KUBERNETES

Why ?

Cluster

- Without Kubernetes: installation scripts would often be used to start applications, but they did not allow recovery from machine failure.
- With Kuberneted deployment: creates your application instances and keeps them running across Nodes.

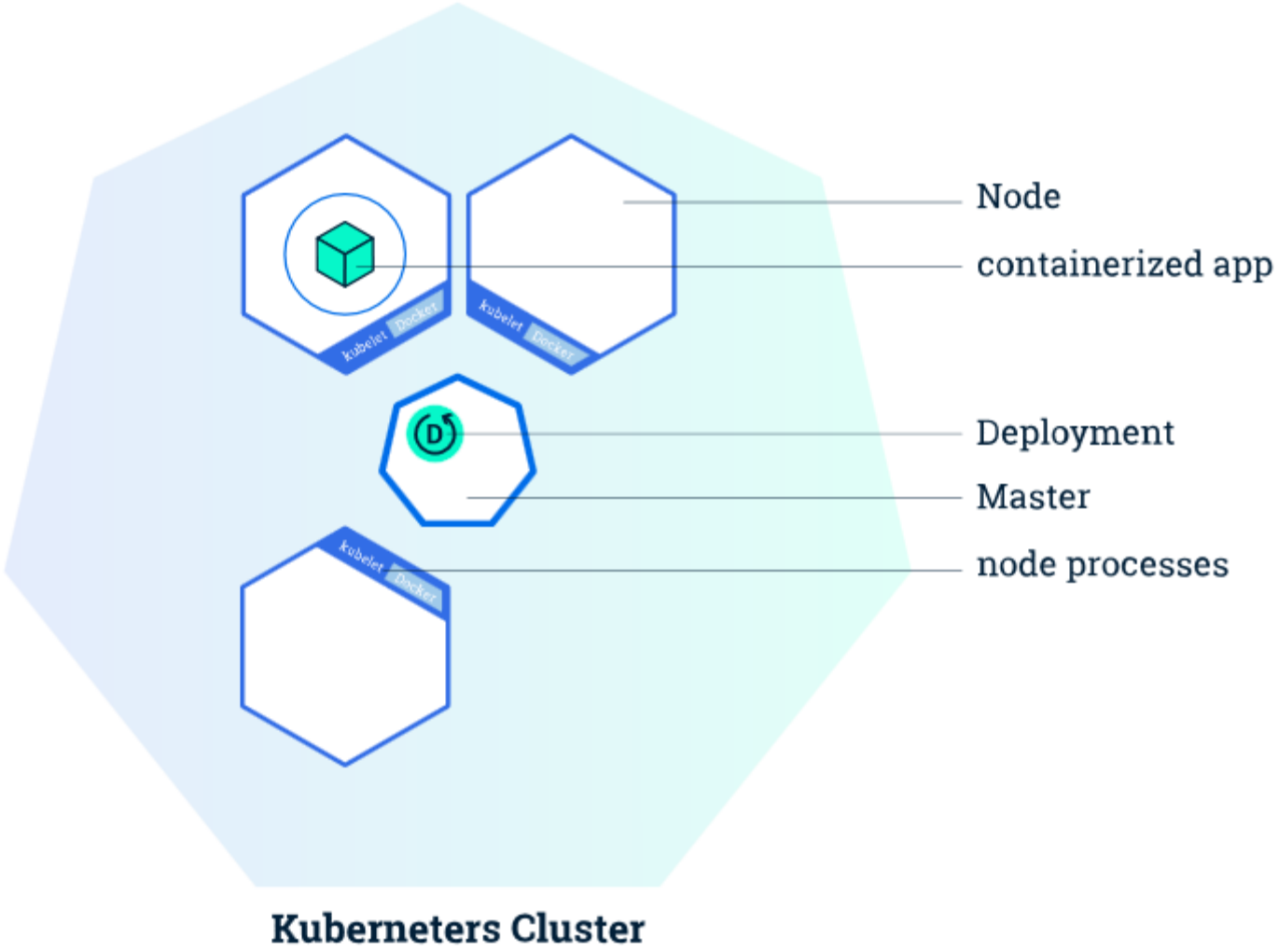
- With modern web services:
    - users expect applications to be available 24/7
    - developers expect to deploy new versions of those applications several times a day.
- Containerization helps package software to serve these goals, enabling applications to be released and updated in an easy and fast way without downtime.
- Kubernetes helps you make sure those containerized applications run where and when you want, and helps them find the resources and tools they need to work.

Cluster ?

Cluster

- Kubernetes coordinates a highly available cluster of computers that are connected to work as a single unit.
- Applications need to be packaged in a way that decouples them from individual hosts: they need to be containerized.
- Containerized applications are more flexible and available than in past deployment models, where applications were installed directly onto specific machines as packages deeply integrated into the host.
- **Kubernetes automates the distribution and scheduling of application containers across a cluster in a more efficient way.**
- A Kubernetes cluster can be deployed on either physical or virtual machines.
- Kubernetes is an open source platform and is production-ready.
- Kubernetes cluster consists of two types of resources:
    - The **Master** coordinates the cluster
    - **Nodes** are the workers that run applications

Cluster


Node
containerized app
Deployment
Master
node processes
kubelet
Docker
Kuberneters Cluster

Cluster

- **The Master is responsible for managing the cluster :** scheduling applications, maintaining applications' desired state, scaling applications...
- **A node is a VM or a physical computer that serves as a worker machine in a Kubernetes cluster.**
    - Each node has a Kubelet, which is an agent for managing the node and communicating with the Kubernetes master. It manages the Pods and the containers running on a machine.
    - A container runtime (like Docker, rkt) responsible for pulling the container image from a registry, unpacking the container, and running the application.
- A Kubernetes cluster that handles production traffic should have a minimum of three nodes.

Solutions

Solutions

- Kubernetes is open source
- Local machine solutions:
    - Minikube recommended for a single node
    - Ubuntu or IBM
- Installation for open source:
    - Kubernetes: k8s
    - k3s: ligthweight solution for edge computing, IoT, Raspberry Pi…
- Hosted solutions:
    - Google container
    - Azure container
    - …
- Turnkey solutions (IaaS providers with a few command):
    - Google Compute Engine
    - AWS
    - …
- …

Command line interface

Kubectl

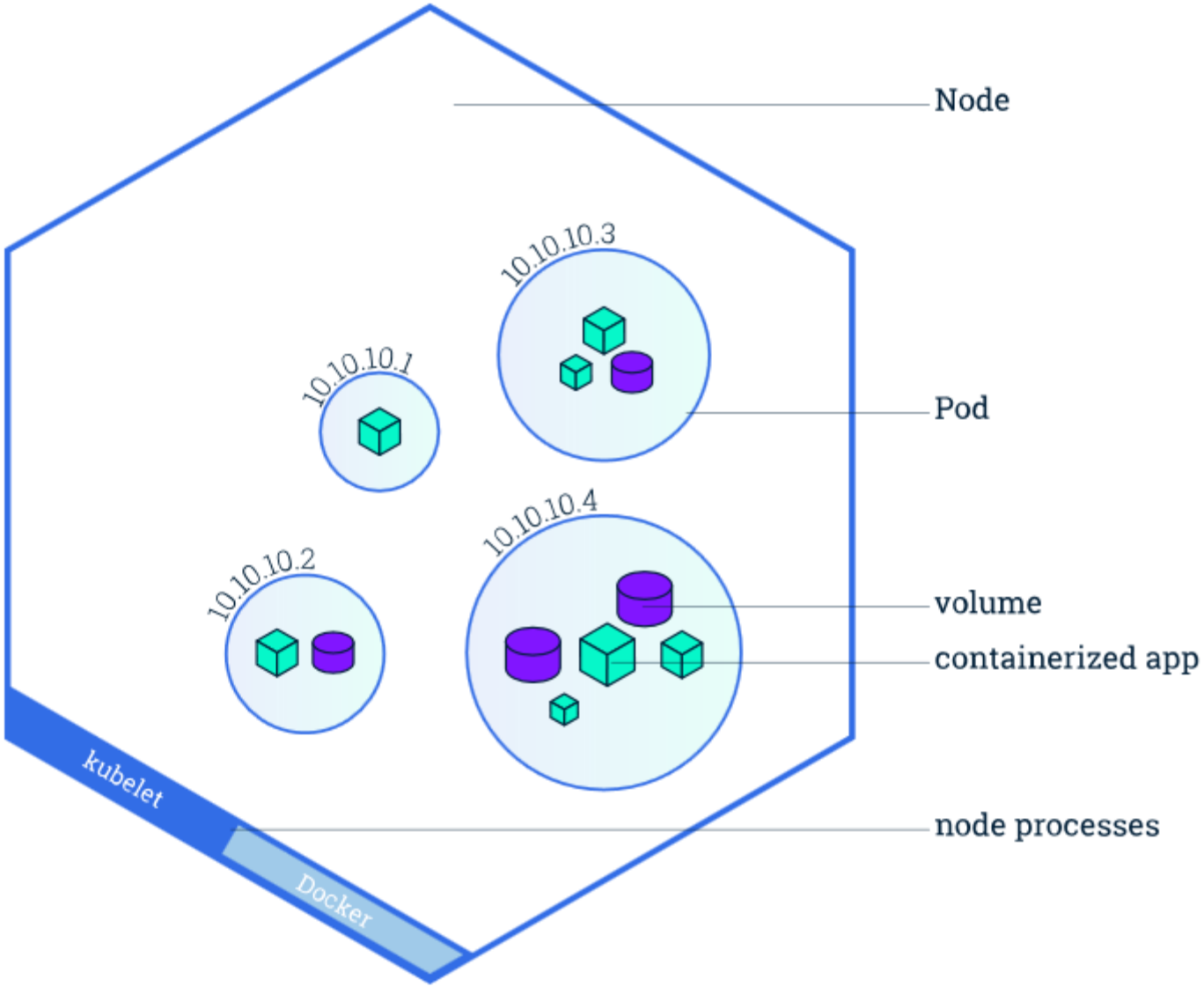
- You can create and manage a Deployment by using the Kubernetes command line interface, Kubectl. Kubectl uses the Kubernetes API to interact with the cluster. In this module, you'll learn the most common Kubectl commands needed to create Deployments that run your applications on a Kubernetes cluster.
- When you create a Deployment, you'll need to specify the container image for your application and the number of replicas that you want to run. You can change that information later by updating your Deployment
- When you created a Deployment in Module 2, Kubernetes created a Pod to host your application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers. Those resources include:
    - Shared storage, as
    - Volumes Networking
    - a unique cluster IP address
    - Information about how to run each container, such as the container image version or specific ports to use

Kubectl

- View nodes in the cluster:
    - kubectl get nodes
- Let’s run our first app on Kubernetes with the kubectl run command. The run command creates a new deployment. We need to provide the deployment name and app image location (include the full repository url for images hosted outside Docker hub). We want to run the app on a specific port so we add the --port parameter:
    - kubectl run kubernetes-bootcamp –image=docker.io/jocatalin/kubernetes-bootcamp:v1 --port=8080
    - Kubectl get deployments

Pods

Pods


Node
10.10.10.3
10.10.10.1
Pod
10.10.10.4
10.10.10.2
volume
containerized app
kubelet
Docker
node processes

Pods

- Pods are the atomic unit on the Kubernetes platform. When we create a Deployment on Kubernetes, that Deployment creates Pods with containers inside them (as opposed to creating containers directly). Each Pod is tied to the Node where it is scheduled, and remains there until termination (according to restart policy) or deletion. In case of a Node failure, identical Pods are scheduled on other available Nodes in the cluster.
- When you created a Deployment in Module 2, Kubernetes created a Pod to host your application instance. A Pod is a Kubernetes abstraction that represents a group of one or more application containers (such as Docker or rkt), and some shared resources for those containers. Those resources include:
    - Shared storage, as
    - Volumes Networking
    - a unique cluster IP address
    - Information about how to run each container, such as the container image version or specific ports to use

Usual workflow

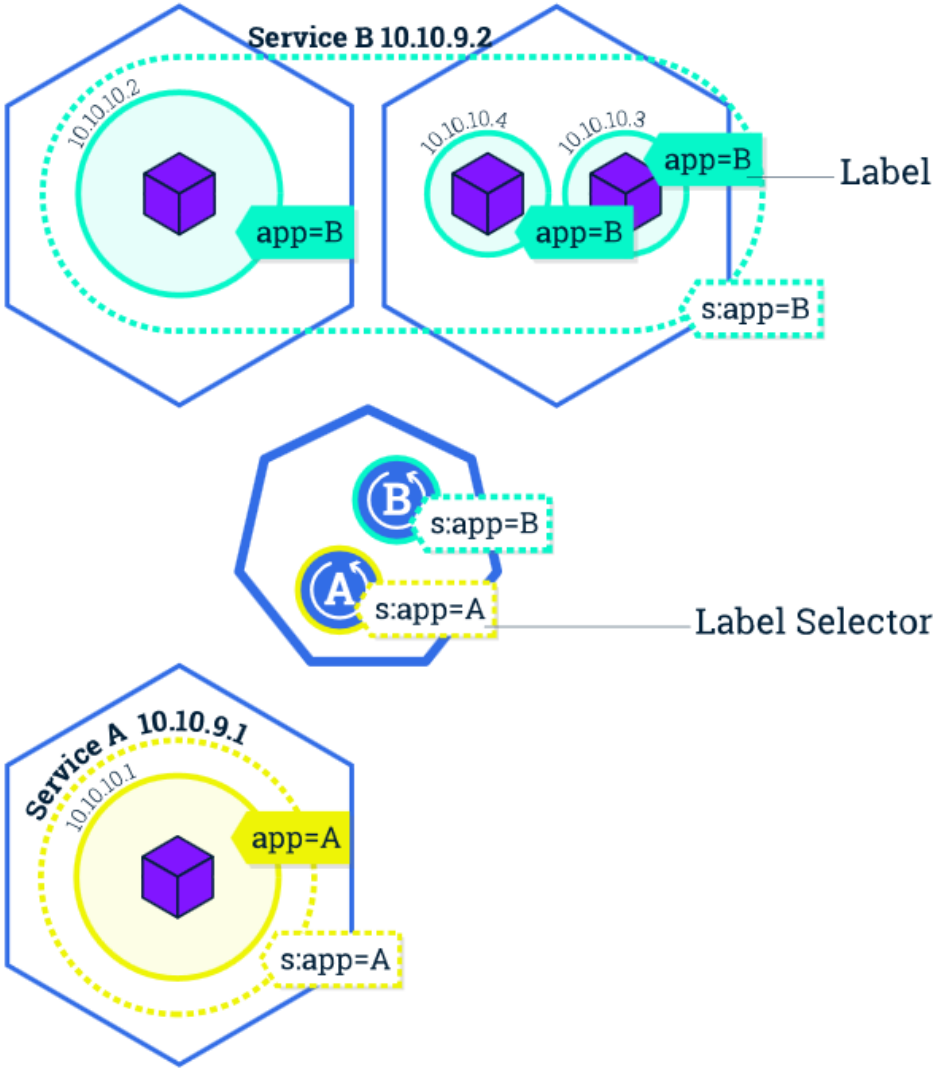
- Deploy a containerized application on a cluster
- Scale the deployment
- Update the containerized application with a new software versio
- Debug the containerized application

Get information about applications

- get information about deployed applications and their environments. The most common operations can be done with the following kubectl commands:
    - kubectl get - list resources
    - kubectl describe - show detailed information about a resource
    - kubectl logs - print the logs from a container in a pod
    - kubectl exec - execute a command on a container in a pod

Services

Services


Service B 10.10.9.2
10.10.10.2
app=B
10.10.10.4
10.10.10.3
app=B
app=B
Label
s:app=B
B
s:app=B
A
s:app=A
Label Selector
Service A 10.10.9.1
10.10.10.1
app=A
s:app=A

Service

- A Kubernetes Service is an abstraction layer which defines a logical set of Pods and enables external traffic exposure, load balancing and service discovery for those Pods.
- Kubernetes Pods are mortal. Pods in fact have a lifecycle. When a worker node dies, the Pods running on the Node are also lost. A ReplicationController might then dynamically drive the cluster back to desired state via creation of new Pods to keep your application running.
- Although each Pod has a unique IP address, those IPs are not exposed outside the cluster without a Service. Services allow your applications to receive traffic.
- A Service routes traffic across a set of Pods. Services are the abstraction that allow pods to die and replicate in Kubernetes without impacting your application. Discovery and routing among dependent Pods (such as the frontend and backend components in an application) is handled by Kubernetes Services.

Service

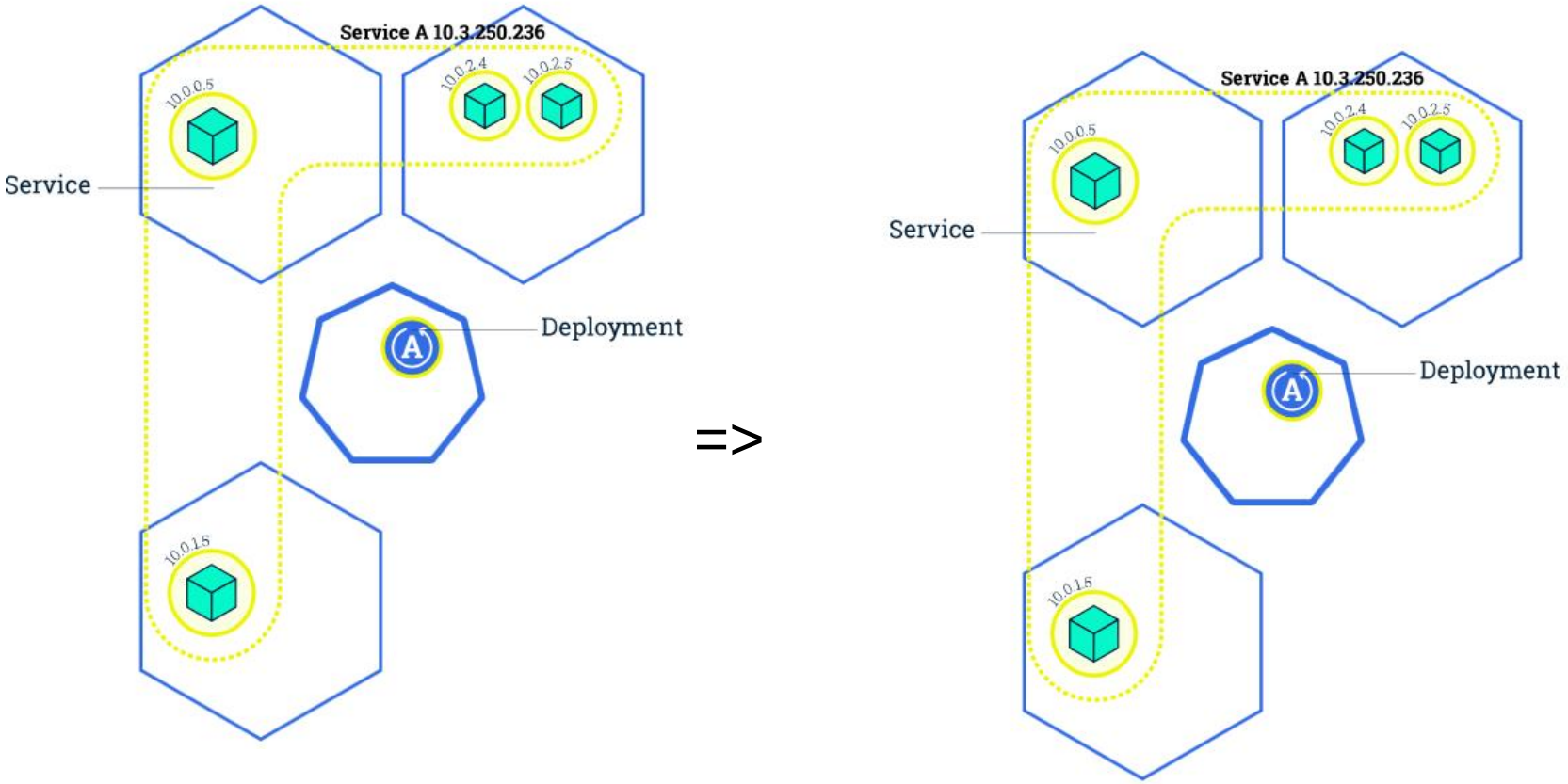
- You can create a Service at the same time you create a Deployment by using --expose in kubectl
- Create a service: to create a new service and expose it to external traffic we'll use the expose command with NodePort as parameter (minikube does not support the LoadBalancer option yet):
    - kubectl expose deployment/kubernetes-bootcamp --type="NodePort" --port 8080
    - kubectl get services
    - We have now a running Service called kubernetes-bootcamp. Here we see that the Service received a unique cluster-IP, an internal port and an external-IP (the IP of the Node).
    - To find out what port was opened externally (by the NodePort option):
    - kubectl describe services/kubernetes-bootcamp

Service

- Create an environment variable called NODE_PORT that has as value the Node port: export NODE_PORT=$(kubectl get services/kubernetes-bootcamp -o go-template='{{(index .spec.ports 0).nodePort}}') echo NODE_PORT=$NODE_PORT
- Now we can test that the app is exposed outside of the cluster using: curl host01:$NODE_PORT

Scale an app

Scale an application



Scale an application

- When traffic increases, we will need to scale the application to keep up with user demand.
- Scaling is accomplished by changing the number of replicas in a Deployment
- You can create from the start a Deployment with multiple instances using the --replicas parameter for the kubectl run comman
- Scaling up a Deployment will ensure new Pods are created and scheduled to Nodes with available resources. Scaling down will reduce the number of Pods to the new desired state. Kubernetes also supports autoscaling of Pods.
- Running multiple instances of an application will require a way to distribute the traffic to all of them. Services have an integrated load-balancer that will distribute network traffic to all Pods of an exposed Deployment. Services will monitor continuously the running Pods using endpoints, to ensure the traffic is sent only to available Pods.

Scale an application

- kubectl get deployments
  - The DESIRED state is showing the configured number of replicas
  - The CURRENT state show how many replicas are running now
  - The UP-TO-DATE is the number of replicas that were updated to match the desired (configured) state
  - The AVAILABLE state shows how many replicas are actually AVAILABLE to the users
- let's scale the Deployment to 4 replicas:
  - kubectl scale deployments/kubernetes-bootcamp --replicas=4
- Next, let's check if the number of Pods changed:
- kubectl get pods –o wide

Rolling update

- Users expect applications to be available all the time and developers are expected to deploy new versions of them several times a day. In Kubernetes this is done with rolling updates. **Rolling updates** allow Deployments' update to take place with zero downtime by incrementally updating Pods instances with new ones. The new Pods will be scheduled on Nodes with available resources.
- In the previous module we scaled our application to run multiple instances. This is a requirement for performing updates without affecting application availability. By default, the maximum number of Pods that can be unavailable during the update and the maximum number of new Pods that can be created, is one.

Rolling update