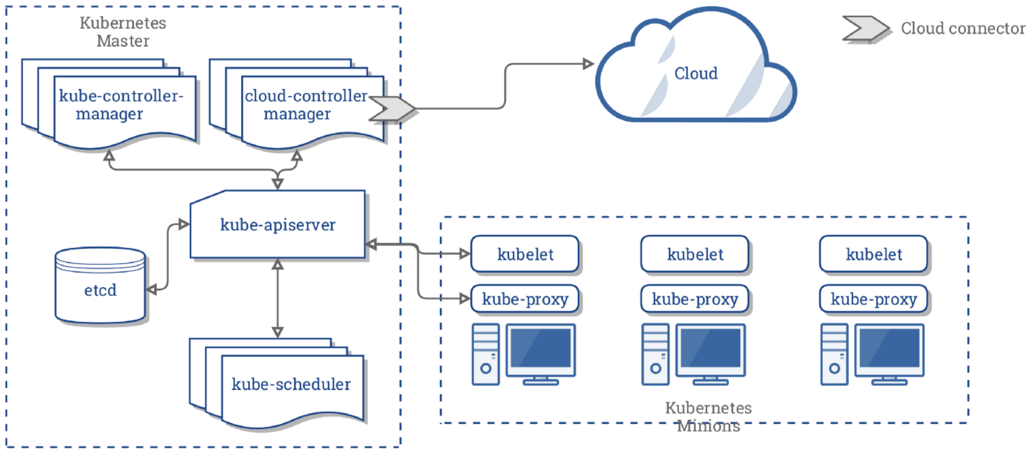
Kubernetes Overview

Overview

Kubernetes, k8s, or kube, is an open source platform that automates container operations. It eliminates most of the existing manual processes, which involve the deploying, scaling, and managing of containerized applications.

Components



Kubernetes Master

The Kubernetes master is responsible for maintaining the desired state for your cluster. When you interact with Kubernetes, such as by using the `kubectl` command-line interface, you're communicating with your cluster's Kubernetes master.

The "master" refers to a collection of processes managing the cluster state. Typically these processes are all run on a single node in the cluster, and this node is also referred to as the master. The master can also be replicated for availability and redundancy.

The **Kubernetes Master** is a collection of three processes that run on a single node in your cluster, which is designated as the master node.

Those processes are:

- kube-apiserver: the single point of management for the entire cluster. The API server implements a RESTful interface for communication with tools and libraries. The `kubectl` command directly interacts with the API server.
- kube-controller-manager: regulates the state of the cluster by managing the different kinds of controllers.
- kube-scheduler: schedules the workloads across the available nodes in the cluster.

Kubernetes Nodes

The nodes in a cluster are the machines (VMs, physical servers, etc) that run your applications and cloud workflows. The Kubernetes master controls each node; you'll rarely interact with nodes directly.

Each individual non-master node in your cluster runs two processes:

- kubelet, which communicates with the Kubernetes Master.
- kube-proxy, a network proxy which reflects Kubernetes networking services on each node.

Objects

The basic Kubernetes objects are as follows:

- Pod is the smallest deployable unit on a Node. It's a group of containers which must run together. Quite often, but not necessarily, a Pod usually contains one container.
- Service is used to define a logical set of Pods and related policies used to access them.
- Volume is essentially a directory accessible to all containers running in a Pod.
- Namespaces are virtual clusters backed by the physical cluster.

There are a number of Controllers provided by Kubernetes. These Controllers are built upon the basic Kubernetes Objects and provide additional features. The Kubernetes Controllers include:

- ReplicaSet ensures that a specified number of Pod replicas are running at any given time.
- Deployment is used to change the current state to the desired state.
- StatefulSet is used to ensure control over the deployment ordering and access to volumes, etc.
- DaemonSet is used to run a copy of a Pod on all the nodes of a cluster or on specified nodes.
- Job is used to perform some task and exit after successfully completing their work or after a given period of time.

Describing an Object

When you create an object in Kubernetes, you must provide the object spec that describes its desired state, as well as some basic information about the object (such as a name). When you use the Kubernetes API to create the object (either directly or via kubectl), that API request must include that information as JSON in the request body. Most often, you provide the information to kubectl in a .yaml file. kubectl converts the information to JSON when making the API request.

Here's an example .yaml file that shows the required fields and object spec for a Kubernetes Deployment:

application/deployment.yaml

```yaml
apiVersion: apps/v1 # for versions before 1.9.0 use apps/v1beta2
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods matching the template
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.7.9
        ports:
        - containerPort: 80
```

One way to create a Deployment using a .yaml file like the one above is to use the kubectl create command in the kubectl command-line interface, passing the .yaml file as an argument. Here's an example:

```
$ kubectl create -f https://k8s.io/examples/application/deployment.yaml --record
```

The output is similar to this:

deployment.apps/nginx-deployment created

Services

A Kubernetes Service is an abstraction which defines a logical set of Pods and a policy by which to access them - sometimes called a micro-service.

Defining a Service

```
kind: Service
apiVersion: v1
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
  - protocol: TCP
    port: 80
    targetPort: 9376
```

If we want to define a service which connects to outside IP addresses (maybe for a database in production), we would not specify a selector for our service and we would define an endpoint.

```
kind: Service
apiVersion: v1
metadata:
  name: customer-db
spec:
  ports:
  - protocol: TCP
    port: 154321
    targetPort: 54321
```

Since no selector is specified, kubernetes will not create an Endpoint so we will have to.

```
kind: Endpoints
apiVersion: v1
metadata:
  name: customer-db
subsets:
  - addresses:
      - ip: 1.2.3.4
    ports:
      - port: 54321
```

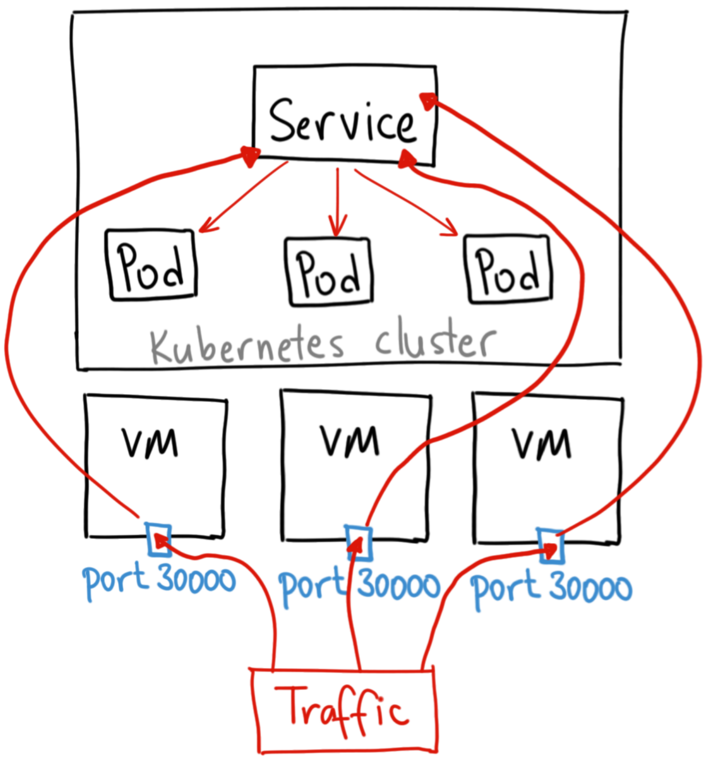
Types of Services

ClusterIP

Exposes the service on a cluster-internal IP. Choosing this value makes the service only reachable from within the cluster. This is the default ServiceType.

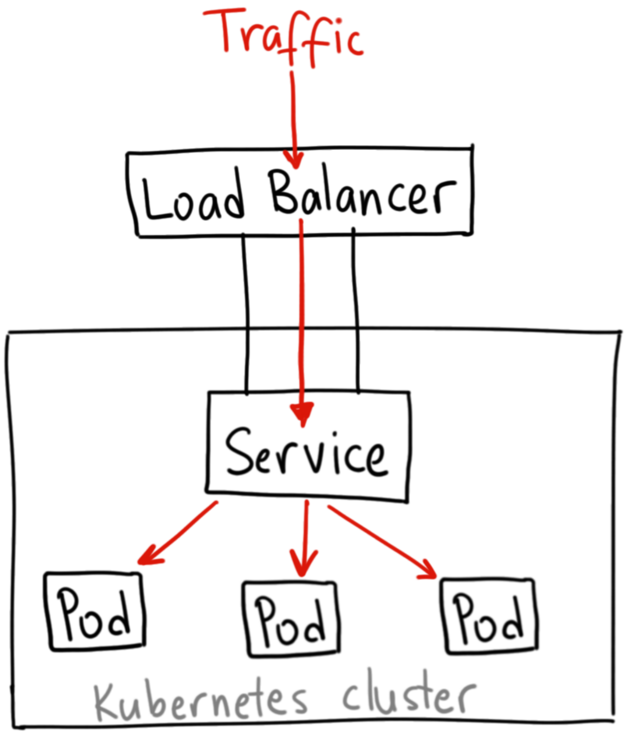
NodePort

Exposes the service on each Node’s IP at a static port (the `NodePort`). A `ClusterIP` service, to which the `NodePort` service will route, is automatically created. You’ll be able to contact the `NodePort` service, from outside the cluster, by requesting `<NodeIP>:<NodePort>`



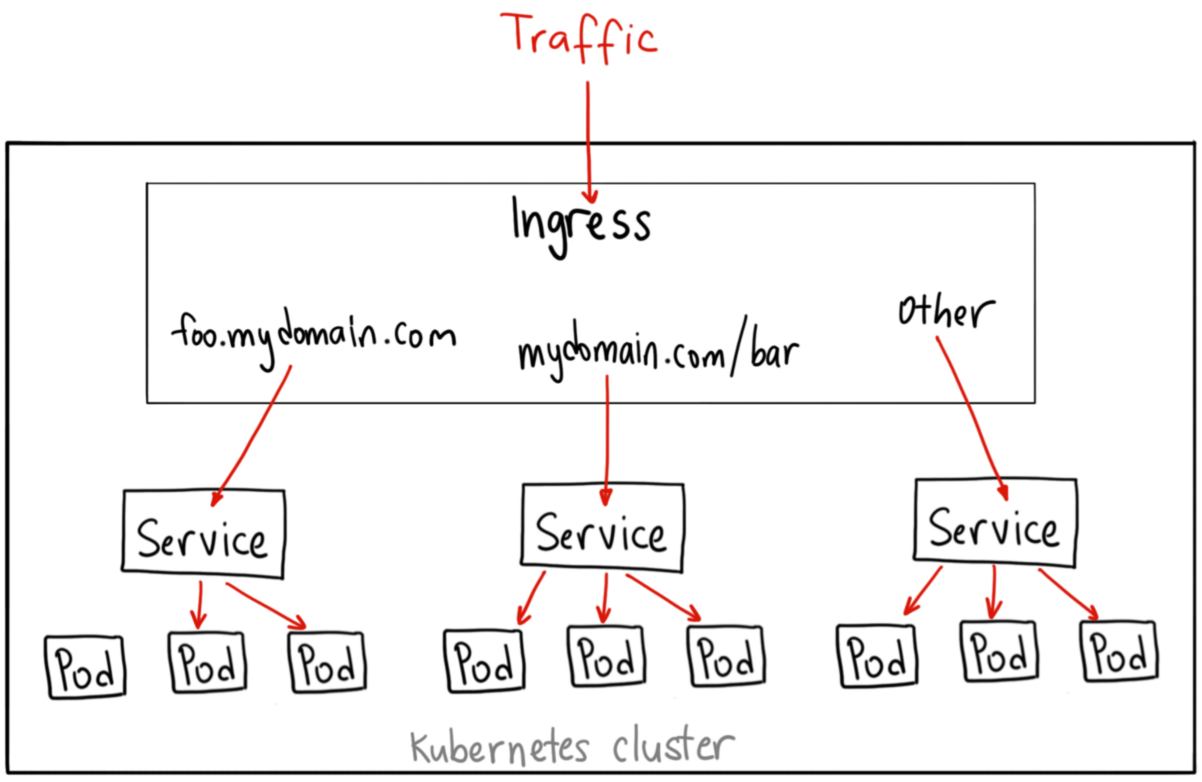
LoadBalancer

Exposes the service externally **using a cloud provider's load balancer**. `NodePort` and `ClusterIP` services, to which the external load balancer will route, are automatically created.



Ingress

Ingress, added in Kubernetes v1.1, exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the ingress resource.



An ingress can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, and offer name based virtual hosting. An ingress controller is responsible for fulfilling the ingress, usually with a loadbalancer, though it may also configure your edge router or additional frontends to help handle the traffic.

An ingress does not expose arbitrary ports or protocols. Exposing services other than HTTP and HTTPS to the internet typically uses a service of type Service.Type=NodePort or Service.Type=LoadBalancer.

Ingress providers:

https://kubernetes.io/docs/concepts/services-networking/ingress/#ingress-controllers

References

Reference	URL
A friendly introduction to Kubernetes	https://medium.freecodecamp.org/a-friendly-introduction-to-kubernetes-670c50ce4542
Kubernetes Concepts	https://kubernetes.io/docs/concepts/
Load-Balancing in Kubernetes	https://rancher.com/load-balancing-in-kubernetes/
Kubernetes NodePort vs LoadBalancer vs Ingress? When should I use what?	https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0
What is Ingress?	https://kubernetes.io/docs/concepts/services-networking/ingress/#what-is-ingress
Configuration Best Practices	https://kubernetes.io/docs/concepts/configuration/overview/