Running Containers and Exposing Services I

Minikube

Minikube

> Minikube is a tool that makes it easy to run Kubernetes locally. Minikube runs a single-node Kubernetes cluster inside a VM on your laptop

Current Features

- DNS
- NodePorts
- ConfigMaps and Secrets
- Dashboards
- Container Runtime: Docker, and rkt
- Enabling CNI (Container Network Interface)

Usage

```
minikube start
```

```
kubectl cluster-info

> Kubernetes master is running at https://192.168.42.2:8443
> kubernetes-dashboard is running at https://192.168.42.2:8443/api/v1/proxy/namesp
aces/kube-system/services/kubernetes-dashboard
```

Watch the dashboard in a browser: `bash minikube dashboard`

See what's running from the commandline: `bash kubectl get --all-namespaces pods`

```
minikube stop
```

```
minikube delete
```

Basic Concepts in Kubernetes I

Pods

Pods are the smallest deployable units of computing that can be created and managed in Kubernetes. They contain one or more containers that run inside the pod as if they were running on a single host.

All containers in a pod are scheduled together and the pod counts as running only when all of its containers are also running.

Pods are not the same as containers! Although they can contain a single container, they can also group several containers that run inside the pod as if they were running on the same host. Thus, containers in a post are able to talk to each other over `localhost` and share all namespaces. Further, all containers in a pod have access to shared volumes, that is they can mount and work on the same volumes if needed.

This grouping is not meant to be used to "deploy your whole LAMP stack together". The main motivation for the pod concept is supporting co-located, co-managed helper containers next to the application container. These include things like: logging or monitoring agents, backup tooling, data change watchers, event publishers, proxies, etc. If you are not sure what to use pods for in the beginning, you can for now use them with single containers like you might be used to from Docker.

The pod is the most basic concept in Kubernetes. By itself, it is ephemeral and won't be rescheduled to a new node once it dies. If we want to keep one or more instances of a pod alive we need another concept: replica sets. But before that we need to understand what labels and selectors are.

Labels & Selectors

Labels are key/value pairs that can be attached to objects, such as pods, but also any other object in Kubernetes, even nodes. They should be used to specify identifying attributes of objects that are meaningful and relevant to users. You can attach labels to objects at creation time and modify them or add new ones later.

Labels are a key concept of Kubernetes as they are used together with selectors to manage objects or groups thereof. This is done without the actual need for any specific information about the object(s), not even the number of objects that exist.

Especially the fact that the number of objects is unkown should be kept in mind when working with label selectors. In general, you should expect many objects to carry the same label(s).

Currently, there are two types of selectors in Kubernetes: equality-based and set-based. The former use key value pairs to filter based on basic equality (or inequality). The latter are a bit more powerful and allow filtering keys according to a set of values.

Using label selectors a client or user can identify and subsequently manage a group of objects. This is the core grouping primitive of Kubernetes and used in many places. One example of its use is working with replica sets.

Deployments (& Replica Sets)

Deployments are a declarative way of defining the deployment of pods onto the cluster. They manage replication as well as updates of these pods and keep an audit log of all changes.

As mentioned-above pods are ephemeral by themsleves and do not get rescheduled if they die. However, you should usually not create pods manually, but instead use a Deployment. A deployment defines both the specification of the pod to be deployed as well as a so-called replica set, which controls the replication of the pod. It holds information like number of replicas (instances) and more.

When the deployment gets created it creates a replica set, which in turns creates pods in the cluster. Thus, the chain is like following: Deployment -> Replica Set -> Pod(s).

The replica set ensures that a specific number of pods (desired state) is always running in the cluster.

Services

Services work by defining a logical set of pods and a policy by which to access them. The selection of pods is based on label selectors. In case you select multiple pods, the service automatically takes care of load balancing and assigns them a single (virtual) service IP (which you can also set manually).

They are a basic concept that is especially useful when working with microservice architectures as they decouple individual services from each other. For example service A accessing service B doesn't know what kind of and how many pods are actually doing the work in B. The actual pods and even their implementation could change completely.

Additionally, services can abstract away other kinds of backends that are not Kubernetes pods. You can for example abstract away an external database cluster behind a service. This way you can for example use a simple local database for your development environment and a professionally managed database cluster in production without having to change the way that your other services talk to that database service.

You can use the same feature if some of your workloads are running outside of your Kubernetes cluster, i.e. either on another Kubernetes cluster (or namespace) or even completely outside of Kubernetes. Latter is especially interesting if you are just starting to migrate workloads.

You can discover and talk to services in your cluster either via environment variables or via DNS. Latter is a cluster addon, which comes OOTB in most Kubernetes installs (including Giant Swarm).

In case you want to use another type of service discovery and don't want the load balancing and single service IP provided by the service, there's an option to create a so-called "headless" services. You can use this if you are already using a service discovery or want to reduce coupling to the Kubernetes system.

Ingress

An Ingress is a collection of rules that allow inbound connections from outside the cluster to reach the cluster services. It can be configured to give services externally-reachable URLs, load balance traffic, terminate SSL, offer name based virtual hosting etc.

An Ingress controller is responsible for fulfilling the Ingress, usually with a loadbalancer, though it may also configure an edge router or additional frontends to help handle the traffic in an HA manner.

There are several use cases and features for Ingress:

- Single Service Ingress
- Simple fanout
    - Route different paths to different services
- Name based virtual hosting
    - Route different host names to different services
- TLS/SNI

Further Reading

- Getting Started With A Local Kubernetes Environment
- Minikube
- Understanding Basic Kubernetes Concepts I - An Introduction To Pods, Labels, and Replicas
- Understanding Basic Kubernetes Concepts II - Using Deployments To Manage Your Services Declaratively
- Understanding Basic Kubernetes Concepts III - Services Give You Abstraction
- Pods Reference Documentation
- Labels & Selectors Reference Documentation
- Services Reference Documentation
- Deployments Reference Documentation
- Services Reference Documentation
- Ingress Reference Documentation