The Example with Kubernetes

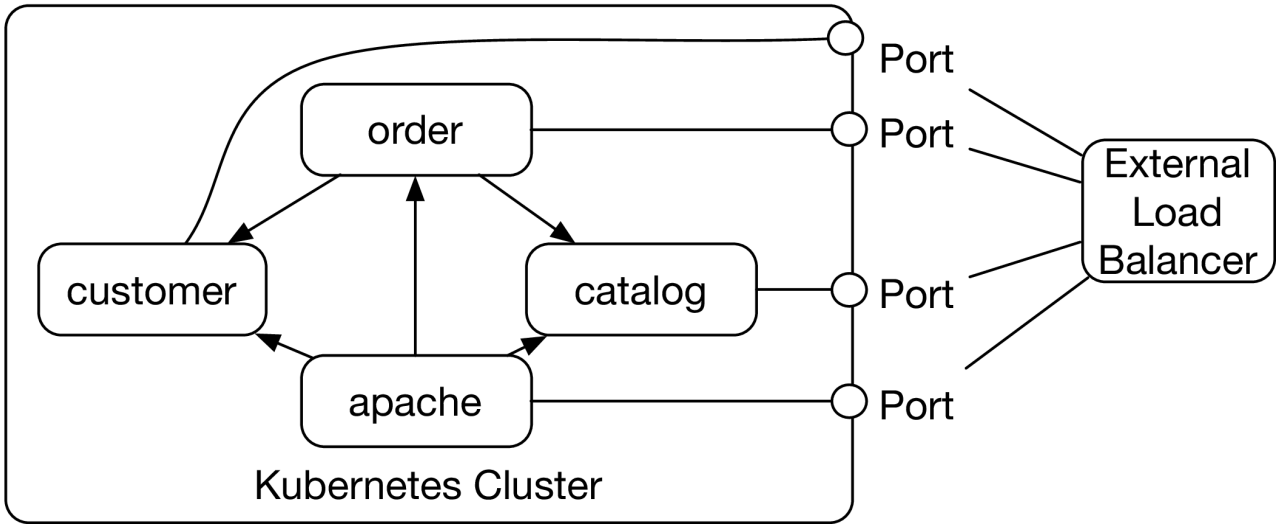
In this lesson, we'll introduce an example with Kubernetes.

We'll cover the following

- Introduction
- Implementing microservices with Kubernetes
- Service discovery
- Fail-safety
- Load balancing
- Service discovery, fail-safety, and load balancing without code dependencies
- Routing with Apache httpd
- Routing with node ports
- Routing with load balancers
- Routing with Ingress

Introduction

The services in this example are identical with the examples presented in the two preceding chapters (see Example (https://www.educative.io/collection/page/10370001/5441945024331776/6627192651907072)).



The Microservices System in Kubernetes

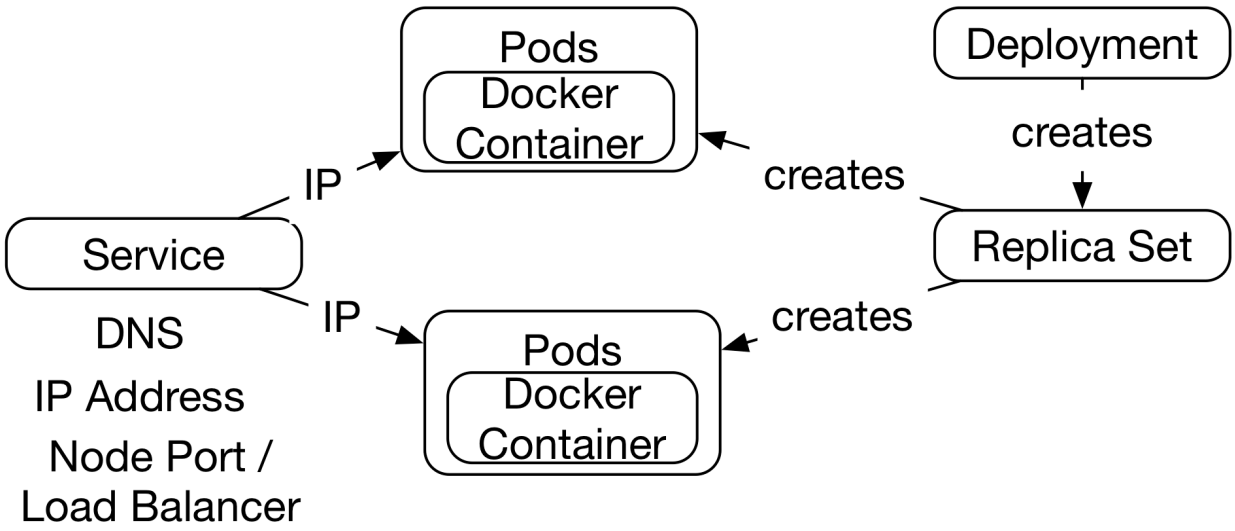
- The **catalog** microservice manages the information about the items. It provides an HTML UI and a REST interface.
- The **customer** microservice stores the customer data and also provides an HTML UI and a REST interface.
- The **order** microservice can receive new orders. It provides an HTML UI and uses the REST interfaces of the catalog and customer microservice.
- An **Apache web server** facilitates access to the individual microservices. It forwards the calls to the respective services.

The drawing above shows how the microservices interact. The order microservice communicates with the catalog and the customer microservice while the Apache httpd server communicates with all other microservices to display the HTML UIs.

The microservices are accessible from the outside via node ports. On each node in the cluster, a request to a specific port will be forwarded to the service. However, the port numbers are assigned by Kubernetes, so there is no port number in the figure.

A load balancer is set up by the Kubernetes service to distribute the load across Kubernetes nodes.

Implementing microservices with Kubernetes



A Microservice in Kubernetes

The drawing above shows the interaction of the Kubernetes components for a microservice.

- A **deployment** creates a **replica set** with the help of Docker images.
- The **replica set** starts one or multiple pods.
- The **pods** in the example only comprise a single Docker container in which the microservice is running.

Service discovery

A **service** makes the replica set **accessible**:

1. The service provides the pods with an IP address and a DNS record.

2. Other pods communicate with the service by reading the IP address from the DNS record.

3. Thereby, Kubernetes implements **service discovery with DNS**.

In addition, microservices receive the IP addresses of other microservices via **environment variables**. Thus, they could also use this information to access the service.

Fail-safety

The microservices are so fail-safe because the replica set ensures that a certain number of pods is always running. **If a pod fails, a new one is started.**.

Load balancing

Load balancing is also covered. The number of pods is determined by the replica set meaning that the service implements load balancing. All pods can be accessed with the same IP address which the service defines. Requests go to this IP address but **are distributed to all instances**.

The service implements this functionality by interfering with the IP network between the Docker containers. Since the IP address is cluster-wide unique, this mechanism works even if the pod is moved from one node to another.

Kubernetes **does not implement load balancing at the DNS level**. If it did, different IP addresses would be returned for the same service name for each DNS lookup so that the load would be distributed during DNS access.

However, this kind of approach presents a number of challenges. DNS supports caching and if a different IP address is to be returned each time

supports caching and if a different IP address is to be returned each time a DNS access occurs, the caching must be configured accordingly.

However, problems still occur because caches are not invalidated in time.

Service discovery, fail-safety, and load balancing without code dependencies

For load balancing and service discovery, no special code is necessary.

A URL like http://order:8080/ (http://order:8080/) suffices. Accordingly, the microservices do not use any special Kubernetes APIs. There are no code dependencies to Kubernetes and no specific Kubernetes libraries are used.

Routing with Apache httpd

In the example, Apache httpd is configured as a reverse proxy. It therefore routes access from the outside to the correct microservice. Apache httpd leaves load balancing, service discovery, and fail-safety to the Kubernetes infrastructure.

Routing with node ports

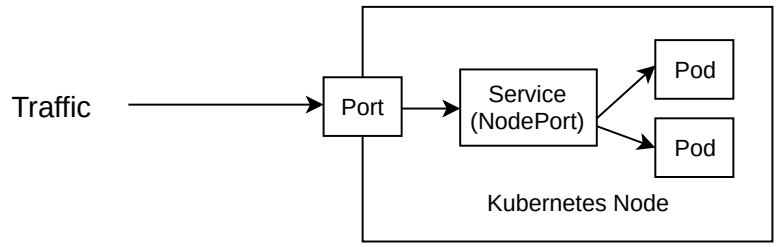
Services also offer a solution for routing, i.e., external access to microservices.

1. The service generates a node port

1. The service generates a node port.
2. Under this port, the services are available on every Kubernetes node.
3. If the pod that implements the service is not available on the called Kubernetes node, Kubernetes forwards a request to the node port to another Kubernetes node where the pod is running.

In this way, an external load balancer can distribute the load to the nodes in the Kubernetes cluster.

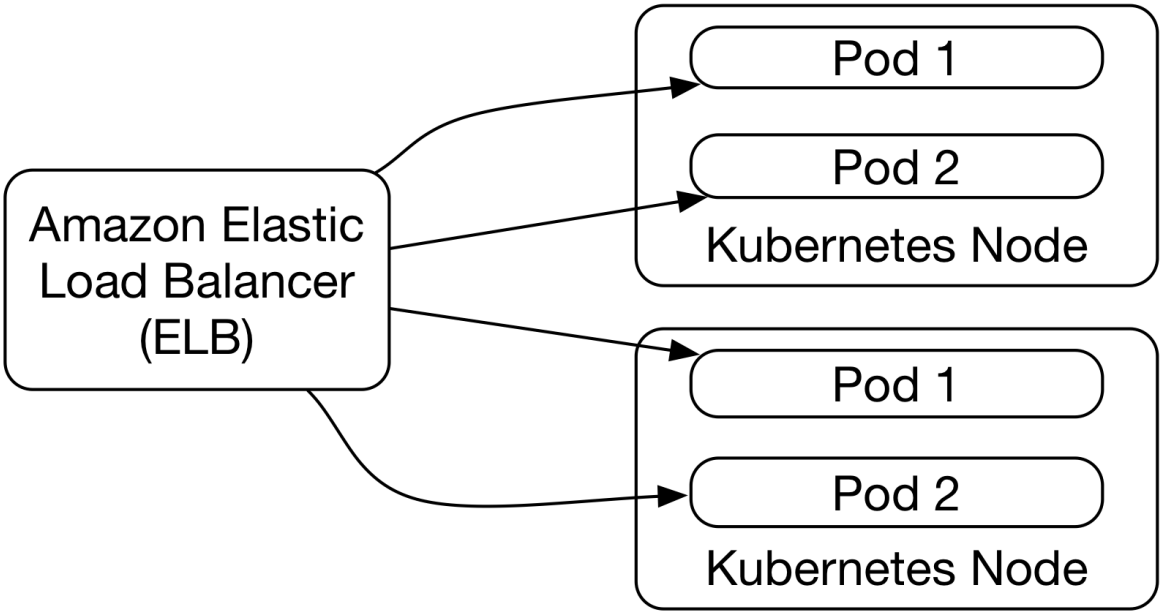
The requests are simply distributed to the node port of the service on all nodes in the cluster. The service type `NodePort` is used for this purpose, which must be specified when creating the service.



Routing with node ports

Routing with load balancers

Kubernetes can create load balancers in a Kubernetes production environment. In an Amazon environment, for example, Kubernetes configures an ELB (Elastic Load Balancer) to access the node ports in the cluster. The service type `LoadBalancer` is used for this purpose.



Kubernetes Service Type LoadBalancer in the Amazon Cloud

The services in the example are of the type `LoadBalancer`. However, if they run on Minikube, they are treated as services of type `NodePort` because Minikube cannot configure and provide load balancers.

Routing with Ingress

Kubernetes offers an extension called Ingress (https://kubernetes.io/docs/concepts/services-networking/ingress/), which can **configure and alter the access of services from the Internet**.

Ingress can also implement:

- load balancing
- terminate SSL
- implement virtual hosts

This behavior is implemented by an **Ingress controller**.

In the example, the Apache web server forwards requests to the

individual microservices. This could also be done with a **Kubernetes Ingress**. Then the routing would be done by a part of the Kubernetes infrastructure which might work better.

The **configuration is done in Kubernetes YAML files** and supports other Kubernetes parts like services. However, the example contains a few static web pages that the Apache web server provides. So, the Apache web server has to be configured anyways.

Using an Ingress is therefore not a huge advantage.

QUIZ

1 How are the microservices in the example accessible from outside?

- A) Through their individual ports via the load balancer.
- B) Through their individual ports via the apache server and the load balancer.
- C) Through their node ports.

D) Both A and B.

E) Both B and C.

Submit Answer

Question 1 of 2
0 attempted

Reset Quiz

In the next lesson, we'll discuss this same example in much more detail.

← Back

Introduction

Next →

The Example in Detail

Mark as Completed

Report an Issue