QUEUE STRUCTURES

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

WHO AM I?

Co-Founder & CEO @Summarify

Senior Data Scientist @ Hepsiburada

Summarify

kafkave

hepsiburada

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

OUTLINE

01 WHAT IS QUEQUE

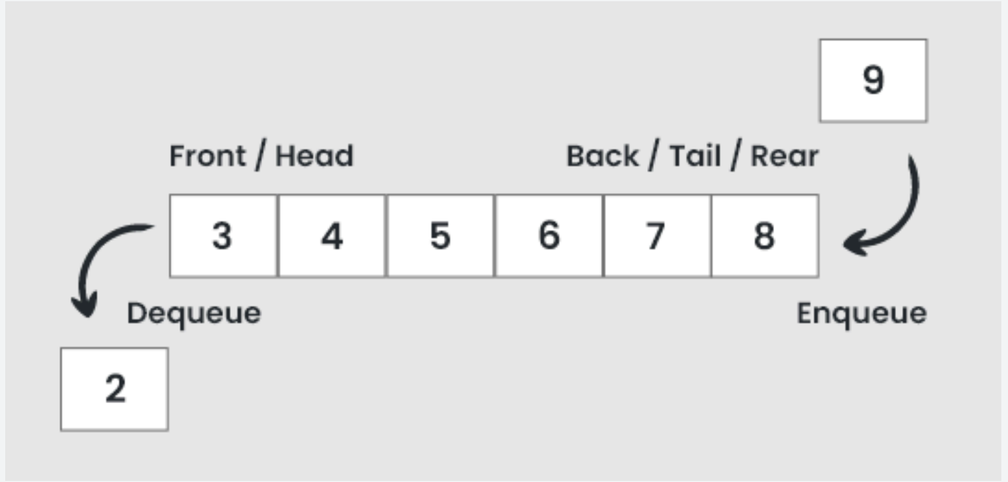
02 KAFKA, RABBİTMQ, REDIS

03 COMPARISON AND USAGE EXAMPLES

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

KUYRUK YAPISI NEDİR?

The **queue structure** (queue) is a simple and effective data structure that ensures data is processed in sequence. This structure operates on the principle that the data entered first is processed first **(FIFO: First In, First Out)**. In other words, the element added to the front of the queue is the first to be processed or removed.



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

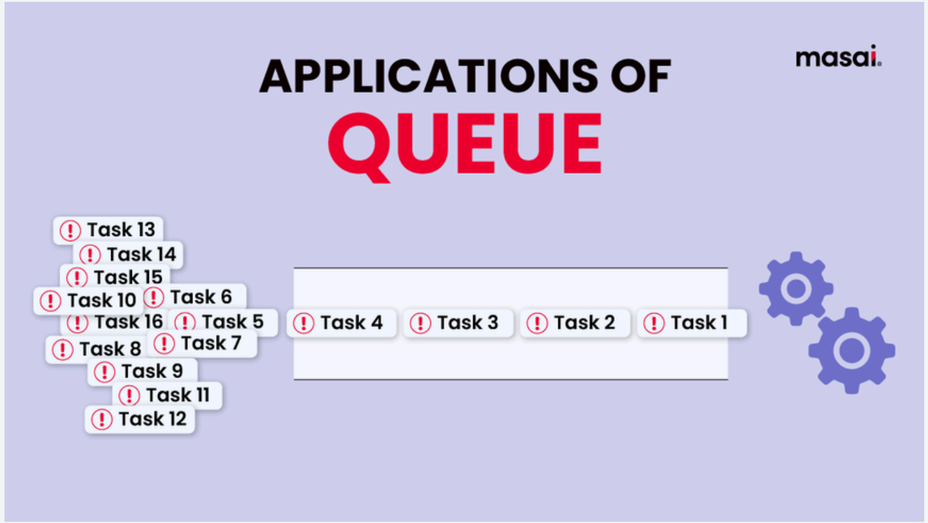
APPLICATIONS IF QUEQUE

Case: On a popular e-commerce site, especially during campaign periods, thousands of users try to place orders simultaneously. This situation creates a significant load on order processing and inventory management.

- **Order Management:** When users place orders, these orders are queued. The accuracy of the orders, stock status, and payment processes are handled in sequence within the queue. This prevents the system from crashing and ensures that each order is processed properly.

Case: Social media platforms must process actions such as posts, comments, and likes in real-time. These platforms need to manage the interactions of millions of users simultaneously.

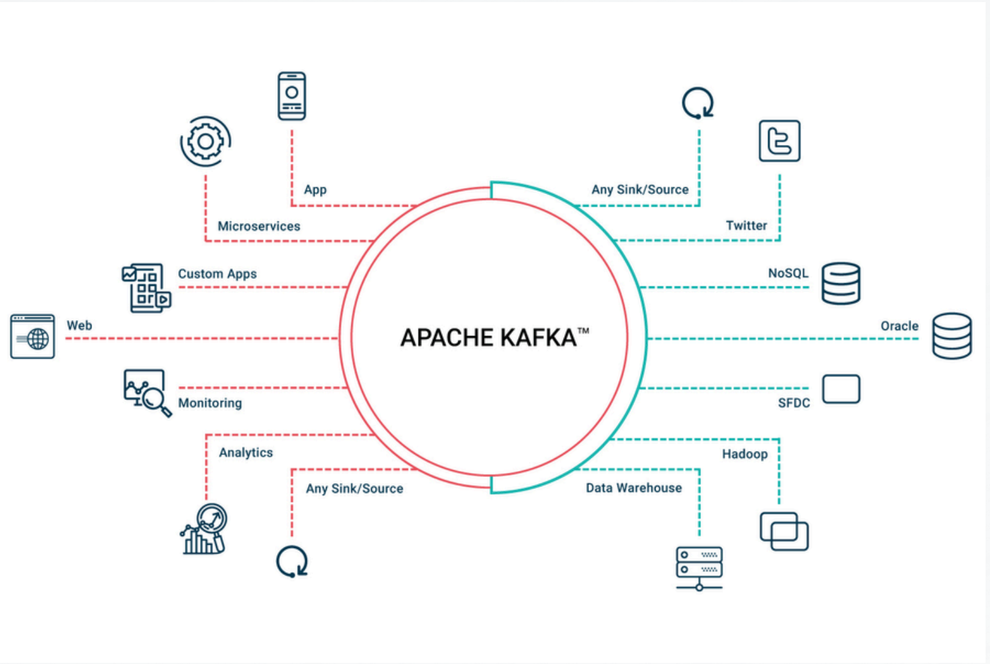
- **Post Processing:** When users share a post, these posts are queued and then published. On large-scale social media platforms, this structure balances the server load and ensures timely publication of posts.



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

WHAT IS kafka

Kafka is a distributed stream processing platform developed by LinkedIn and later open-sourced by the Apache Software Foundation. Kafka is particularly used for managing real-time data streams and is known for its ability to handle high volumes of data. By providing a distributed, log-based structure, it enables reliable and scalable data processing.

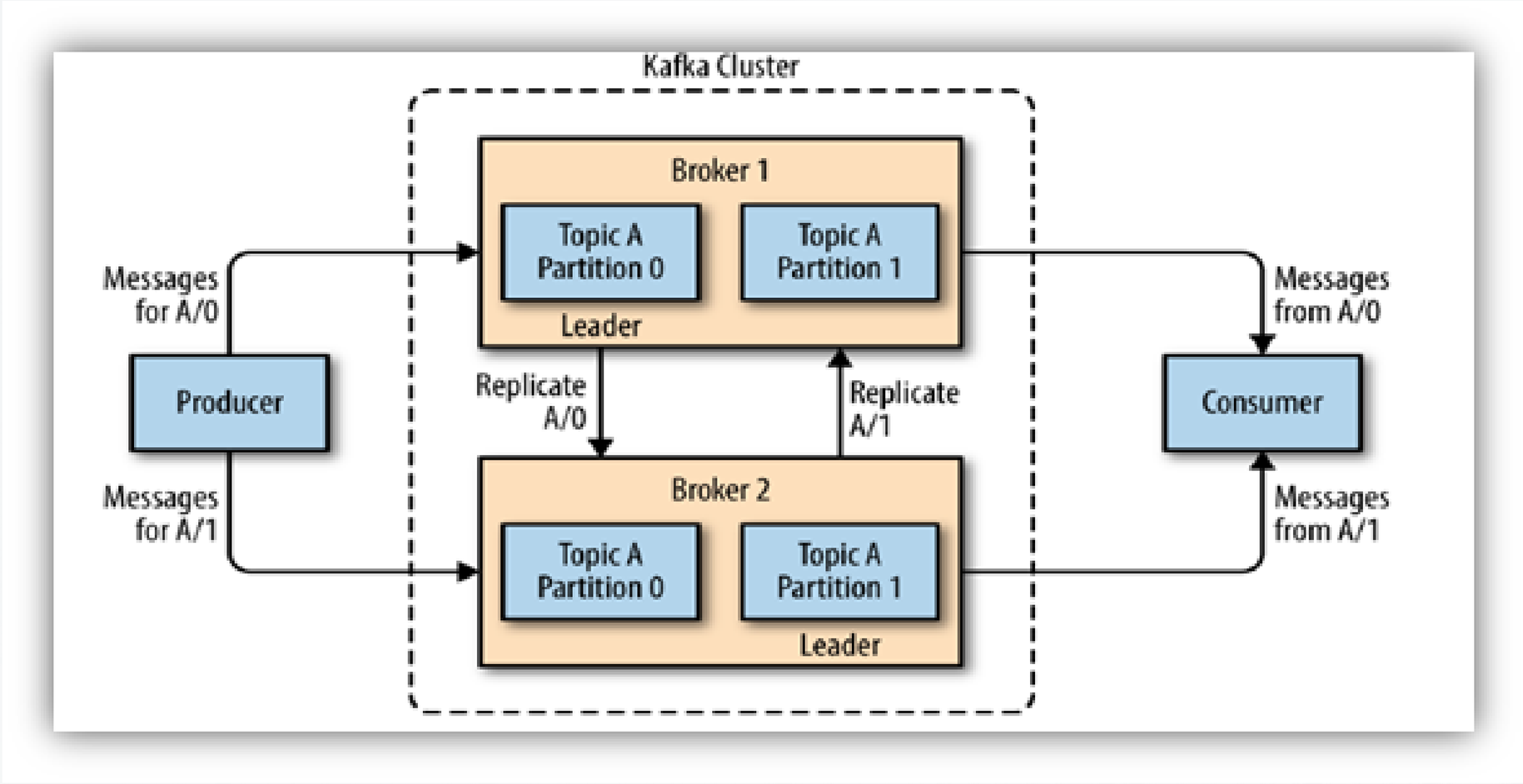


- Distributed and Log-Based Structure
- Easy Integration and Use
- Comprehensive Ecosystem Support

- Higher Configuration Requirements
- Mandatory Persistence Requirement
- Large Storage Needs

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

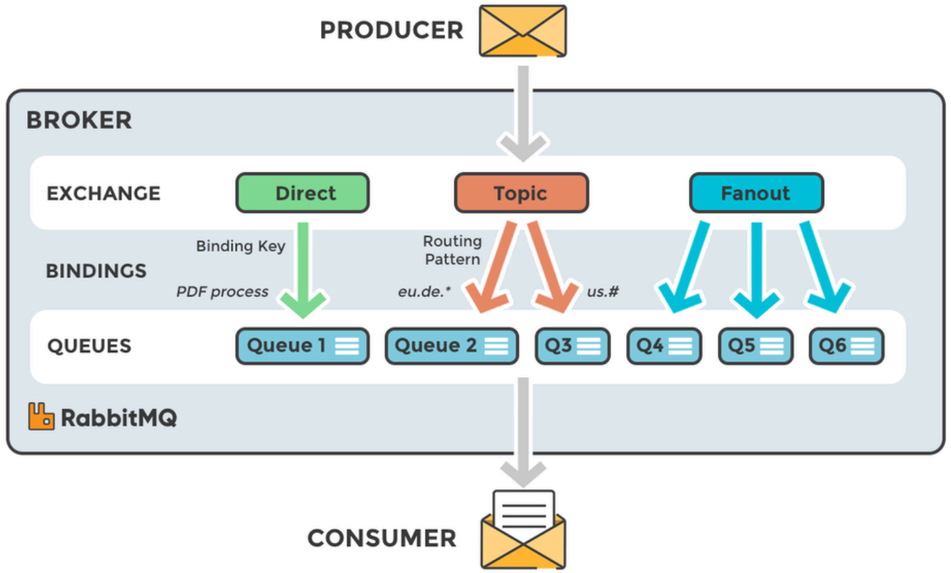
KAFKA



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

WHAT IS RabbitMQ

RabbitMQ is a lightweight and high-performance open-source message broker used for message queuing. Built on the AMQP (Advanced Message Queuing Protocol) protocol, RabbitMQ ensures the secure routing of messages. Thanks to its flexible structure, it supports a wide range of messaging scenarios and is frequently preferred in microservice architectures.



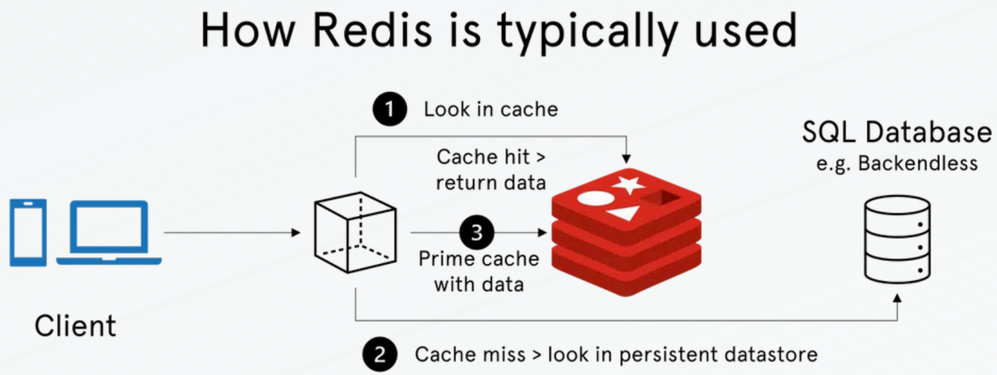
- Flexible Routing Mechanism
- Extensible Plugin Support
- Ideal for Microservice Architectures

- Performance Degradation Under High Traffic
- More Complex Configuration Requirements
- Short-Term Message Retention

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

WHAT IS redis

Redis is a high-performance, open-source database and queue system where data is stored in memory. It offers a key-value based structure and is commonly used in applications that require rapid data access. Redis is not only a database but can also function as a messaging queue system.



- Extremely Fast Data Access
- Flexible Data Structures
- Simple and Easy to Use

- Requires Additional Configuration for Data Persistence
- Limited Scalability
- Insufficient for Large-Scale Data Storage

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

KAFKA / RABBITMQ / REDIS

Özellik	Kafka	RabbitMQ	Redis
Kuyruk Modeli	Dağıtık, log tabanlı	Geleneksel kuyruk	Anahtar-değer tabanlı, basit kuyruk
Mesaj Saklama	Uzun süreli, disk tabanlı, isteğe bağlı olarak sonsuza kadar saklanabilir	Kısa süreli, genellikle bellek tabanlı	Kısa süreli, bellek tabanlı, kalıcı disk depolama yok
Tüketici Tipi	Çoklu tüketicili, konum takibi yapılabilir	Tüketici grubu, round-robin dağıtım	Basit tüketici modeli, genellikle FIFO
Performans	Yüksek throughput, düşük gecikme	Orta seviye throughput, düşük gecikme	Çok yüksek performans, çok düşük gecikme
Zaman Uyumsuz İşleme	Evet, doğrudan desteklenir	Evet, doğrudan desteklenir	Temel seviyede destek
Yük Dengeleme	Partisyon tabanlı, veri dağılımı sağlar	Mesaj yönlendirme, routing key ile	Temel, tek sunucuda sınırlı
Kalıcılık	Disk üzerine kalıcı yazma	Kalıcılık isteğe bağlı, disk üzerine yazma	Bellek tabanlı, kalıcılık için snapshot kullanılabilir
Mesaj Dağıtımı	Yayın, abonelik modeli, partisyonlar	Kuyruk tabanlı, yönlendirme anahtarı	Basit kuyruğa alım ve çıkış
Mesaj Sıralama	Partisyon bazında sıralama	FIFO, mesaj sıralama yok	FIFO sıralama, tek sunucu
Ölçeklenebilirlik	Yüksek, yatay ve dikey ölçeklenebilir	Yatay ölçeklenebilir, belirli limitler	Yatay ölçeklenebilir, Redis Cluster ile
Kullanım Alanları	Büyük veri analitiği, akış işleme, olay odaklı mimari	Mikro servisler, mesajlaşma, olay odaklı sistemler	Geçici veri saklama, basit mesajlaşma, caching

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

DEMO

KUBERNETES

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

OUTLINE

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

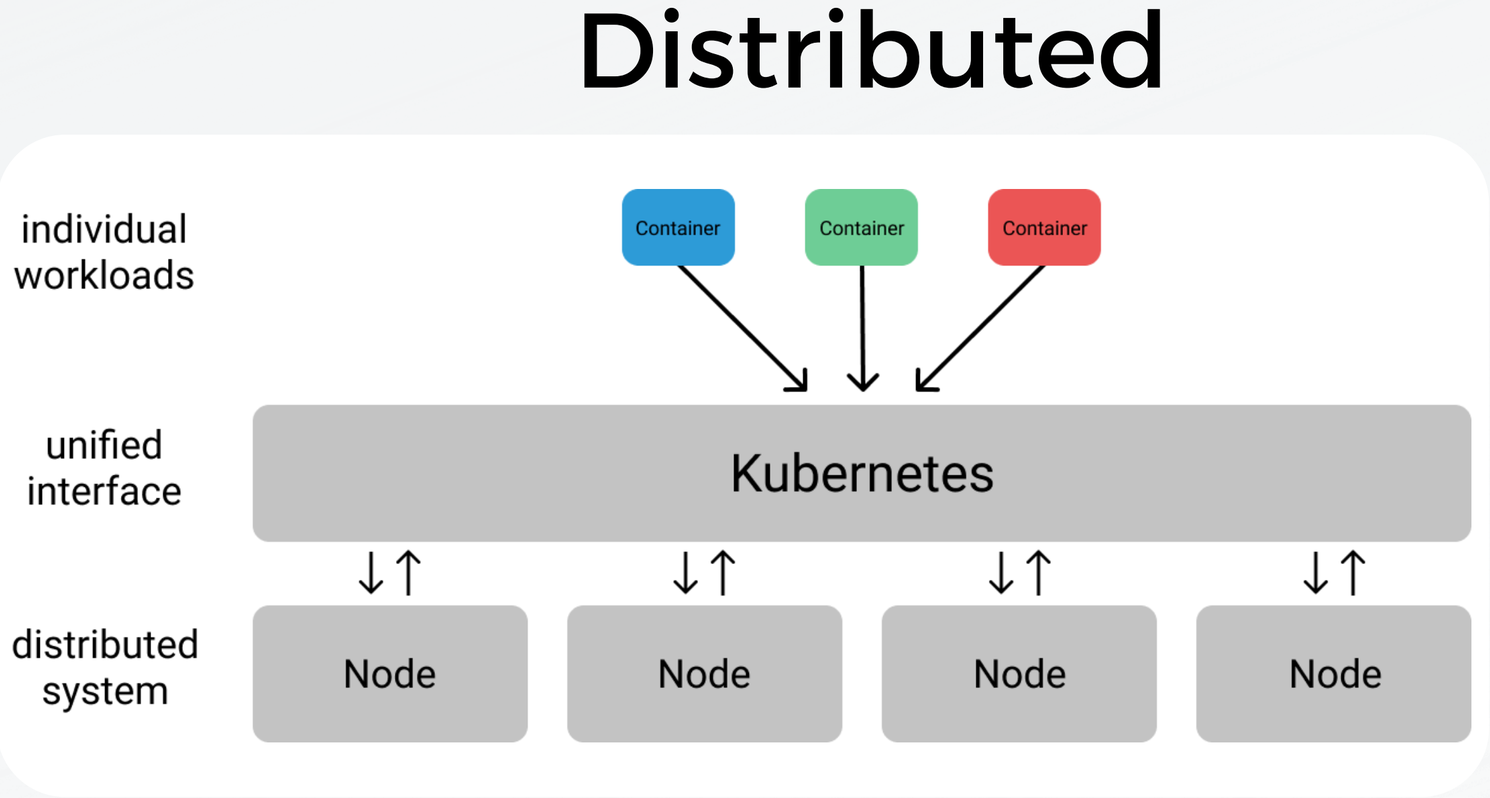
WHAT IS KUBERNETES?

Kubernetes is a container orchestration platform

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

KEY CONCEPTS OF KUBERNETES

Controller:
The system continuously monitors all containers, and if it detects a container that is not responding, it launches a new replica to replace it. It handles restarts as needed and manages scaling operations internally when necessary.



KEY CONCEPTS OF KUBERNETES

Separated ;

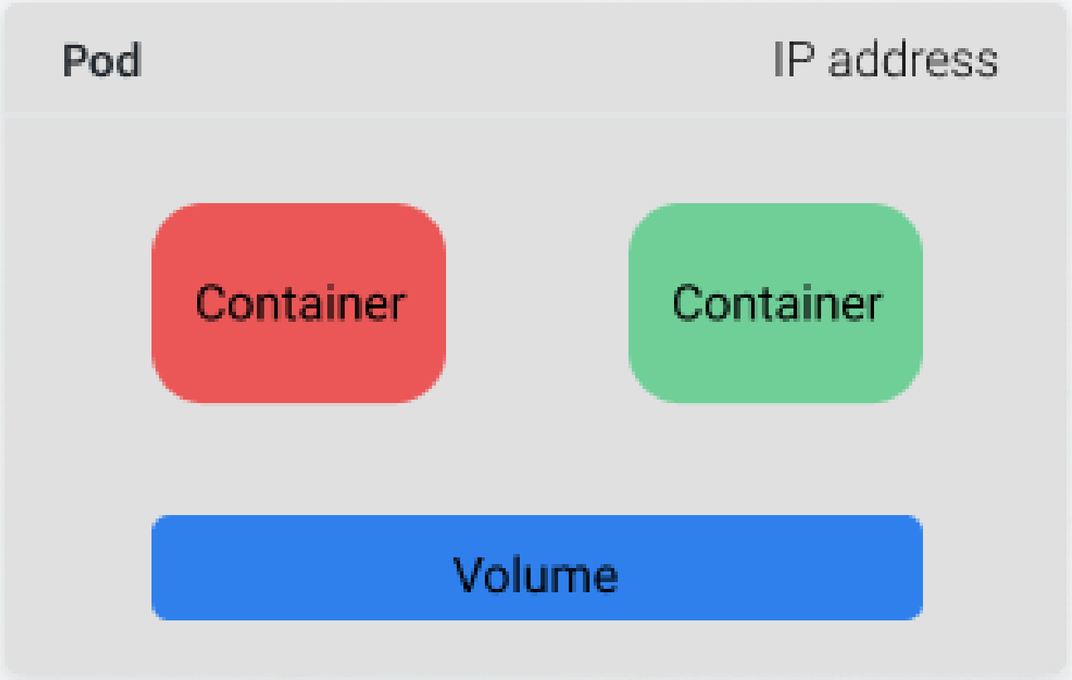
- A separated consideration in Kubernetes architecture development is "designing software applications as suites of independently deployable services."

Immutable Infrastructure;

- Once the environment is set up, instead of creating an entirely new container every time an issue occurs, it provides an infrastructure that allows us to replace the faulty container.

STEP BY STEP KUBERNETES: POD

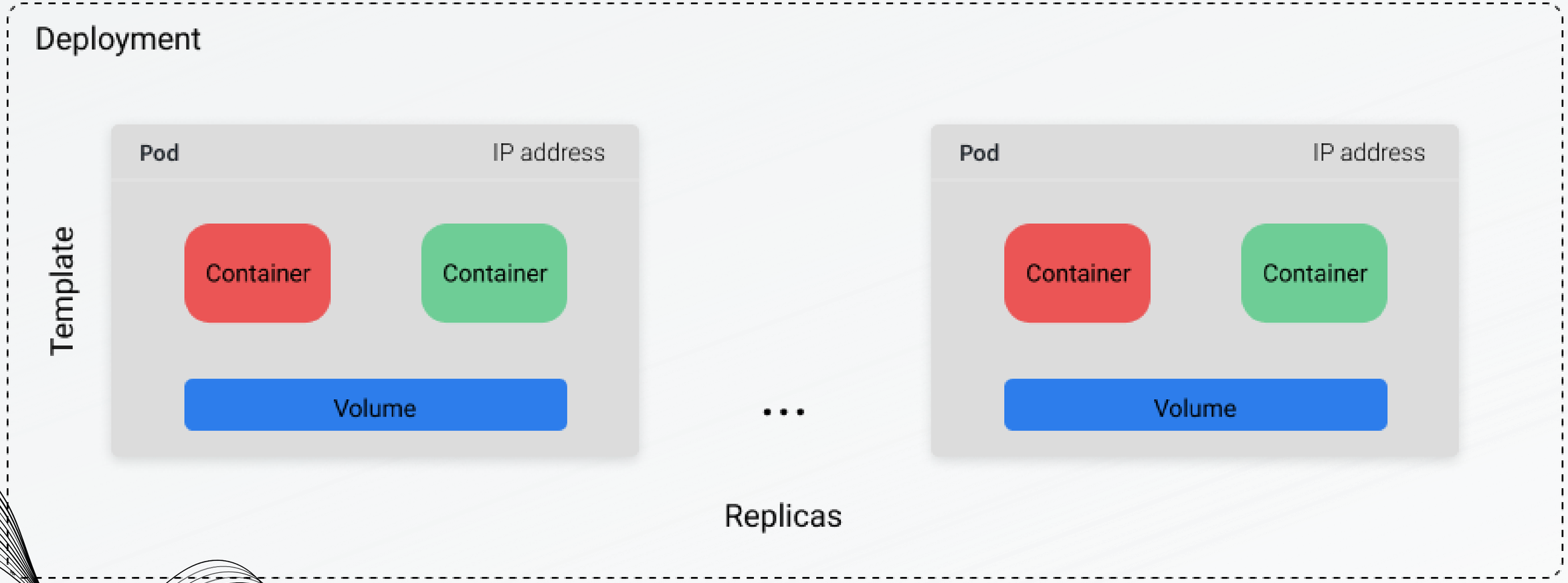
Kubernetes objects can be defined using **YAML** or **JSON** files; these files that define the objects are commonly referred to as **manifests**



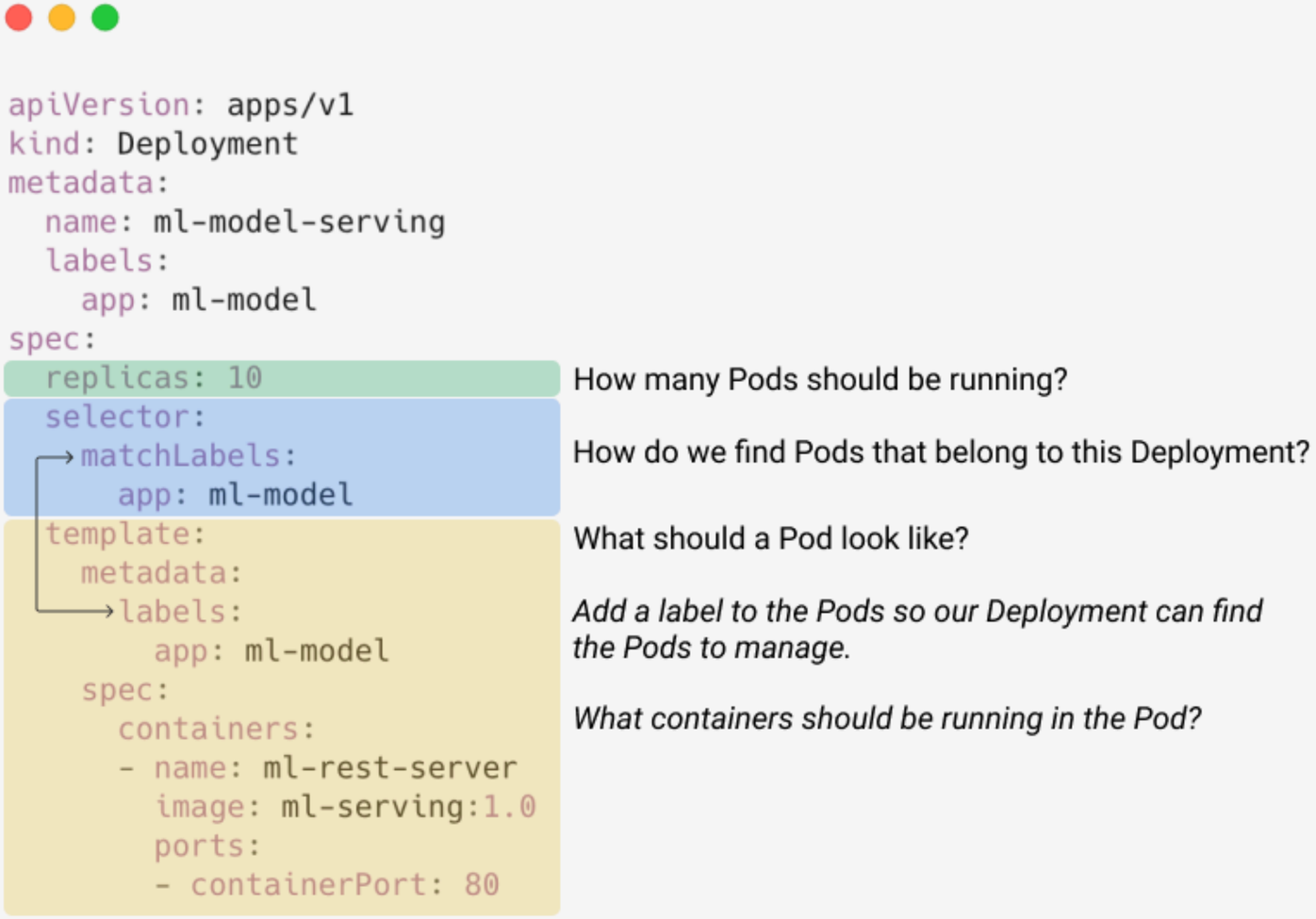
YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: DEPLOYMENT

The increase in replicas can be achieved independently or through a controller (horizontal pod autoscaler).



STEP BY STEP KUBERNETES: DEPLOYMENT



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: ml-model-serving
  labels:
    app: ml-model
spec:
  replicas: 10
  selector:
    matchLabels:
      app: ml-model
  template:
    metadata:
      labels:
        app: ml-model
    spec:
      containers:
      - name: ml-rest-server
        image: ml-serving:1.0
        ports:
        - containerPort: 80
```

How many Pods should be running?

How do we find Pods that belong to this Deployment?

What should a Pod look like?

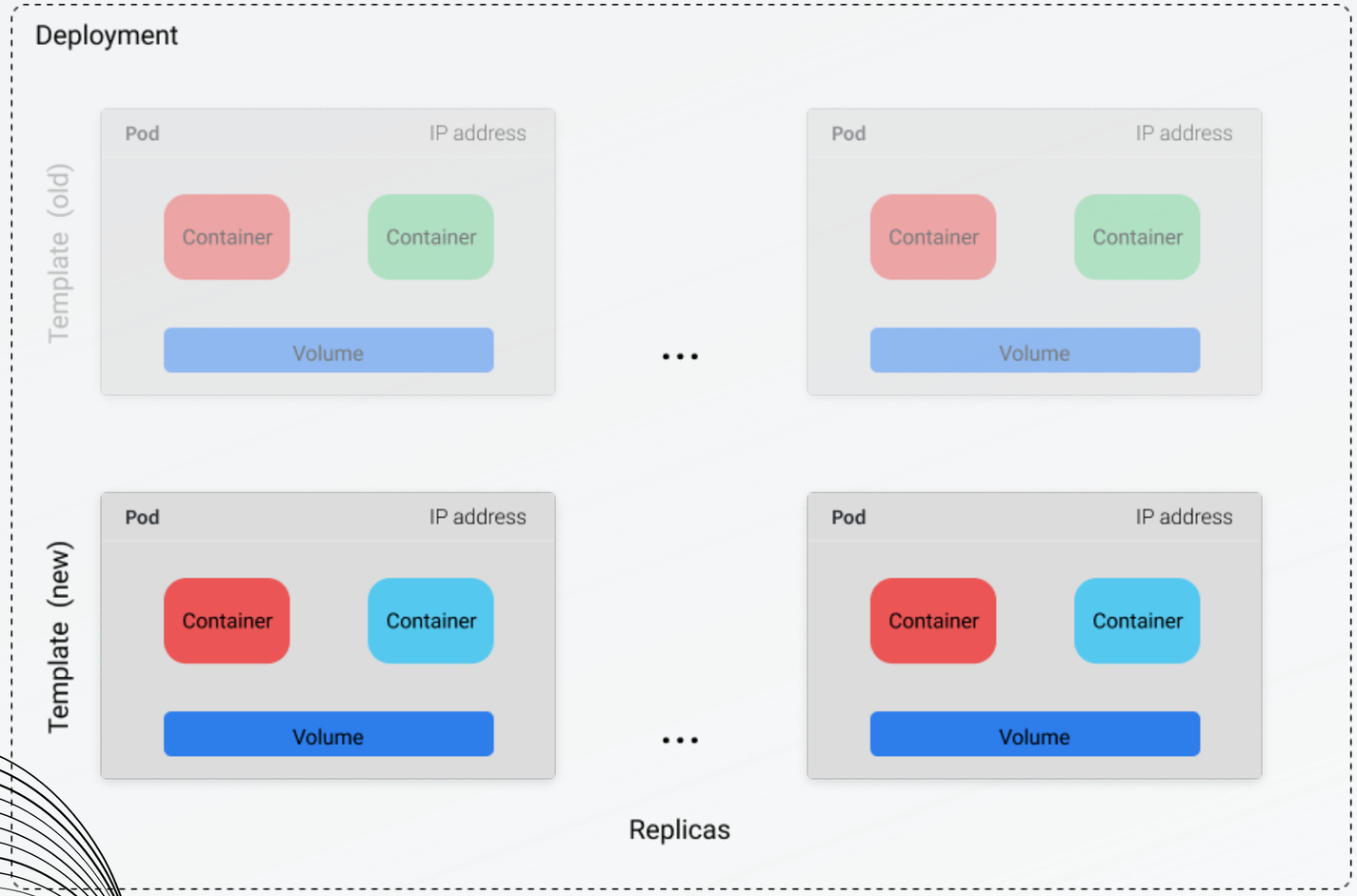
Add a label to the Pods so our Deployment can find the Pods to manage.

What containers should be running in the Pod?

The YAML file on the left is an annotated example of how to define a **Deployment** object. In this example, we want to run **10 replicas** of a container that serves an ML model through a REST application.

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: DEPLOYMENT



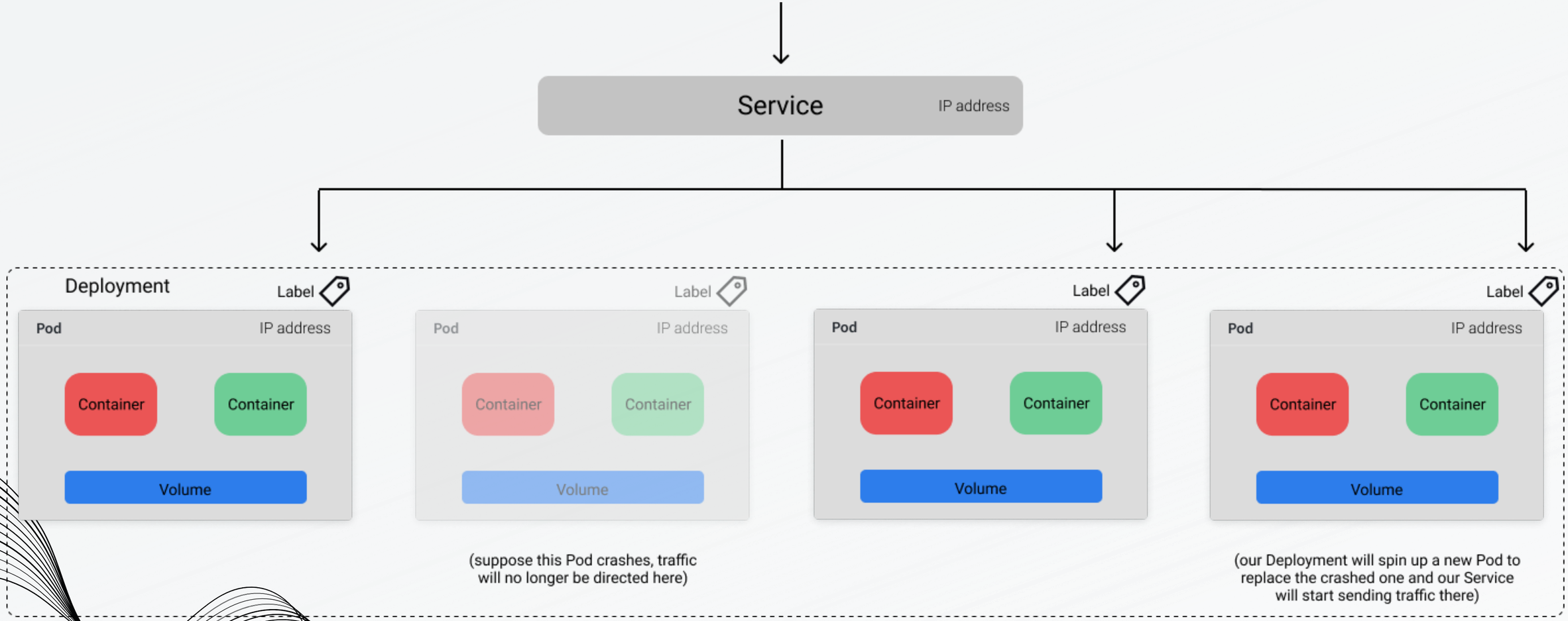
Deployments also allow us to determine how the structure will be updated and changed during the update processes.

```yaml
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 2        # how many pods we can add at a time
      maxUnavailable: 0  # maxUnavailable define how many pods can be unavai
                         # during the rolling update
```

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: SERVICE

Let’s assume that we have different ML-Server containers within each Pod, and at the same time, let’s consider that they have a structure prone to failures.



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: SERVICE

In the YAML file below, you can see how to set up an example service.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: ml-model-svc
  labels:
    app: ml-model
spec:
  type: ClusterIP
  selector:
    app: ml-model
  ports:
  - protocol: TCP
    port: 80
```

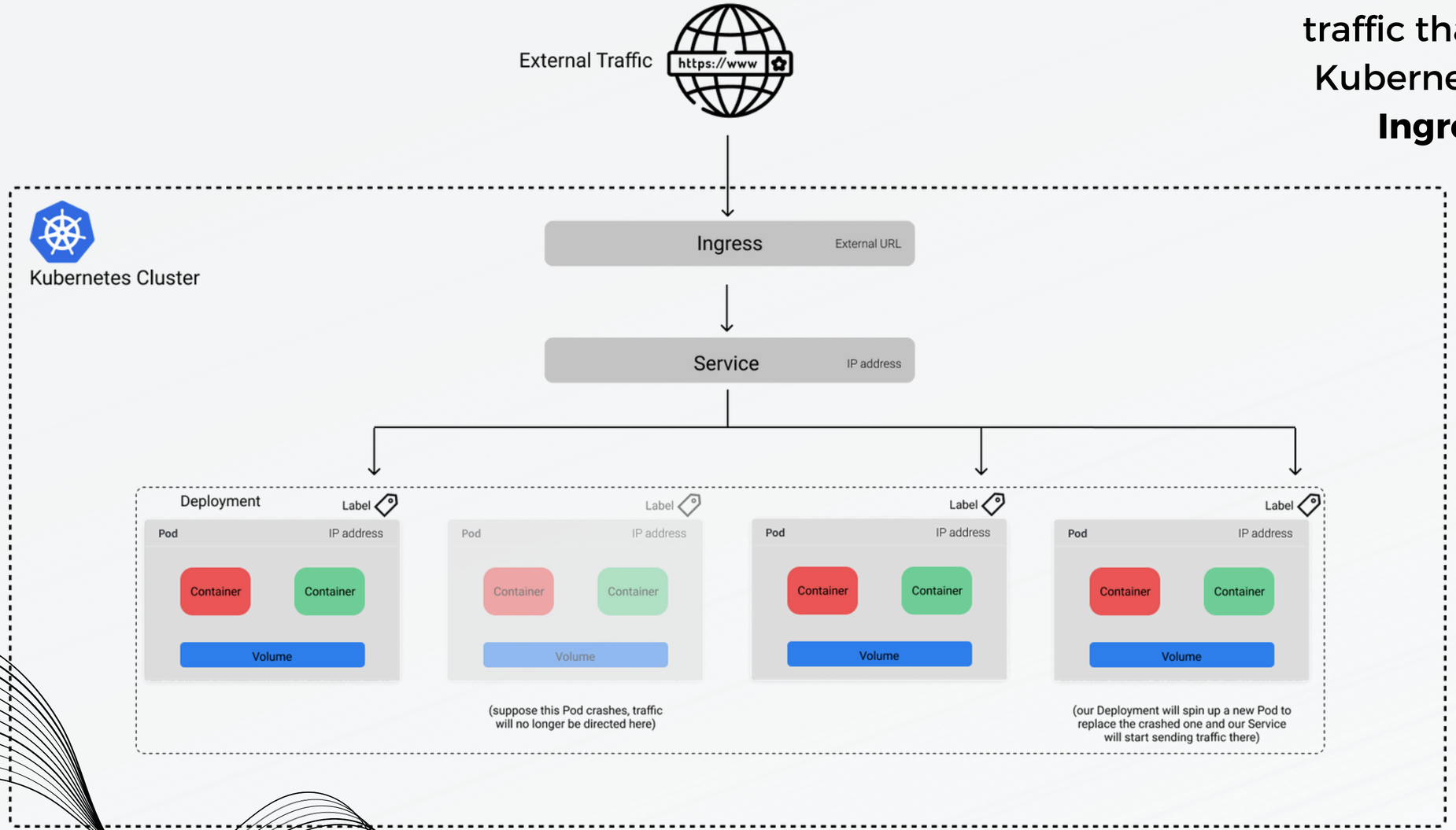
How do we want to expose our endpoint?

How do we find Pods to direct traffic to?

How will clients talk to our Service?

STEP BY STEP KUBERNETES: INGRESS

While the **Service** structure provides traffic that can only be used within Kubernetes, we need to define an **Ingress** object to expose it externally.



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: INGRESS

In the YAML file below, you can see how to set up an example **Ingress**.

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: ml-product-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
      paths:
      - path: /app
        backend:
          serviceName: user-interface-svc
          servicePort: 80
```

Configure options for the Ingress controller.

How should external traffic access the service?

What Service should we direct traffic to?

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: JOB

So far, we've learned how to provide services using various Kubernetes structures. Now, let’s consider that we need to train a Machine Learning model and incorporate it into our service. This is where a **Job** comes into play.

YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

STEP BY STEP KUBERNETES: JOB

In the YAML file below, you can see how to set up an example **Job**

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: model-train-job
spec:
  template:
    spec:
      containers:
      - name: ml-model-train
        image: ml-development:1.2.1
        command: ["python",  "train.py"]
      restartPolicy: Never
  backoffLimit: 5
```

How should we create the Pod for this Job?

How many times should we try to complete this Job before quitting?

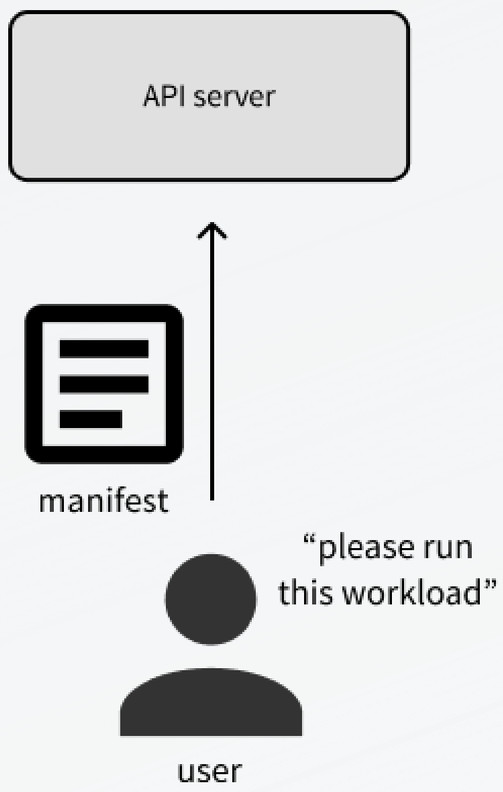
YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

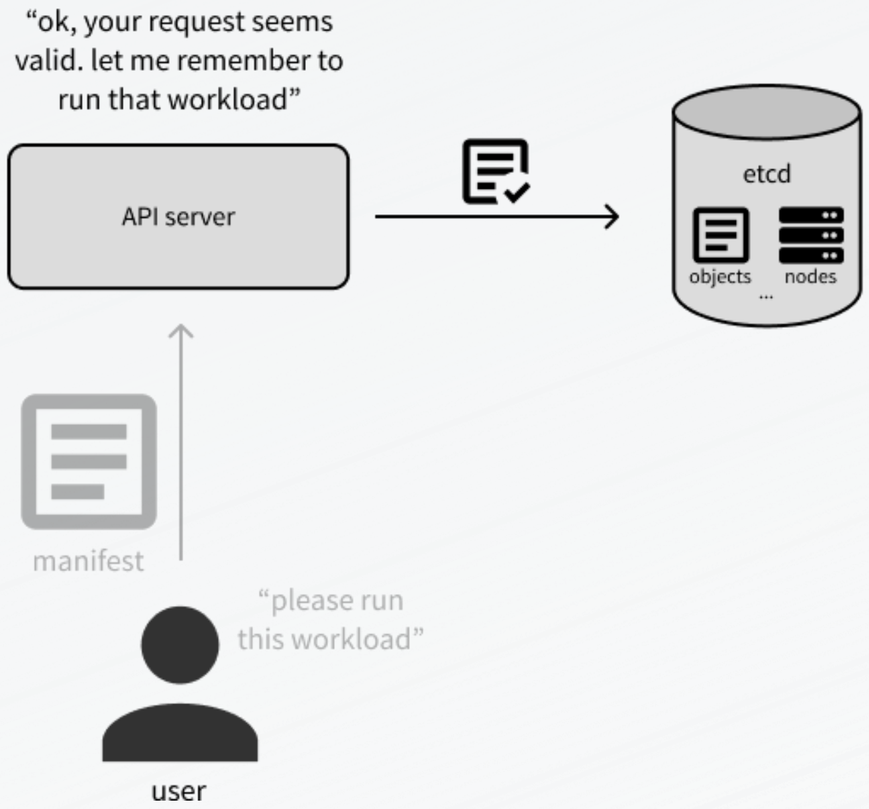
OTHER CONCEPTS

Aside from this, we have many other concepts;

- **Volume**: for managing directories mounted onto Pods
- **Secret**: for storing sensitive credentials
- **Namespace**: for separating resources on your cluster
- **ConfigMap**: for specifying application configuration values to be mounted as a file
- **HorizontalPodAutoscaler**: for scaling Deployments based on the current resource utlization of existing Pods
- **StatefulSet**: similar to a Deployment, but for when you need to run a stateful application

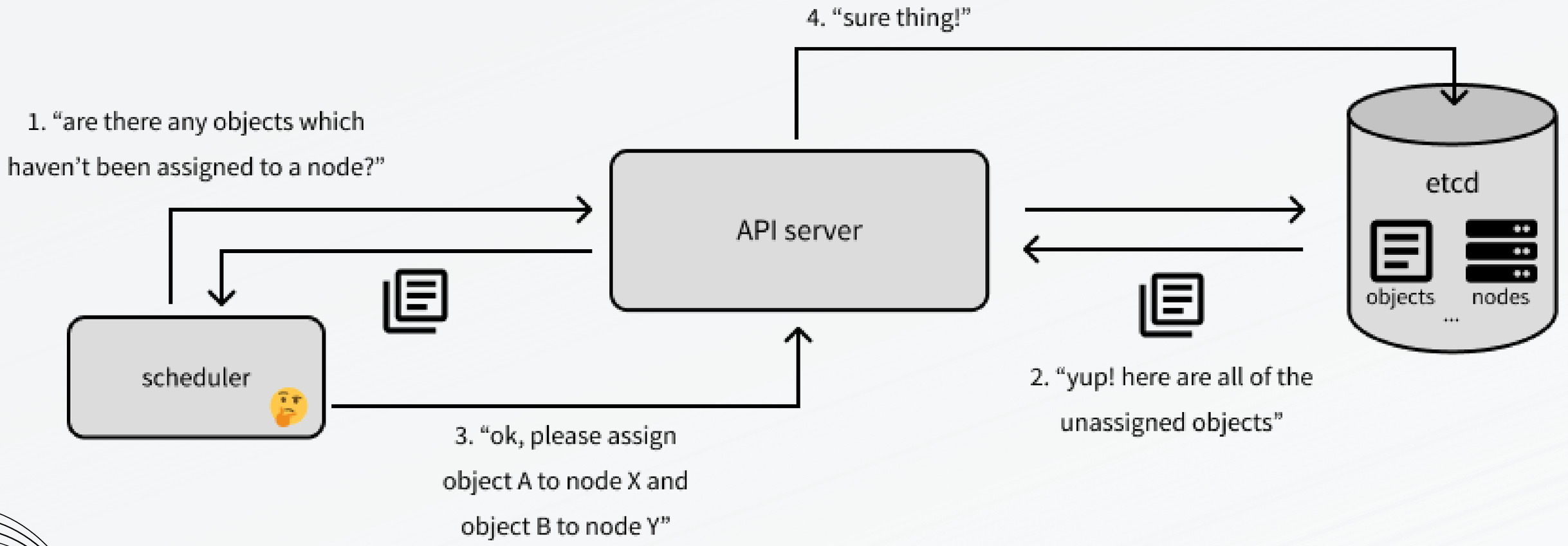
MANAGEMENT OF KUBERNETES





YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

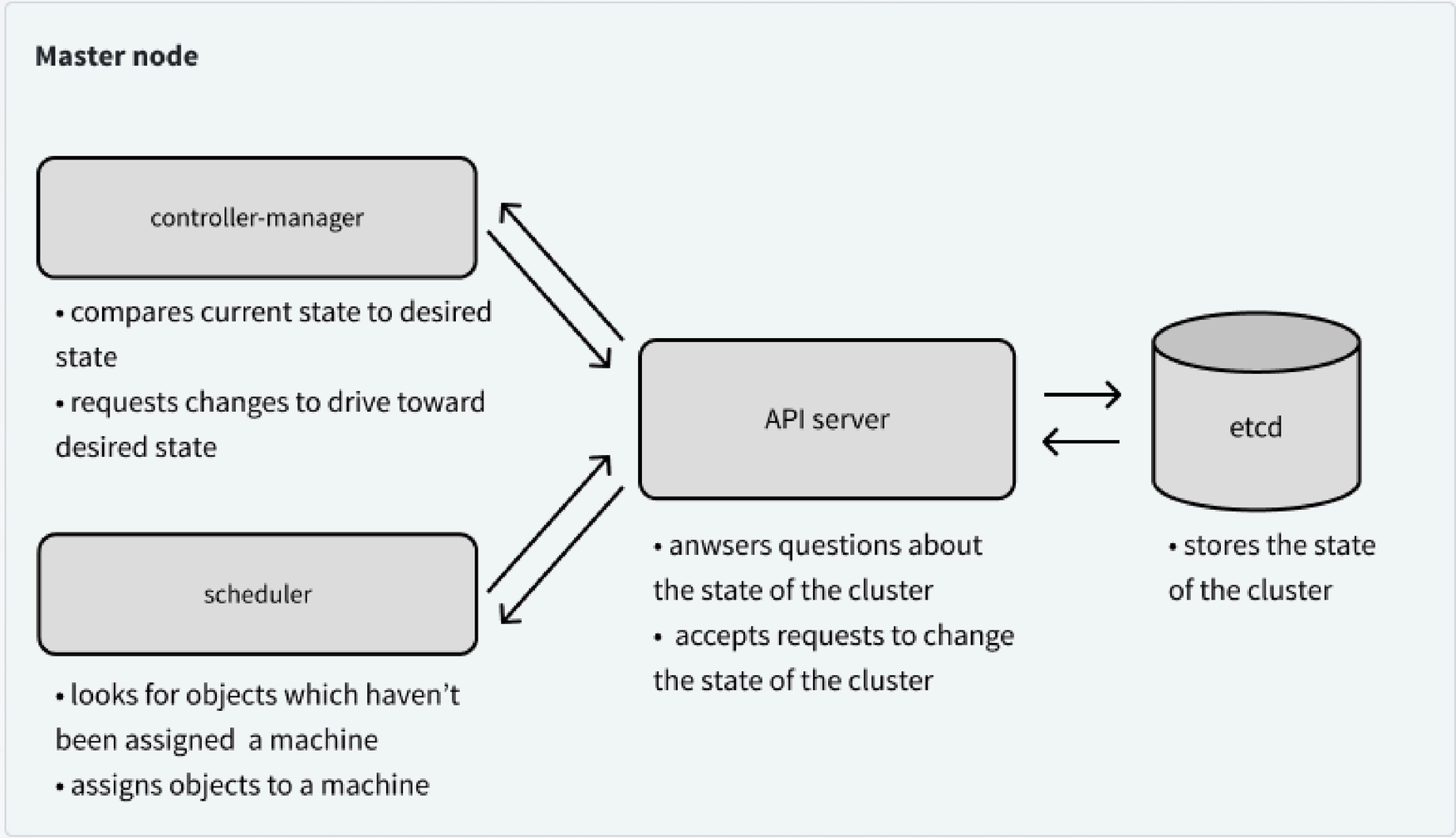
MANAGEMENT OF KUBERNETES



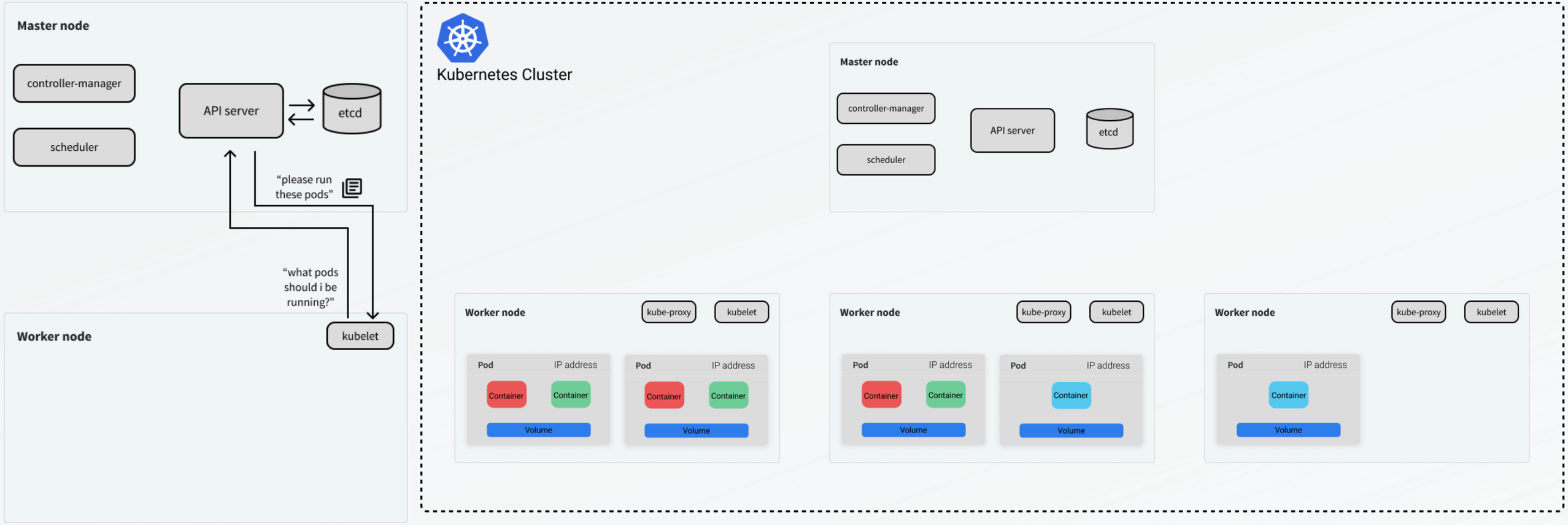
YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

MANAGEMENT OF KUBERNETES

Yönetici



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024



YUNUS EMRE GÜNDOĞMUŞ - 20.12.2024

WHEN SHOULD WE NOT USE KUBERNETES?

- If your system runs on a single machine.
- If you don’t need to scale and can tolerate minor crashes without issue.
- If you have a monolithic architecture and don’t plan to transition to a microservices architecture.
- If, after hearing all this, you’re thinking “this is too complicated” rather than “this is really useful.”

DEMO