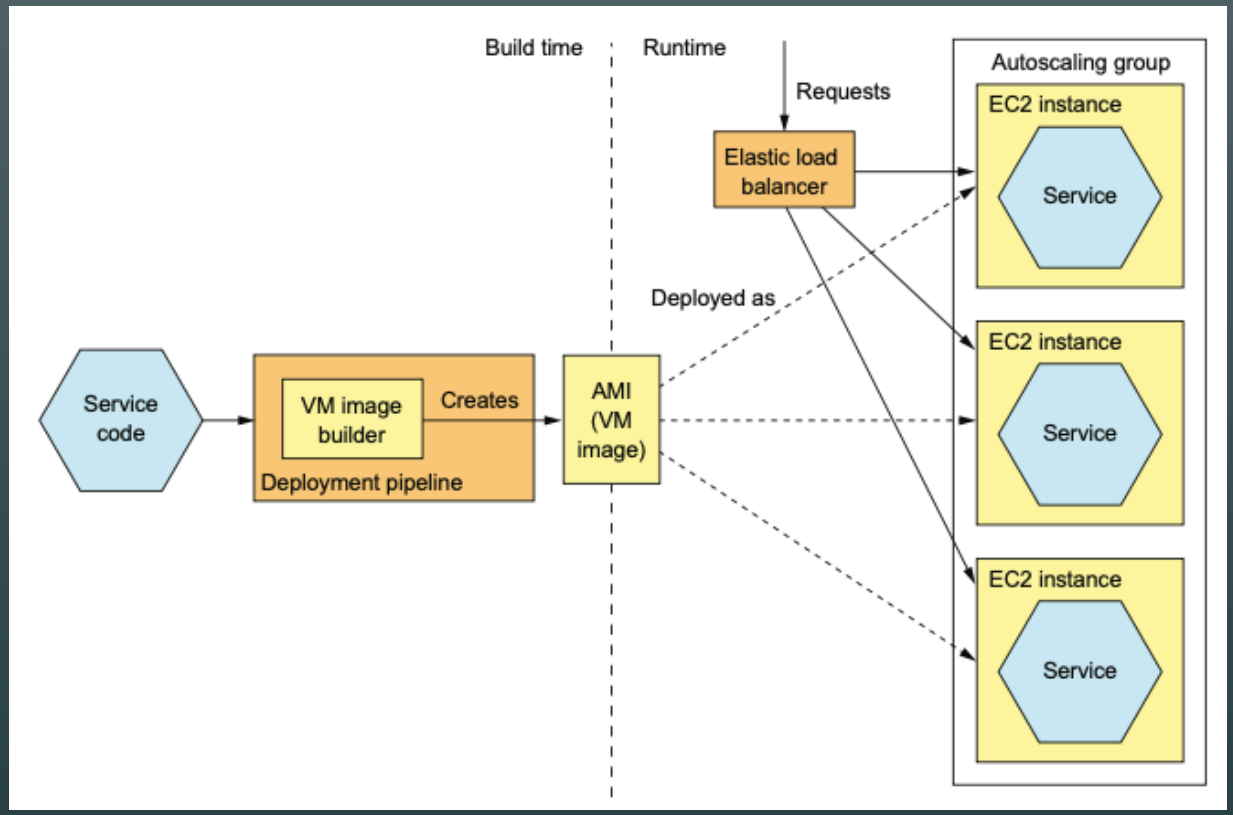
- Benefits
  - Fast deployment
  - Efficient resource utilization
- Drawbacks
  - Lack of encapsulation of the technology stack
  - No ability to constrain the resources consumed by a service instance
  - Lack of isolation when running multiple service instances on the same machine
  - Automatically determining where to place service instances is challenging

Agenda

- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

Deploying services using the Service as a virtual machine pattern

- For the FTGO, one option would be :
  - Create and configure an EC2 instance and
  - Copy onto it the executable or WAR file.
- A better option is to package the service as an Amazon Machine Image (AMI)
  - Each service instance is an EC2 instance created from that AMI.
  - The EC2 instances would typically be managed by an AWS Auto Scaling group,
  - The desired number of healthy instances is always running.



- The deployment pipeline packages a service as a VM image, such as an EC2 AMI.
- At runtime, each service instance is a VM, such as an EC2 instance.
- An EC2 Elastic Load Balancer routes requests to the instances.

Benefits and Drawbacks

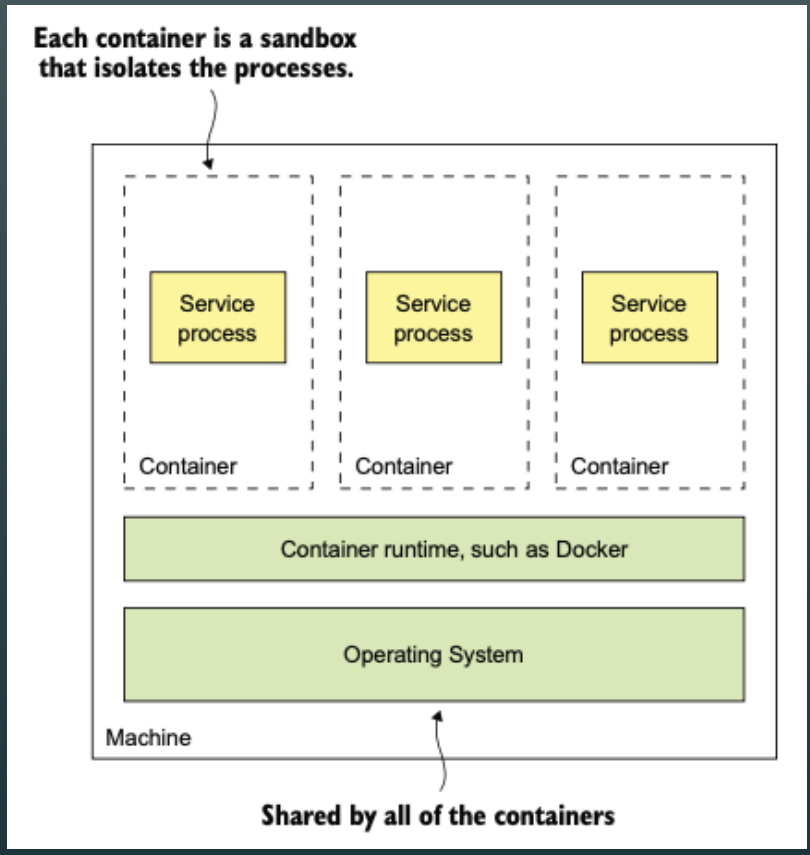
- Benefits
  - The VM image encapsulates the technology stack
  - Service instances are isolated
  - Uses mature cloud infrastructure
- Drawbacks
  - Less-efficient resource utilization
  - Relatively slow deployments
  - System administration overhead

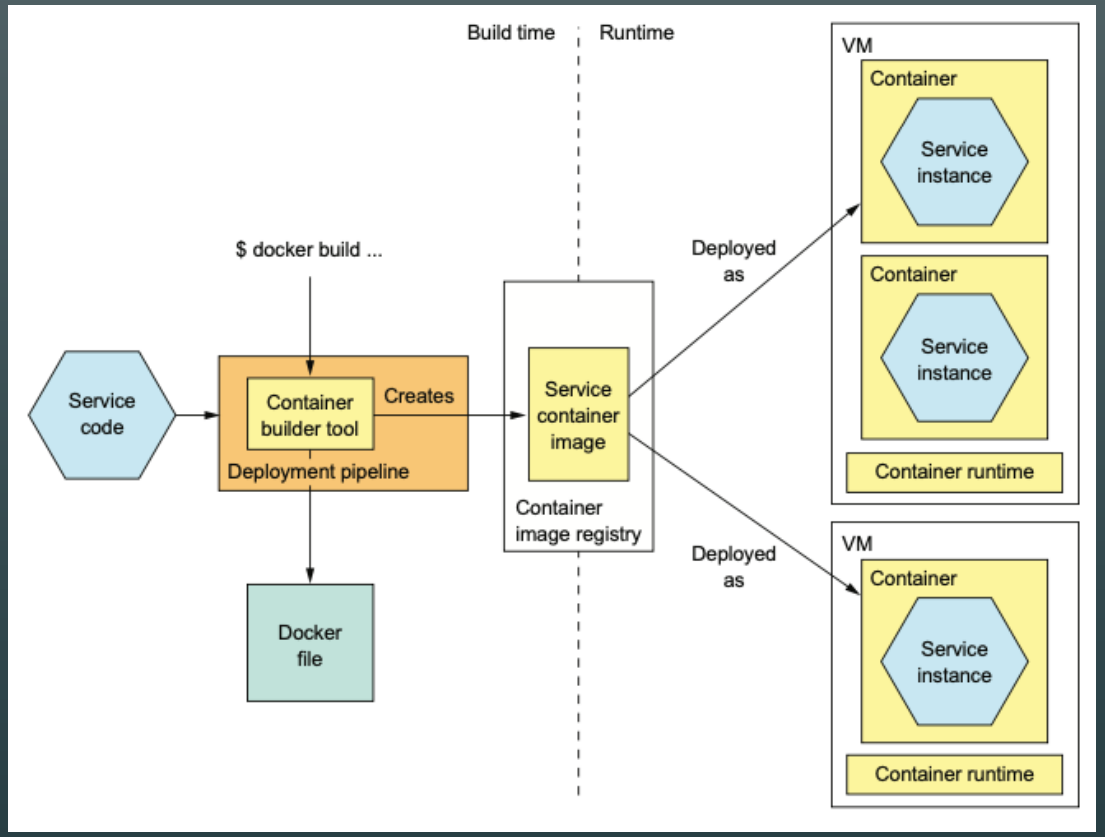
Agenda

- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

Deploying services using the Service as a container pattern

- Containers are a more modern and lightweight deployment mechanism.
- A container consists of one or more processes running in an isolated sandbox.
- Multiple containers usually run on a single machine.
- The containers share the operating system.





- A service is packaged as a container image
- At runtime the service consists of multiple containers instantiated from that image.
- A single VM will usually run multiple containers.

Benefits and Drawbacks

- Benefits
  - Many benefits of VM but more lightweight
  - Typically build faster than VM
- Drawbacks
  - Responsible for the heavy lifting of administering the container images, patching the operating system and runtime.
  - Also administering the container infrastructure and possibly the VM infrastructure it runs on.

Agenda

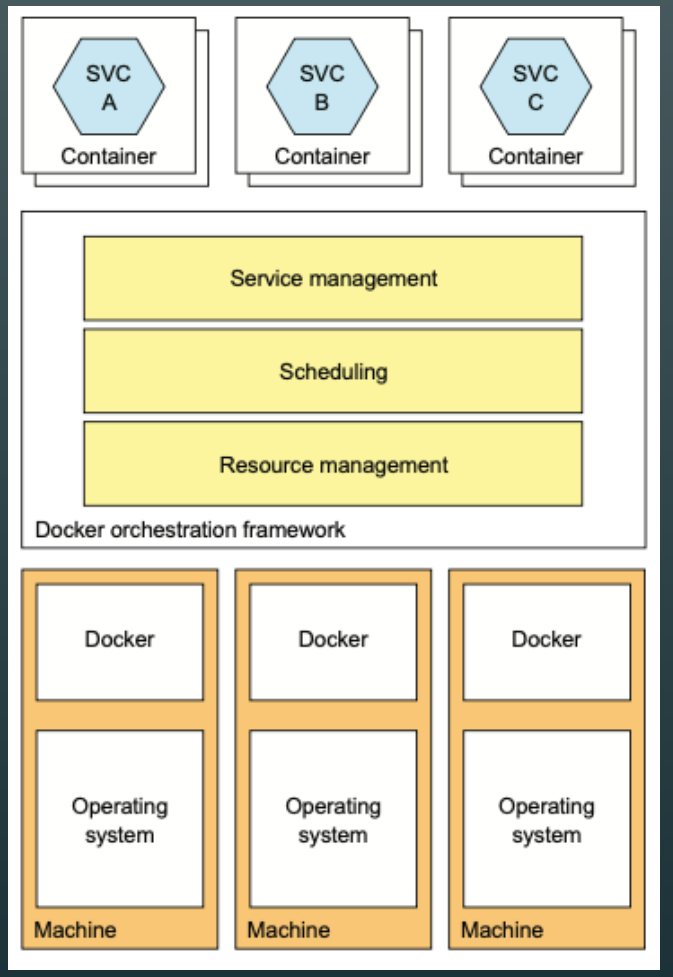
- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

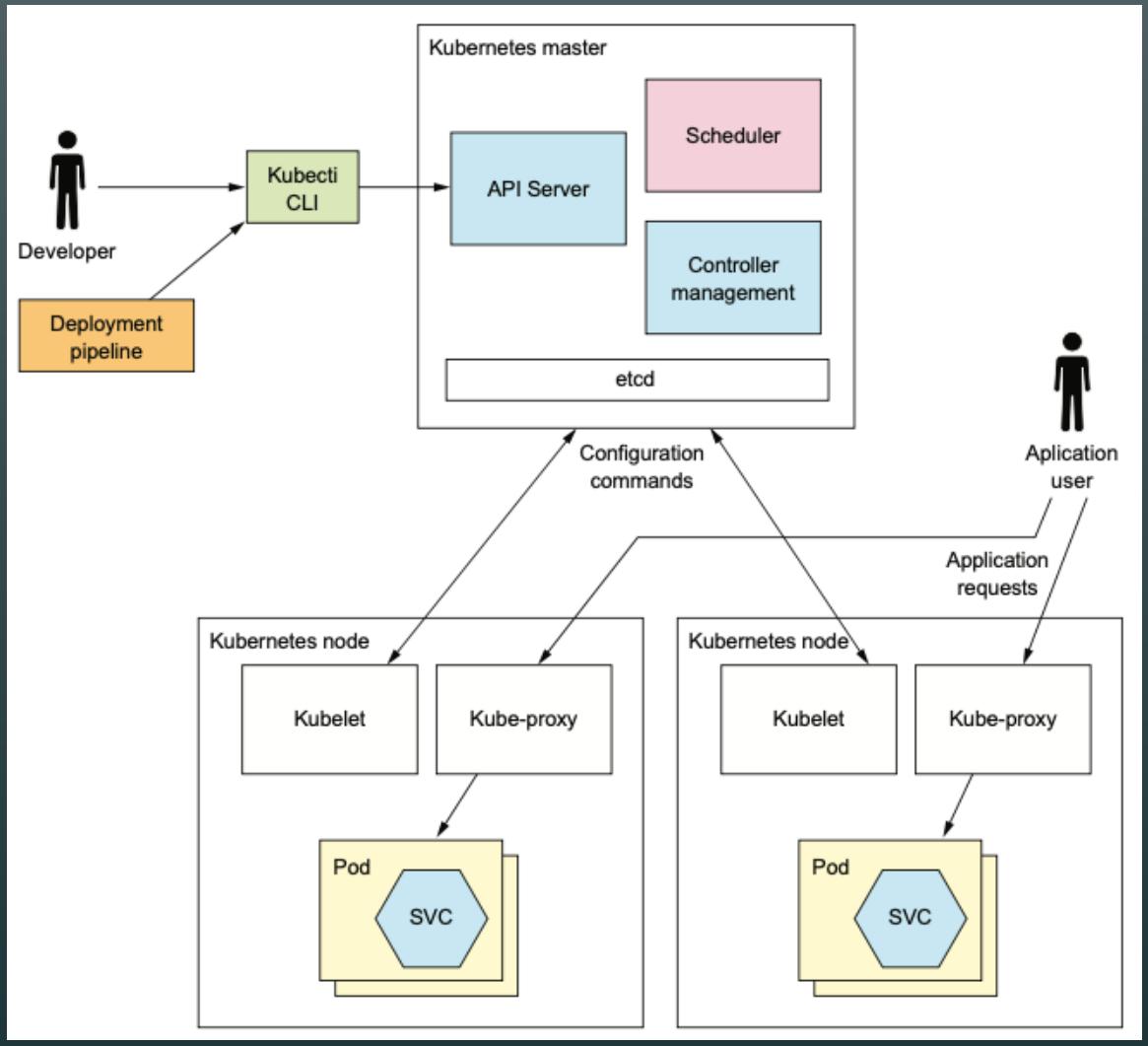
Deploying services using Kubernetes

- Kubernetes is a Docker orchestration framework,
  - A layer of software on top of Docker that turns a set of machines into a single pool of resources for running services.
- The desired number of instances of each service keeps running at all times,
  - Even when service instances or machines crash.

A Docker orchestration framework

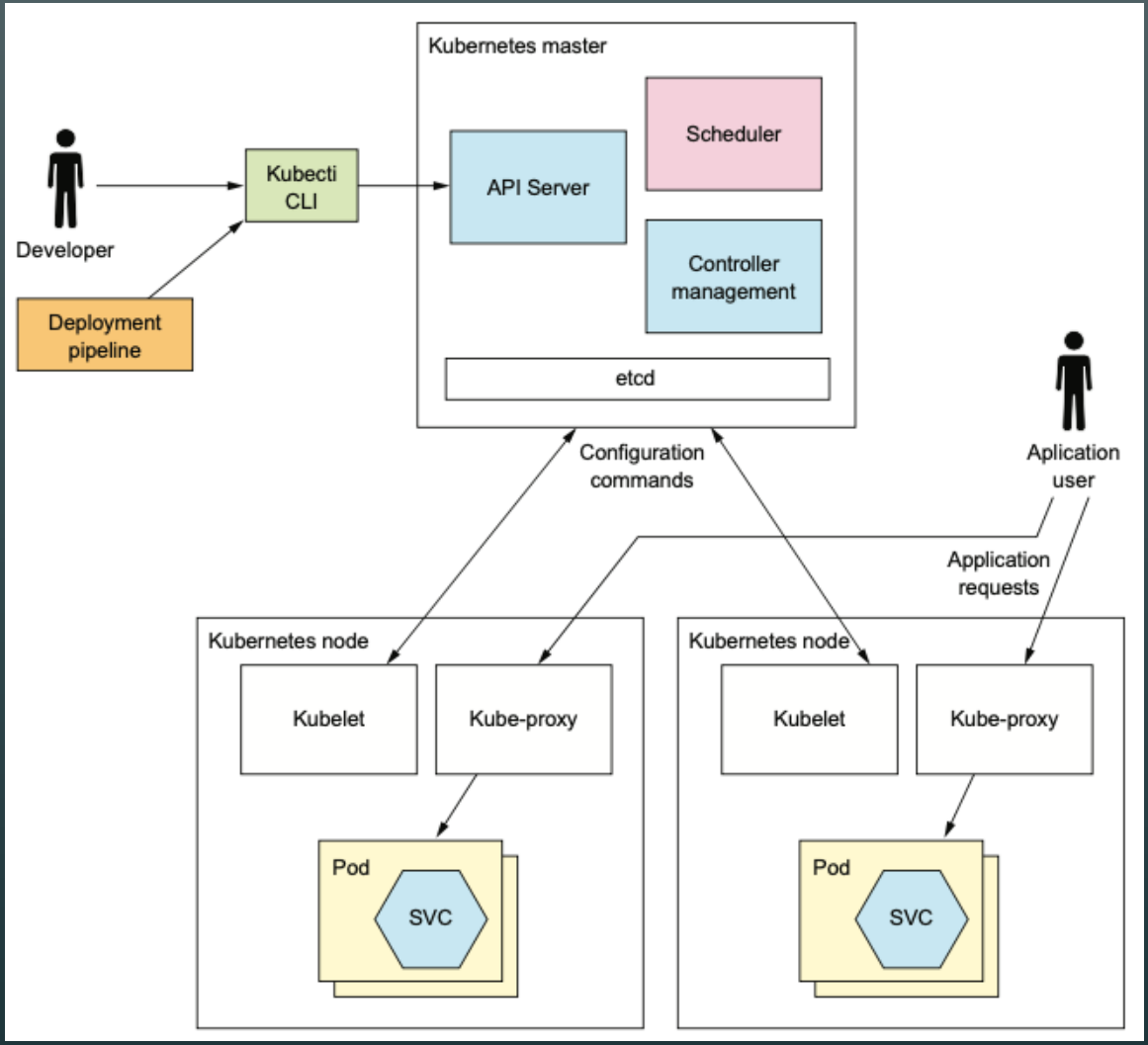
- There are three main functions:
  - Resource management—Treats a cluster of machines as a pool of CPU, memory, and storage volumes
  - Scheduling—Selects the machine to run your container
  - Service management—Implements the concept of named and versioned services that map directly to services in the microservice architecture.





Kubernetes Architecture

- A Kubernetes master runs:
  - API Server—The REST API for deploying and managing services
  - Etcd—A key-value NoSQL database that stores the cluster data.
  - Scheduler—Selects a node to run a pod.
  - Controller manager—ensure that the state of the cluster matches the intended state.



Kubernetes Architecture (cont.)

- A Kubernetes node runs:
  - Kubelet—Creates and manages the pods running on the node
  - Kube-proxy—Manages networking, including load balancing across pods
  - Pods—The application services

Using a service mesh to separate deployment from release

- A reliable way to roll out a new version is to separate deployment from the release:
  - Deployment—Running in the production environment
  - Releasing a service—Making it available to end-users
- Then deploy a service into production using the following steps:
  1. Deploy the new version into production without routing any end-user requests to it.
  2. Test it in production.
  3. Release it to a small number of end-users.
  4. Incrementally release it to an increasingly larger number of users until it's handling all the production traffic.
  5. If at any point there's an issue, revert to the old version

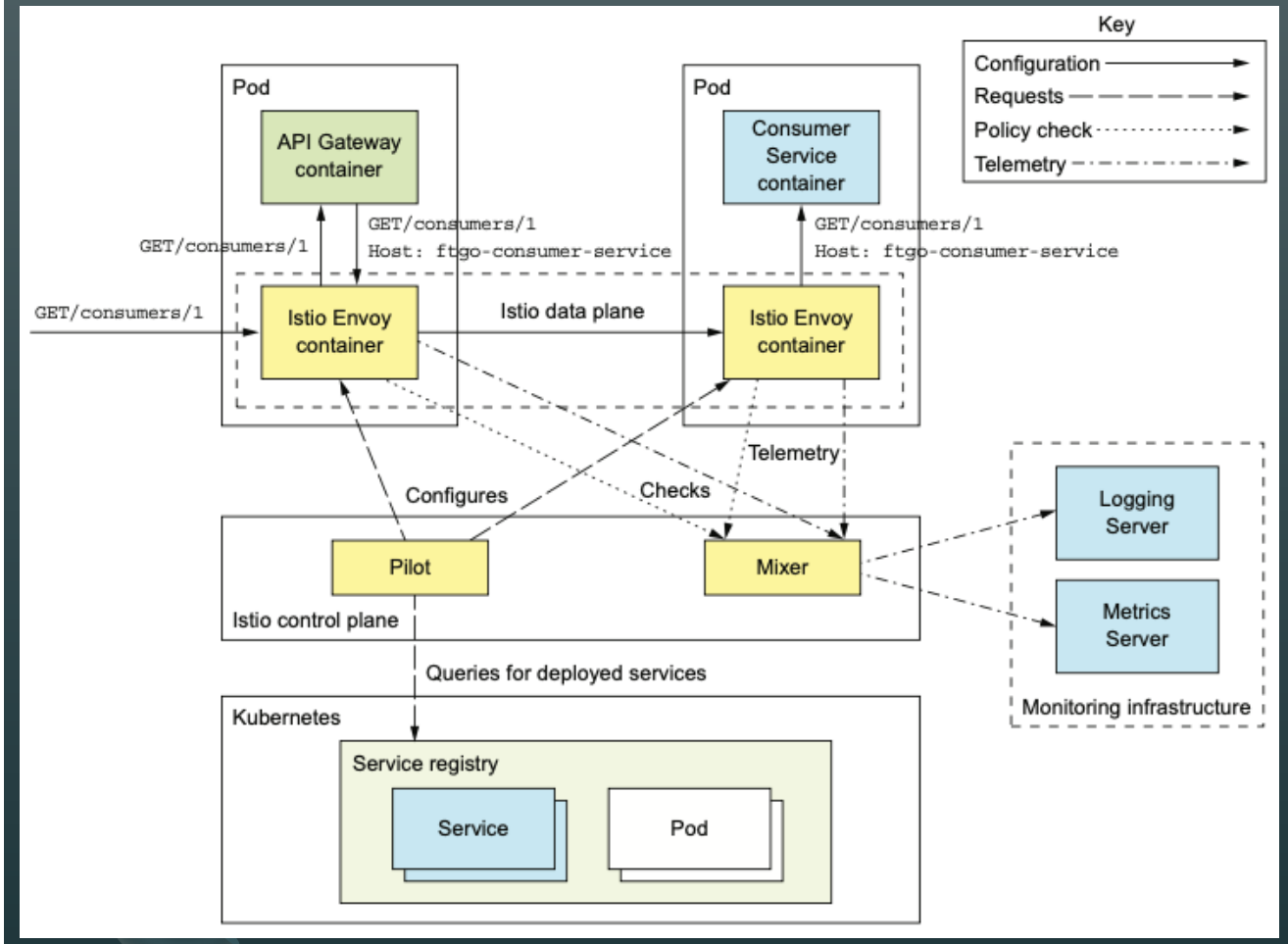
Using a service mesh to separate deployment from release (cont.)

- Traditionally, separating deployments and releases in this way requires a lot of work to implement.
- But using a service mesh makes this style of deployment is a lot easier.
- A service mesh is a networking infrastructure that mediates all communication between a service and other services and external applications.
- A service mesh provides rule-based load balancing and traffic routing
  - let you safely run multiple versions of your services simultaneously.

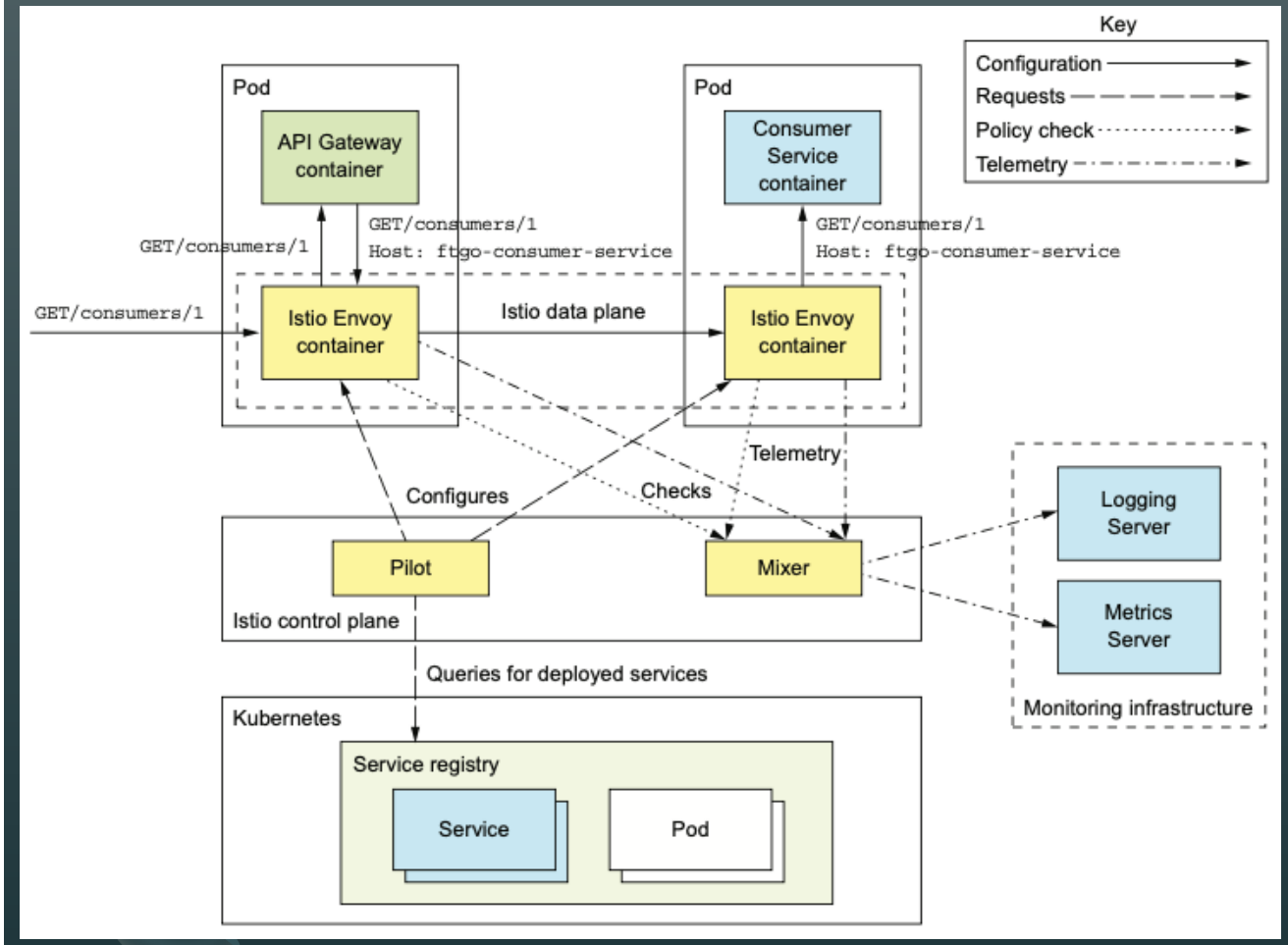
Overview of the Istio Service Mesh

- Istio as an "An open platform to connect, manage, and secure microservices" (https://istio.io)
- Istio has a rich set of features organized into four main categories:
  - Traffic management—Includes service discovery, load balancing, routing rules, and circuit breakers
  - Security—Secures interservice communication using Transport Layer Security (TLS)
  - Telemetry—Captures metrics about network traffic and implements distributed tracing
  - Policy enforcement—Enforces quotas and rate limits

Istio Architecture



- It consists of a control plane and a data plane.
  - The control plane implements management functions, including configuring the data plane to route traffic.
  - The data plane consists of Envoy proxies, one per service instance.



Istio Architecture (cont.)

- The two main components of the control plane are the Pilot and the Mixer.
  - The Pilot extracts information about deployed services from the underlying infrastructure.
  - The Mixer collects telemetry from the Envoy proxies and enforces policies.

Agenda

- An overview of Deploying Microservice
- Deploying services using the Language-specific packaging format pattern
- Deploying services using the Service as a virtual machine pattern
- Deploying services using the Service as a container pattern
- Deploying services using Kubernetes
- Deploying services using the Serverless deployment pattern

“Undifferentiated Heavy Lifting”

- “Heavy Lifting” was introduced by Amazon.
- Referring to pre-provision some computing resources—either physical machines, virtual machines, or containers.
  - The Language-specific packaging / Service as a VM / Service as a container
- There’s a solution : “serverless”.

Overview of Serverless Deployment

- AWS Lambda is an example of serverless deployment technology.
- AWS Lambda was initially for deploying event-driven services.
- AWS Lambda automatically runs enough instances of your microservice to handle incoming requests.
  - Billed for each request based on the time taken and the memory consumed.
- No needs to worry about any aspect of servers, virtual machines, or containers!!

Benefits of AWS Lambda

- Integrated with many AWS services
- Eliminates many system administration tasks
- Elasticity—No need to predict the capacity of VMs or containers.
- Usage-based pricing

Drawbacks of AWS Lambda

- Long-tail latency
  - Because AWS Lambda dynamically runs your code,
  - Some requests have high latency because of the time it takes for AWS to provision an instance of your application to start.
- Limited event/request-based programming model
  - AWS Lambda isn't intended to be used to deploy long-running services,
  - Such as a service that consumes messages from a third-party message broker.

Summary (1)

- You should choose the most lightweight deployment pattern that supports your service's requirements.
- Evaluate the options in the following order:
  1. serverless
  2. containers
  3. virtual machines
  4. language-specific packages

Summary (2)

- A serverless deployment eliminates the need to administer operating systems and runtimes.
- It provides automated elastic provisioning and request-based pricing.
- A serverless deployment isn't a good fit for every service.
  - Long-tail latencies
  - Required to use an event/request-based programming model.

Summary (3)

- Docker containers, a lightweight, OS-level virtualization technology, are more flexible than serverless deployment and have more predictable latency.
- It's best to use a Docker orchestration framework such as Kubernetes.
- The drawback is that you must administer the operating systems and runtimes.

Summary (4)

- Deploying your service as a virtual machine is a heavyweight deployment option and will most likely use more resources than Docker containers.
- Deploying your services as language-specific packages is generally best avoided unless you only have a small number of services.