# Kubernetes Objects

# Table of Contents

1

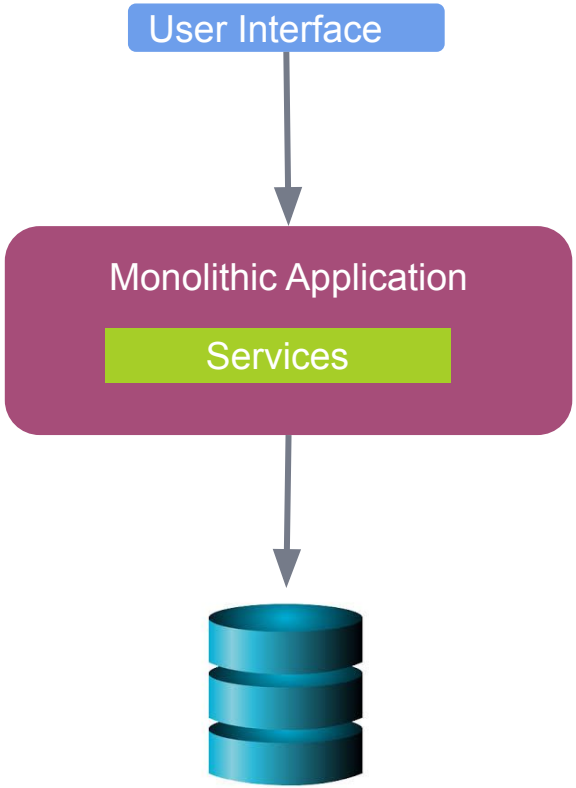
# Monolith vs Microservices

# Monolith vs Microservices

The word 'monolith' means 'one massive stone'. So we can describe monolithic as a large unified block.

# Monolith vs Microservices

In software development, **monolithic architecture** is a traditional way to build an application as a single and indivisible unit.



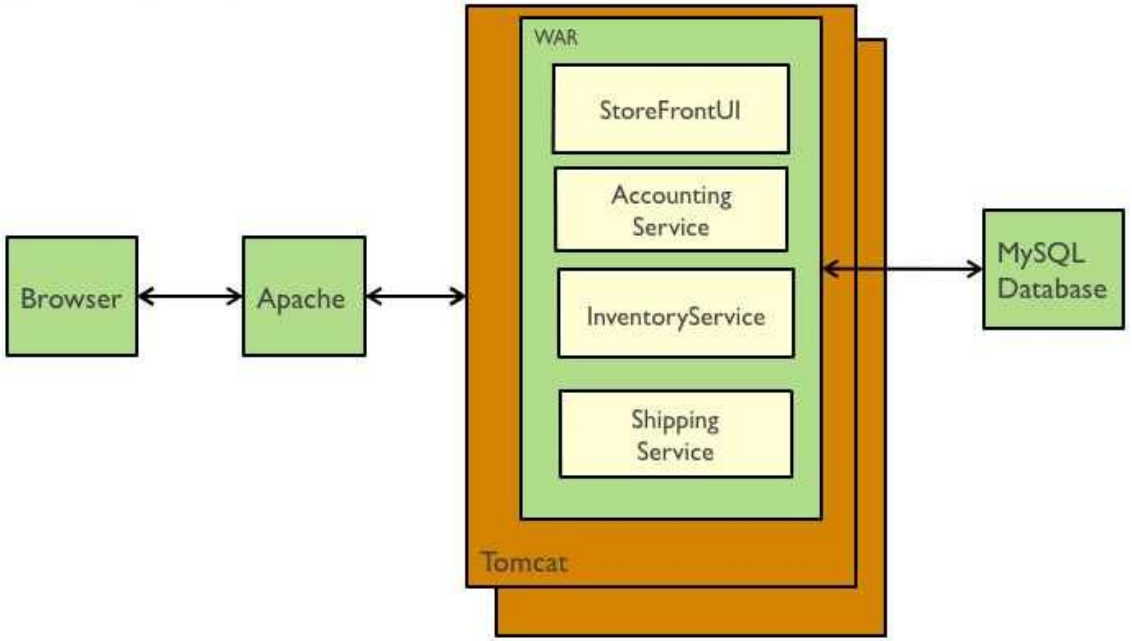
# Monolith vs Microservices

Let’s imagine that we are building an e-commerce application that takes orders from customers, verifies inventory and available credit, and ships them.

The application consists of several components including the StoreFrontUI, which implements the user interface, along with some backend services for checking credit, maintaining inventory and shipping orders.

# Monolith vs Microservices

The application is deployed as a single monolithic application. For example, a Java web application consists of a single WAR file that runs on a web container such as Tomcat.



# Monolith vs Microservices

**Pros of monolithic architecture:**

- **Easier to develop.** As long as the monolithic approach is a standard way of building applications, any engineering team has the right knowledge and capabilities to develop a monolithic application.
- **Easier to deploy.** You need to deploy your application only once instead of performing multiple deployments of different files.
- **Easier to test and debug.** Since a monolithic app is a single indivisible unit, you can run end-to-end testing much faster.

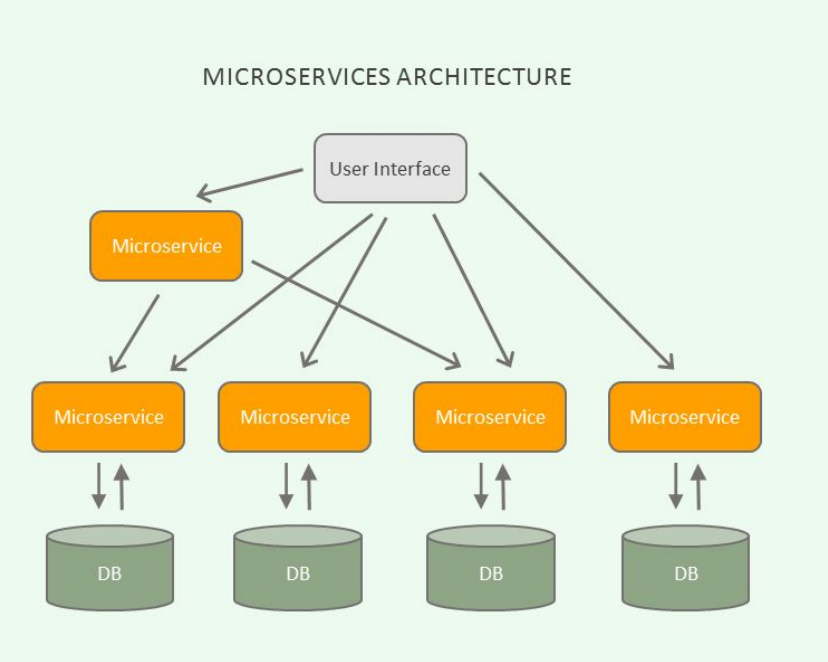
# Monolith vs Microservices

**Cons of monolithic architecture:**

- **Understanding.** When a monolithic application scales up, it becomes too complicated to understand.
- **Making changes.** Any code change affects the whole system so it has to be thoroughly coordinated.
- **Scalability.** You cannot scale components independently, only the whole application.
- **New technology barriers.** It is extremely problematic to apply a new technology in a monolithic application because then the entire application has to be rewritten.

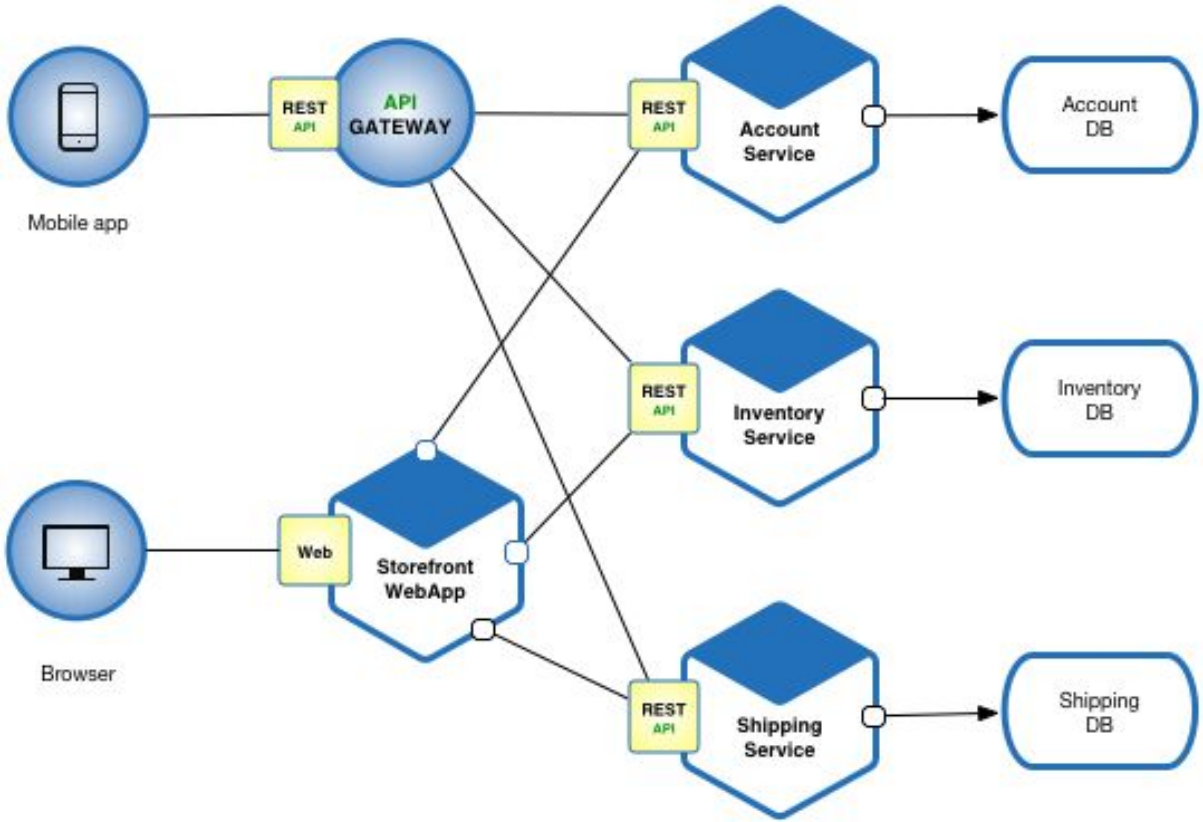
# Monolith vs Microservices

While a monolithic application is a single unified unit, a **microservices architecture** breaks it down into a collection of smaller independent units.



# Monolith vs Microservices

# Monolith vs Microservices



# Monolith vs Microservices

**Pros of microservices:**

- **Independent components.**
  - All the services can be deployed and updated independently, which gives more flexibility.
  - A bug in one microservice has an impact only on a particular service and does not influence the entire application.
  - It is much easier to add new features to a microservice application than a monolithic one.

# Monolith vs Microservices

**Pros of microservices:**

- **Easier understanding.** Split up into smaller and simpler components, a microservice application is easier to understand and manage.
- **Better scalability.** Each element can be scaled independently. So the entire process is more cost- and time-effective than with monoliths when the whole application has to be scaled even if there is no need in it.

# Monolith vs Microservices

**Pros of microservices:**

- **Flexibility in choosing the technology.** The engineering teams are not limited by the technology chosen from the start. They are free to apply various technologies and frameworks for each microservice.
- **The higher level of agility.** Any fault in a microservices application affects only a particular service and not the whole solution. So all the changes and experiments are implemented with lower risks and fewer errors.

# Monolith vs Microservices

**Cons of microservices:**

- **Extra complexity.** Since a microservices architecture is a distributed system, you have to choose and set up the connections between all the modules and databases.
- **System distribution.** A microservices architecture is a complex system of multiple modules and databases so all the connections have to be handled carefully.
- **Testing.** A multitude of independently deployable components makes testing a microservices-based solution much harder.

2

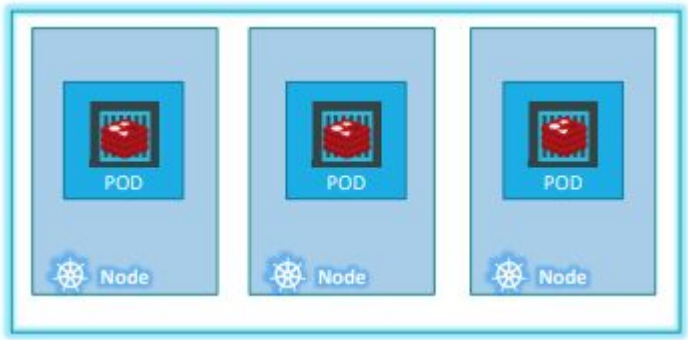
# Kubernetes Objects

# Kubernetes Objects

Kubernetes objects are persistent entities in the Kubernetes system. Kubernetes uses these entities to represent the state of your cluster. Specifically, they can describe:

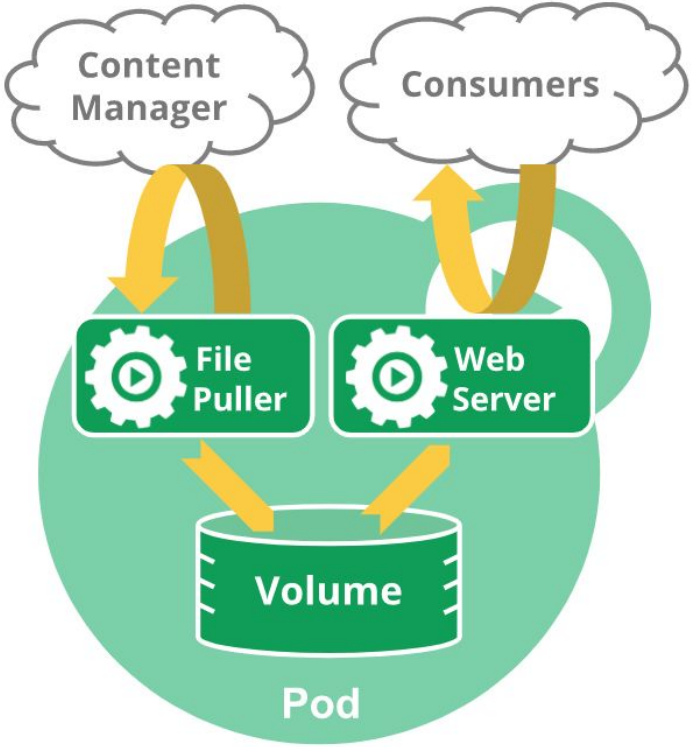
- What containerized applications are running (and on which nodes)
- The resources available to those applications
- The policies around how those applications behave, such as restart policies, upgrades, and fault-tolerance

# 3 PODs

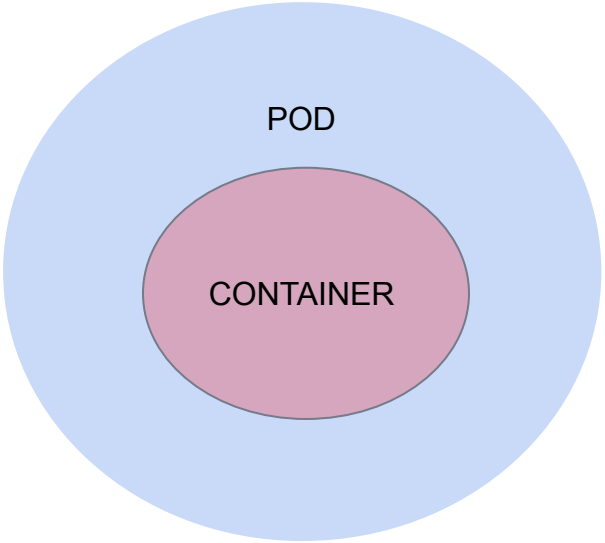


# PODs

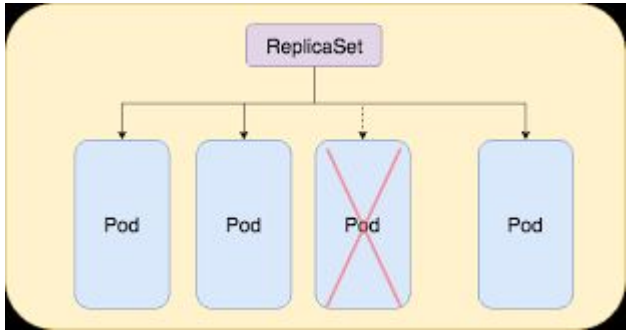
- The containers are encapsulated into a Kubernetes object known as PODs.
- A POD is a single instance of an application.
- A POD is the smallest object, that you can create in kubernetes.



# PODs

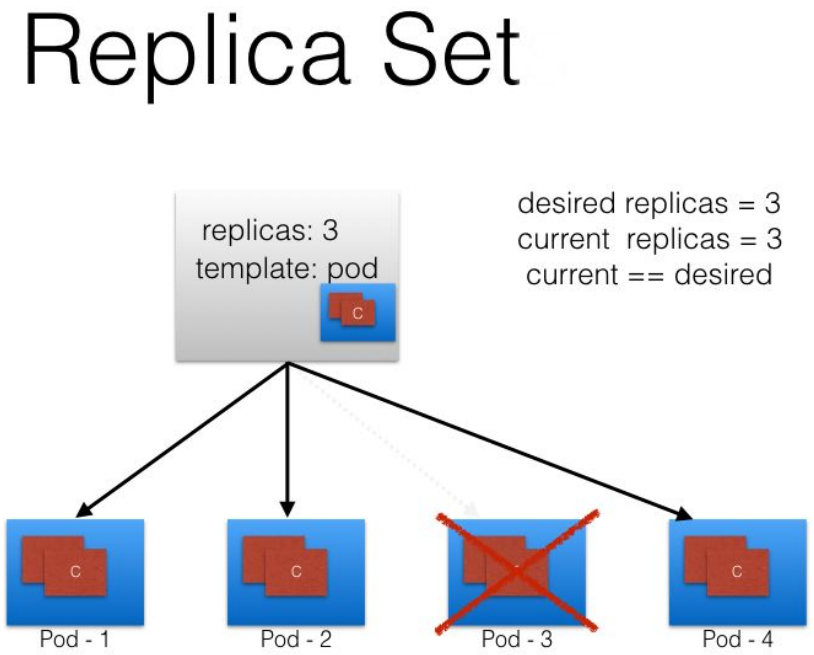


# 4 ReplicaSets

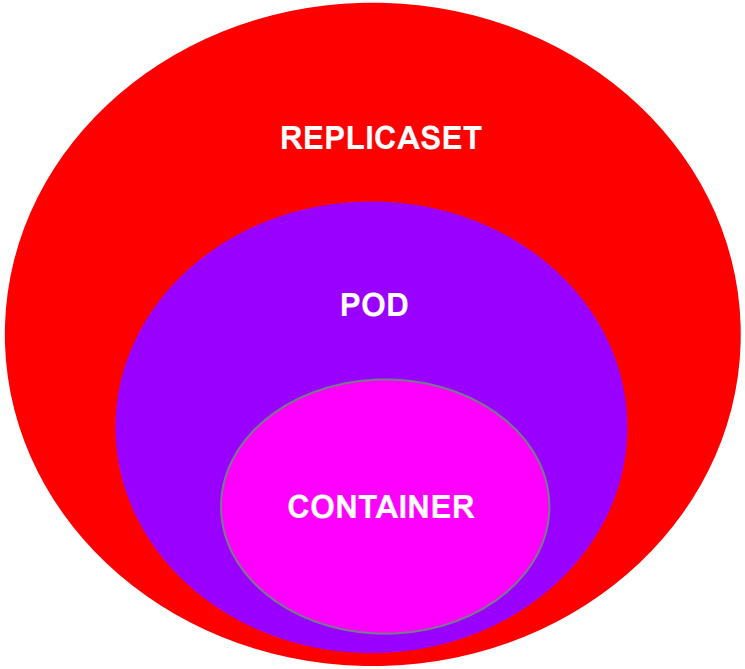


# Replication Sets

A **ReplicaSet's** purpose is to maintain a stable set of replica Pods running at any given time. As such, it is often used to guarantee the availability of a specified number of identical Pods.

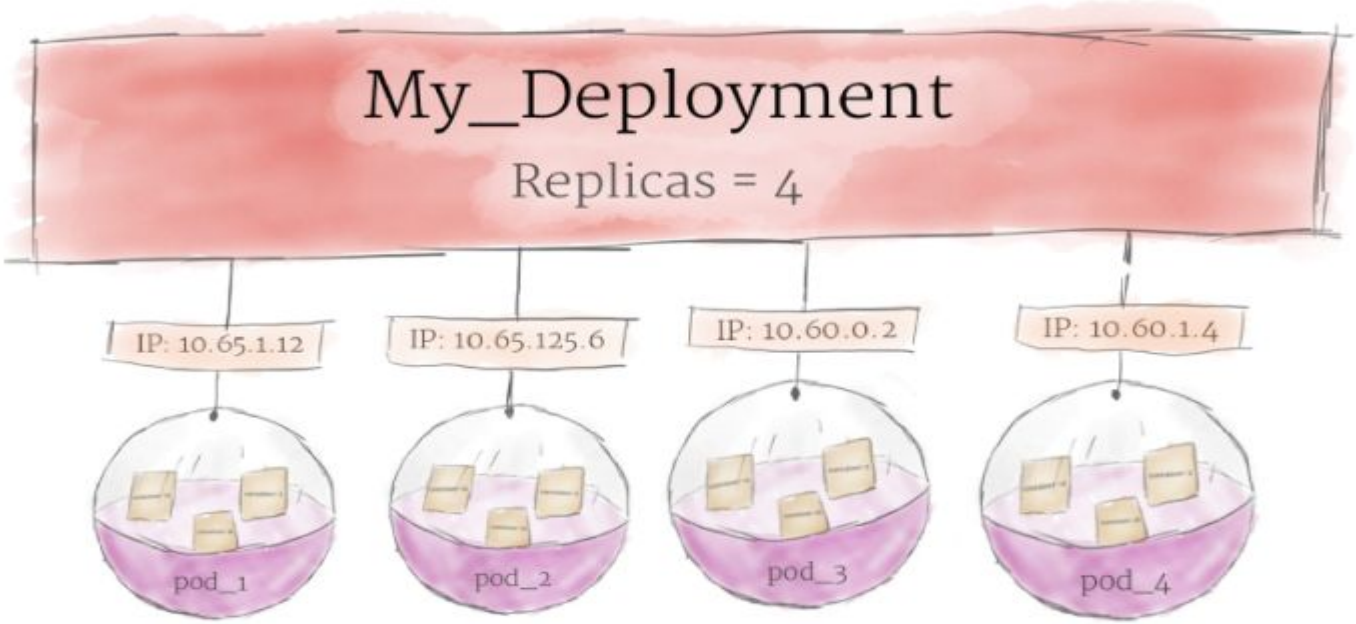


# Replication Sets



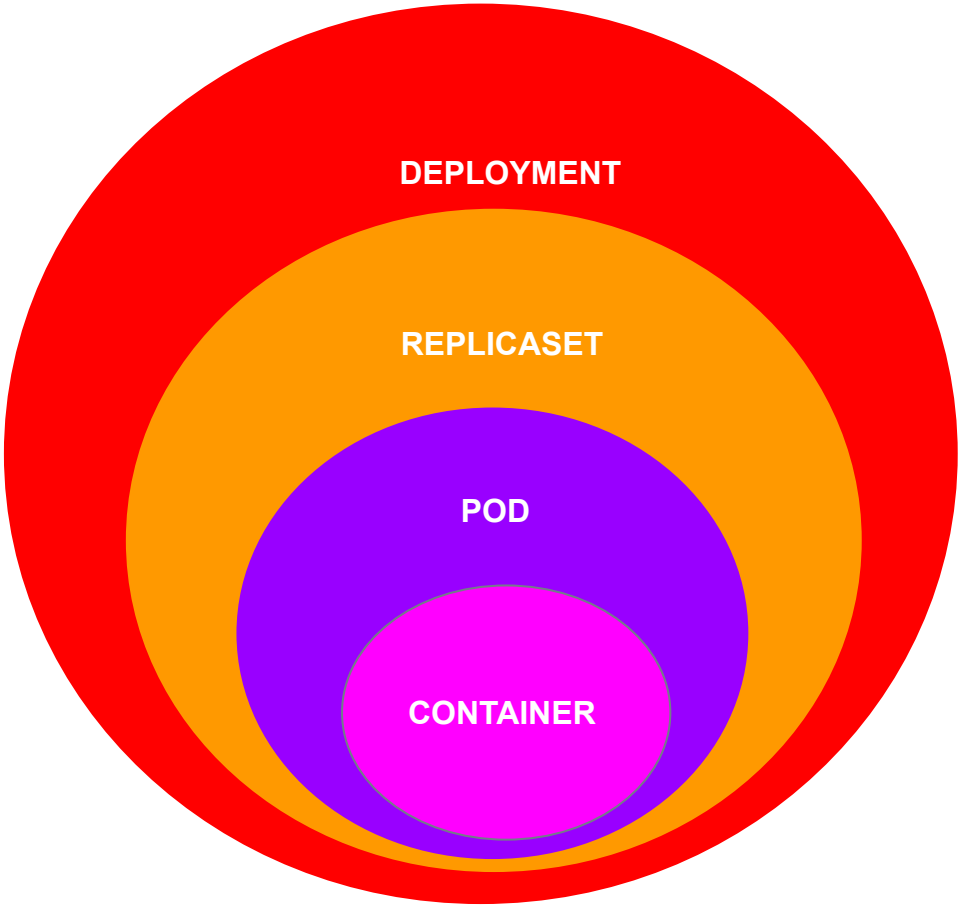
# 5 Deployment

# Deployment




**Deployment** is a method of converting images to containers and then allocating those images to pods in the Kubernetes cluster. A Deployment provides declarative updates for Pods and ReplicaSets.

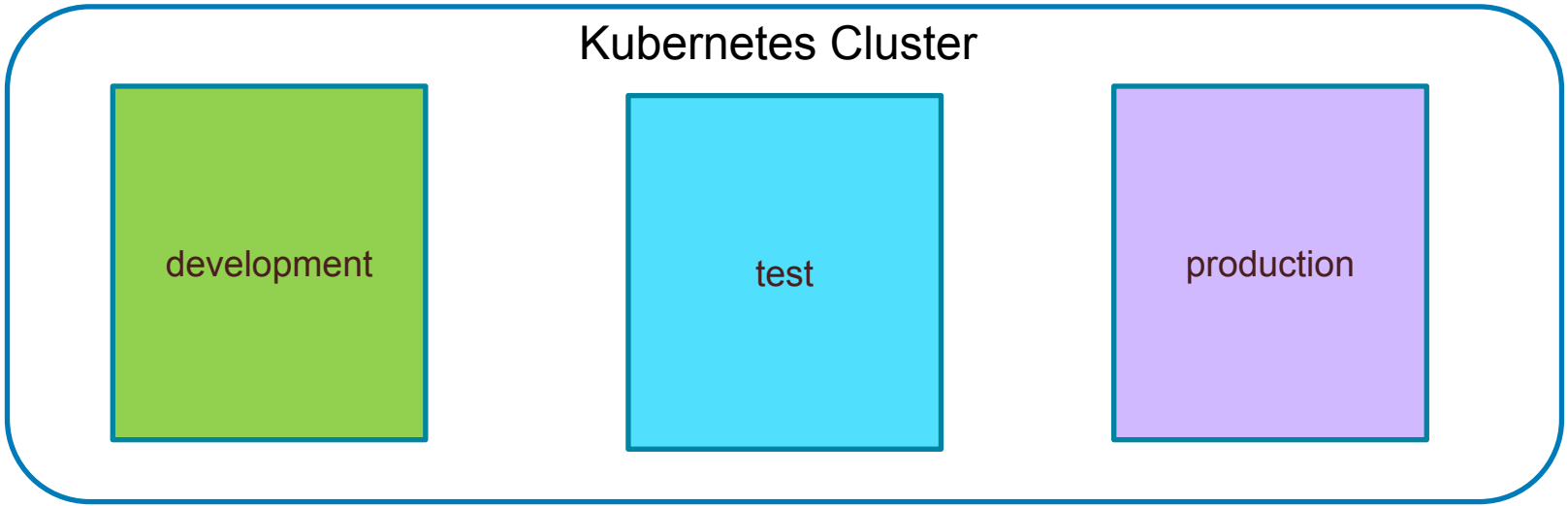
# Deployment



# 6 Namespaces

# Namespaces

- Kubernetes supports multiple virtual clusters backed by the same physical cluster. These virtual clusters are called **namespaces**.
- The names of the resources/objects created inside a Namespace are unique, but not across Namespaces in the cluster.



7

# Declarative vs Imperative

# Declarative vs Imperative

**Declarative** and **imperative** approach is a DevOps paradigm, programmatic approach or Kubernetes management.

- While using an **imperative paradigm**, the user is responsible for defining exact steps which are necessary to achieve the end goal, such as instructions for software installation, configuration, database creation, etc.
- In **declarative paradigm**, instead of defining exact steps to be executed, the ultimate state is defined. The user declares how many machines will be deployed, will workloads be virtualised or containerised, which applications will be deployed, how will they be configured, etc.

# Declarative vs Imperative

imperative focuses on **how** and declarative focuses on **what**.

**Imperative approach:**

1. Build the foundation
2. Put in the framework
3. Add the walls
4. Add the doors and windows

**Declarative approach:**

I want a little tiny house.

# 8 Object Model

# Object Model

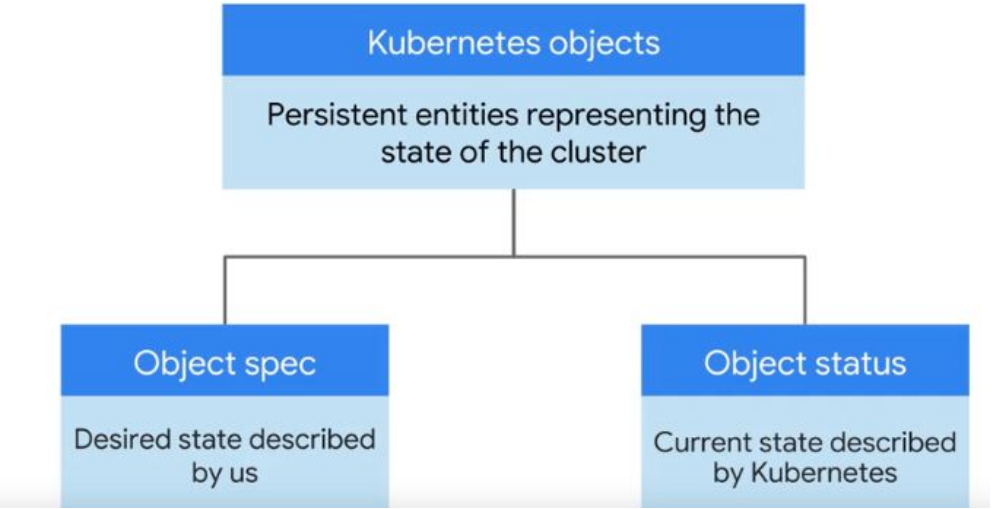
```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

All objects must have **apiVersion, kind, metadata** and **spec** fields.

- **apiVersion:** Which version of the Kubernetes API you're using to create this object
- **kind:** What kind of object you want to create
- **metadata:** Data that helps uniquely identify the object, including a **name** string, **UID**, and optional **namespace**
- **spec:** What state you desire for the object

# Object Model

- Once the Deployment object is created, the Kubernetes system attaches the **status** field to the object.
- **status** is managed by Kubernetes and describes the **actual state** of the object and its history.



# Object Model

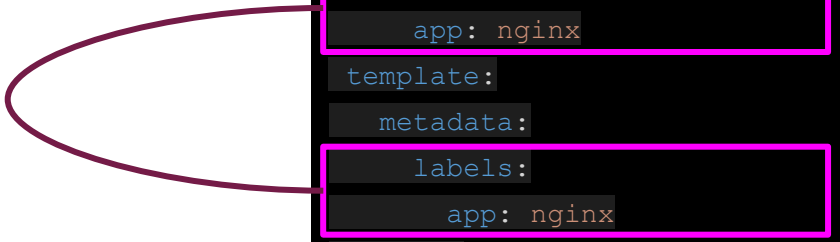
## Pod to ReplicaSet

```yaml
apiVersion: v1
kind: Pod
metadata:
 name: nginx-pod
 labels:
   app: nginx
spec:
 containers:
 - name: mynginx
   image: nginx:1.19
   ports:
   - containerPort: 80
```

```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: nginx-rs
 labels:
   environment: dev
spec:
 replicas: 3
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: mynginx
       image: nginx:1.19
       ports:
       - containerPort: 80
```

# Object Model

## Pod Selector



```yaml
apiVersion: apps/v1
kind: ReplicaSet
metadata:
 name: nginx-rs
 labels:
   environment: dev
spec:
 replicas: 3
 selector:
   matchLabels:
     app: nginx
 template:
   metadata:
     labels:
       app: nginx
   spec:
     containers:
     - name: mynginx
       image: nginx:1.19
       ports:
       - containerPort: 80
```

# THANKS!

## Any questions?

You can find me at:

- Joe@clarusway.com