



Why Do we need Container Orchestration?

Docker
Swarm!

1

Containers are Good...

- Both Linux Containers and Docker Containers
 - Isolate the application from the host

FASTER, RELIABLE, EFFICIENT, LIGHTWEIGHT, AND SCALABLE



Containers Problems!

- Both Linux Containers and Docker Containers
 - Isolate the application from the host



FASTER, RELIABLE, EFFICIENT, LIGHTWEIGHT, AND SCALABLE



3

Containers Problems!

- Both Linux Containers and Docker Containers
 - Isolate the application from the host

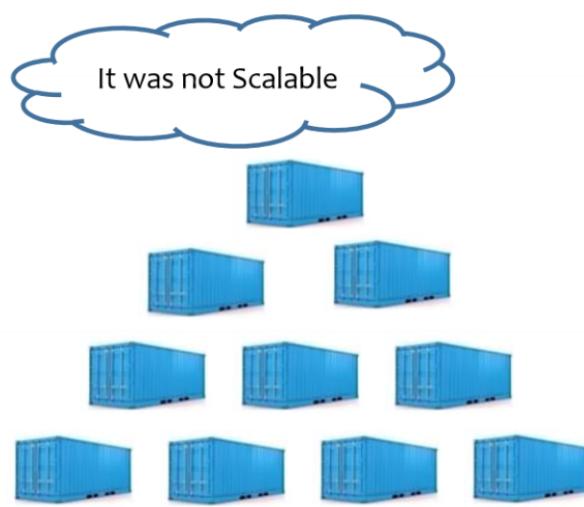


FASTER, RELIABLE, EFFICIENT, LIGHTWEIGHT, AND SCALABLE



4

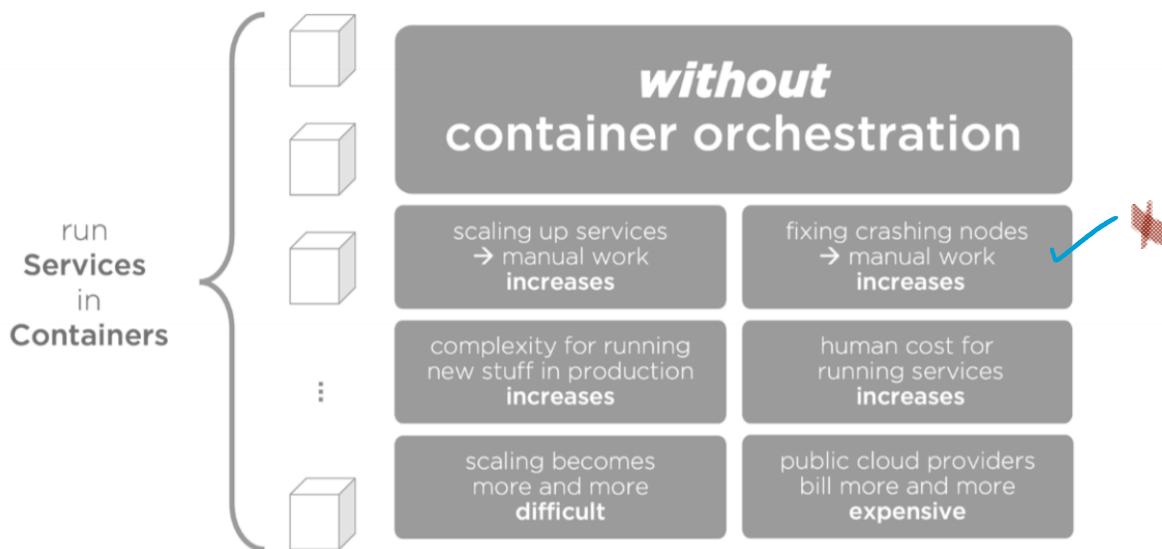
Problems with Scaling up Containers



- 1 Containers could communicate with each other
- 2 Containers had to be deployed appropriately
- 3 Containers had to be managed carefully
- 4 Auto scaling was not possible
- 5 Distributing traffic was still challenging

5

Containers Without Orchestration



6

Container Orchestration

7

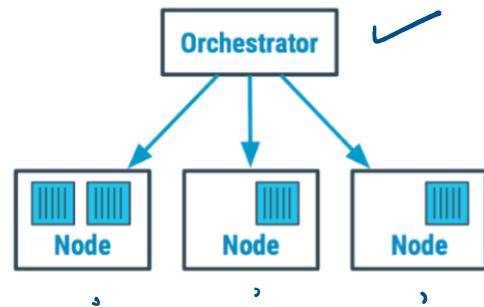
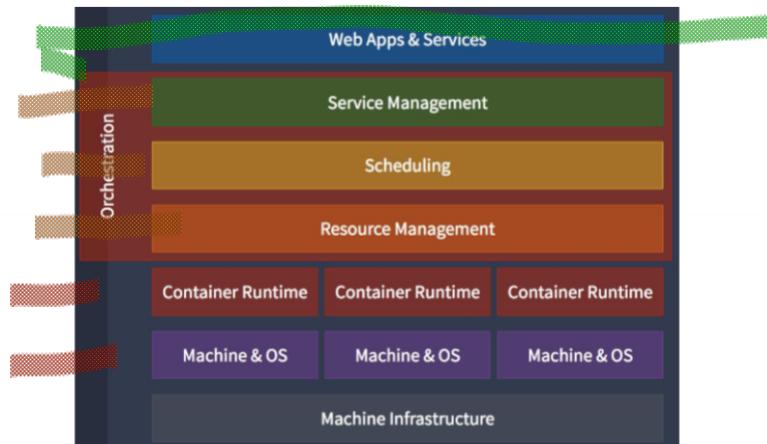
What is container orchestration?

- Container orchestration is the automation of much of the operational effort required to run containerized workloads and services.
- This includes a wide range of things software teams need to manage a container's lifecycle, including:
 - provisioning, ✓
 - deployment, ✓
 - scaling (up and down), ✓
 - networking, ✓
 - load balancing and more. ✓

8

Container Orchestration

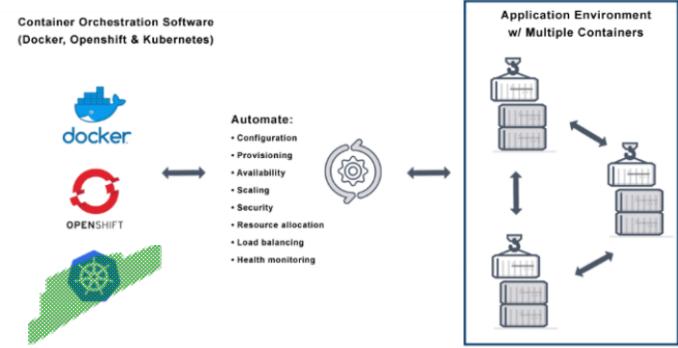
- Container orchestration automates and simplifies provisioning, and deployment and management of containerized applications.



9

Container Orchestration

- Container orchestration is the automatic process of managing or scheduling the work of individual containers for applications based on microservices within multiple clusters.
- The widely deployed container orchestration platforms are based on open-source versions like Kubernetes, Docker Swarm or the commercial version from Red Hat OpenShift.



10

Container Orchestration

- Fault-tolerance
- On-demand scalability
- Optimal resource usage
- Auto-discovery to automatically discover and communicate with each other
- Accessibility from the outside world
- Seamless updates/rollbacks without any downtime.

11

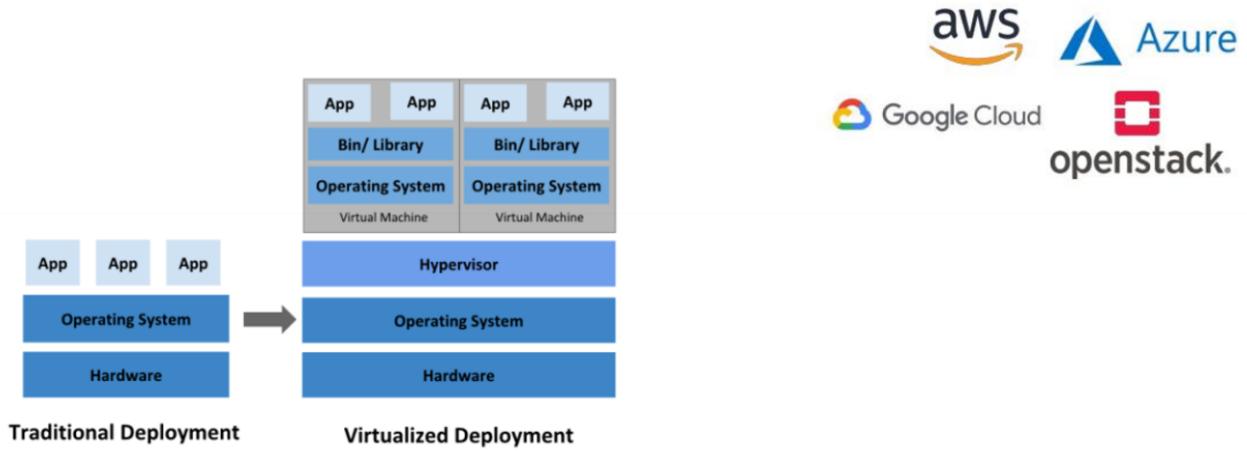
Why Do We Need Container Orchestration?

- Container orchestration is used to automate the following tasks at scale:
 - Configuring and scheduling of containers
 - Provisioning and deployments of containers
 - Availability of containers
 - The configuration of applications in terms of the containers that they run in
 - Scaling of containers to equally balance application workloads across infrastructure
 - Allocation of resources between containers
 - Load balancing, traffic routing and service discovery of containers
 - Health monitoring of containers
 - Securing the interactions between containers.

12

Cloud Orchestration

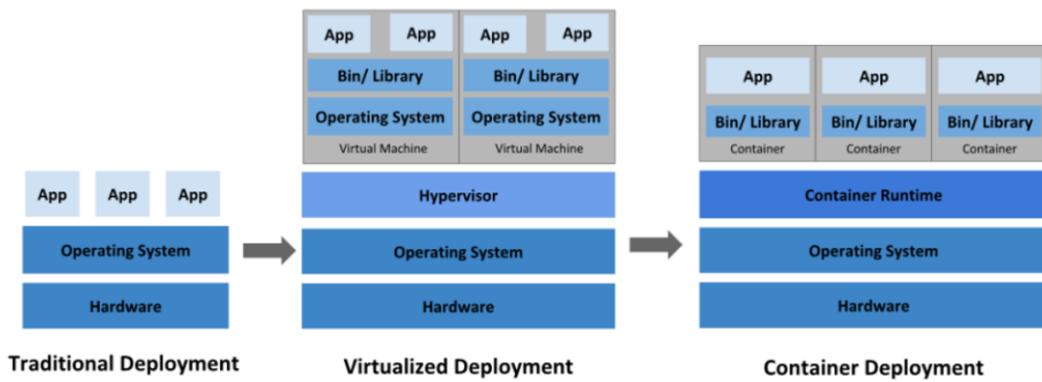
- Infrastructure-as-a-Service (IaaS)
 - Provisioning virtual machines from a cloud service provider (CSP)



13

Container Orchestration

- Application containers
 - Lightweight OS-virtualization
 - Application packaging for portable, reusable software



14

Container Orchestration

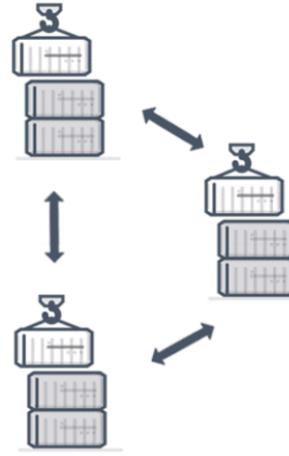
Container Orchestration Software
(Docker, Openshift & Kubernetes)



Automate:

- Configuration
- Provisioning
- Availability
- Scaling
- Security
- Resource allocation
- Load balancing
- Health monitoring

Application Environment
w/ Multiple Containers



15

Container Orchestration Tools

16

Container Orchestration Tools



minikube



What is Kubernetes and How it works

Kubernetes



Kubernetes is an open-source container management tool which automates container deployment, container (de)scaling and container load balancing

- Benefits (Works with all cloud vendors (Public, Private (on-premises), and Hybrid))

More about Kubernetes

- Developed by Google and written in Golang with a huge community
- Can group 'n' containers into one logical unit for managing and deploying them



19

Kubernetes

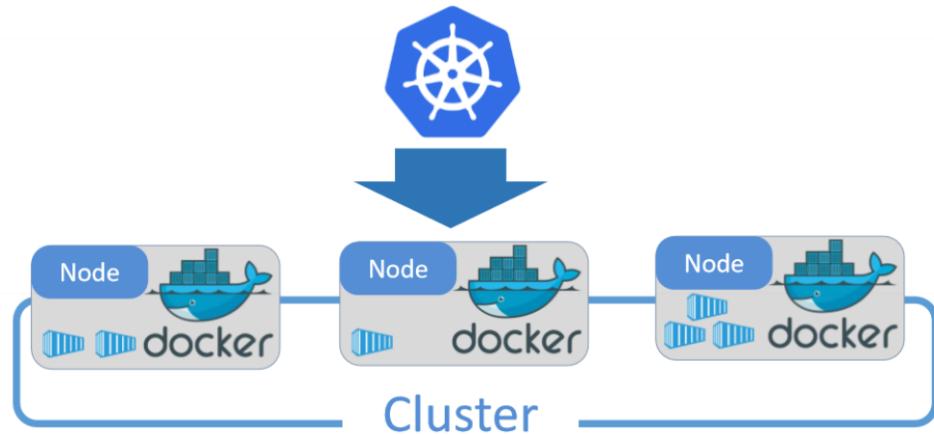
- Kubernetes is an open source container orchestration engine for automating deployment, scaling, and management of containerized application.
- Originally an open source project launched by Google and now part of the Cloud Native Computing Foundation (CNCF).
- Kubernetes is highly **extensible** and **portable**
- Kubernetes is considered **highly declarative**
- Kubernetes initial release (**7 June 2014**)
- Releases every **3 months**



kubernetes

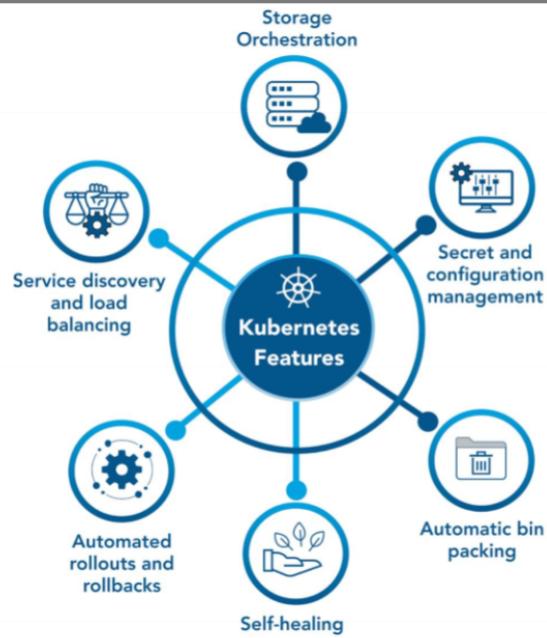
20

Kubernetes



21

Kubernetes Features



22

Kubernetes Myths

- Kubernetes is not:
 - To be compared with Docker
 - For containerizing apps
 - For apps with simple architecture



- Kubernetes is actually:
 - Robust and reliable
 - A container orchestration platform
 - A solution for scaling up Containers
 - Backed by huge community



Kubernetes vs Docker Swarm



Docker Swarm

- 1 No Auto Scaling
- 2 Good community
- 3 Easy to start a cluster
- 4 Limited to the Docker API's capabilities
- 5 Does not have as much experience with production deployments at scale

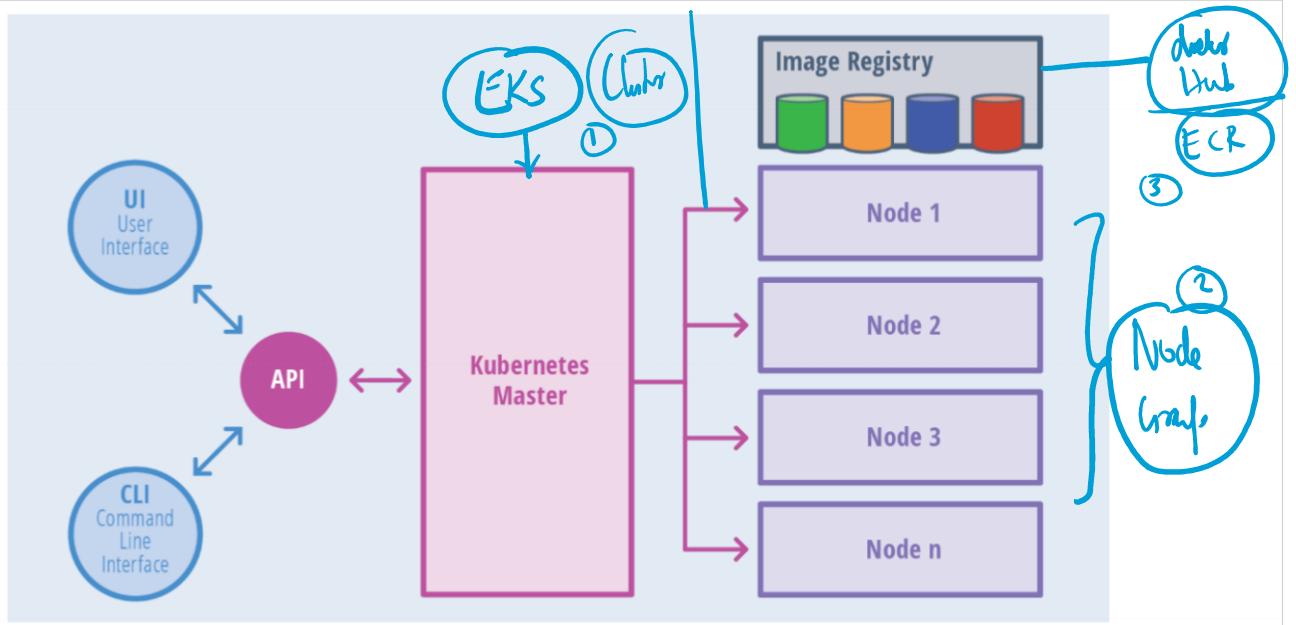
Kubernetes

- 1 Auto Scaling
- 2 Great active community
- 3 Difficult to start a cluster
- 4 Can overcome constraints of Docker and Docker API
- 5 Deployed at scale more often among organizations

Kubernetes Architecture

25

Kubernetes Architecture



26

Kubernetes Architecture

Kubernetes architecture

User interface



API



kubectl

Control plane

API Server

Scheduler

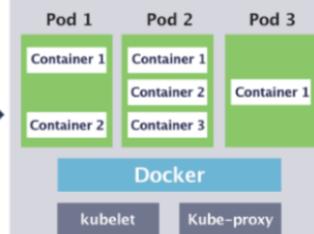
Controller-Manager

etcd

Worker node 1

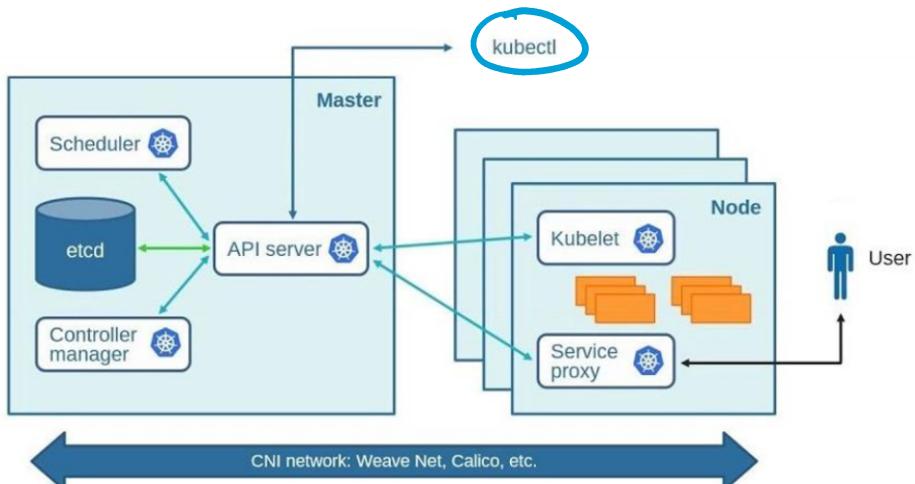


Worker node 2



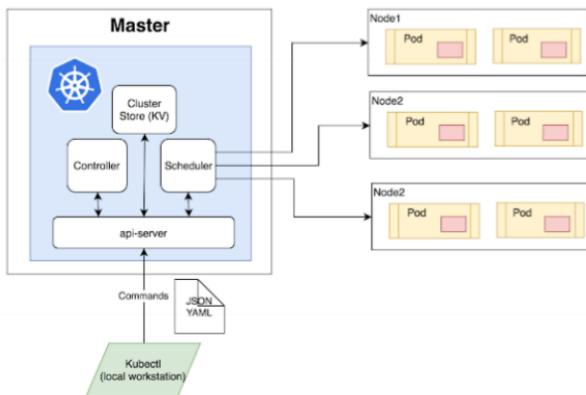
27

Kubernetes Architecture



28

Kubernetes Components

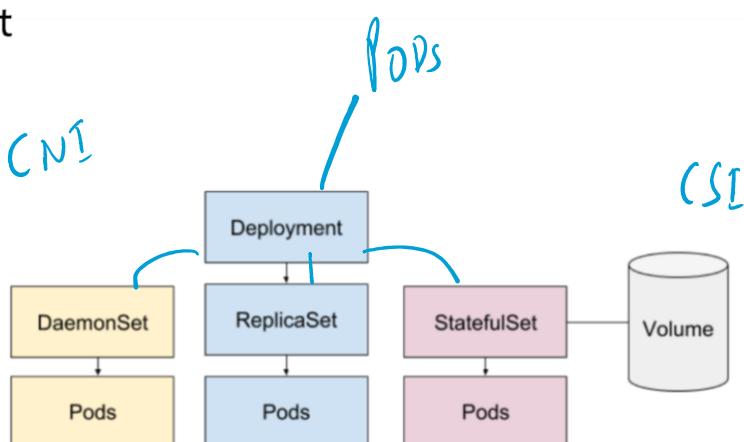


- Cluster:
 - A collection of hosts(servers) that aggregates their available resources.
- Master:
 - A collection of components which make up the control panel of Kubernetes.
- Node:
 - A single host which is capable of running on a physical or virtual machine.
- Namespace:
 - A logical cluster or environment.

29

Basic Kubernetes Objects

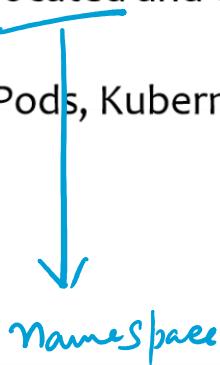
- Pod
- Deployment, ReplicaSet
- DaemonSet
- StatefulSet
- Service
- Secret



30

Pods

- Pods are the smallest deployable units of computing that you can create and manage in Kubernetes.
 - A Pod is a group of one or more containers, with shared storage/network resources, and a specification for how to run the containers.
 - A Pod's contents are always co-located and co-scheduled. *Shared capacity*
- To create and manage multiple Pods, Kubernetes defines multiple resource types:
- Deployment
 - StatefulSet
 - DaemonSet



Deployment

- Deployment: represents a set of multiple, identical Pods with no unique identities.
 - It runs multiple replicas Pods and automatically replaces any instances that fail or become unresponsive.
 - Deployments ensure that one or more instances of Pods are available to serve user requests.
- PodTemplates are specifications for creating Pods, and are included in resource objects such as Deployment object.
- A ReplicaSet is the next-generation of ReplicationControllers
 - It ensures that a specified number of pod replicas are running at any given time.

31

32

Pod Management

- StatefulSet: manages deployment and scaling of a set of Pods, with durable storage and persistent identifiers for each pod.
 - Unlike a Deployment, a StatefulSet maintains a sticky identity for each of its Pods.
- DaemonSet: ensures that all (or some) nodes run a copy of a Pod.
 - As nodes are added to the cluster, Pods are added to them.
 - As nodes are removed from the cluster, those Pods are garbage collected.
 - Deleting a DaemonSet will clean up the Pods it created.

33

Controller

- Controllers are control loops that watch the state of the cluster, then make or request changes where needed.
- A controller tracks at least one Kubernetes resource object.
- The controller(s) for that resource are responsible for making the current state come closer to that desired state (specified in the spec field).
- Kubernetes comes with a set of controllers that run inside the `kubecontroller-manager`
- The Deployment controller is an example of controller that come as part of Kubernetes itself ("built-in" controllers).

34

Service

- The services model relies upon the most basic, though most important,

Service

- The services model relies upon the most basic, though most important, aspect of services: **discovery**.
- a Service is an abstraction which defines a logical set of Pods and a policy by which to access them.
- The set of Pods targeted by a Service is usually determined by a **selector**.
- Services ensure that traffic is always routed to the appropriate Pod within the cluster, regardless of the node on which it is scheduled.
- Each service exposes an **IP address**, and may also expose a **DNS endpoint**, both of which will never change.
 - Internal or external consumers that need to communicate with a set of pods will use the service's IP address, or its more generally known DNS endpoint.

35

Secret

- Secrets are secure objects which store sensitive data, such as passwords, OAuth tokens, and SSH keys, in your clusters.
- Storing sensitive data in Secrets is more secure than plaintext in Pod specifications.
- Using Secrets gives you control over how sensitive data is used, and reduces the risk of exposing the data to unauthorized users.

36

Kubernetes Cluster

