Running Containers and Exposing Services II

Basic Concepts in Kubernetes II

Namespaces

The Kubernetes documentation call Namespaces "virtual clusters" and in a way namespaces do separate the cluster into separate environments. However, namespaces do not bring isolation of resources by default. They only make sure that resources in a certain namespaces by default talk to resources in the same namespace.

They are a great way of keeping your clusters orderly as you can group your deployments (and connected resources) in separate environments. For example you could have a `monitoring` namespace, which holds all your monitoring tools.

You could also create a namespace for a project you're currently working on. A namespace also makes it very easy to clean up after yourself. For example, you could create a `myapp-test` namespace to test out some things you are working on and once you are done, you can delete everything with a single

`kubectl delete namespace myapp-test`

Another use case for namespaces is separating environments like dev, testing, QA, pre-prod, and prod (more on that later).

Node Selectors

In some cases you want to schedule pods to specific nodes. This might be because you want specific pods to run on the same host and be able to communicate faster or because you have a policy of e.g. running frontends on separate machines than backends. You could also have labeled nodes based on the technologies they support, for example if you have nodes with SSDs and others with HDDs.

In these cases you can use Node Selectors to let your pods be scheduled to a specific set of nodes.

For this you need to first assign labels to your pods. You can do this on a per node basis or give a group of pods the same label.

You then use the `nodeSelector` field in the Pod specification to select the nodes you want the pod(s) to be scheduled to.

Config Maps

Many applications require configuration via some combination of config files, command line arguments, and environment variables. These configuration artifacts should be decoupled from image content in order to keep containerized applications portable.

Config Maps hold this configuration information and enable you to inject it into pods.

Not only is configuration independent of your containers and can be changed based on environment, it can also be updated independently. However, depending on how you use the configs provided you might need to reload the configs, e.g. with an API call to prometheus to reload.

Secrets

Secrets are very similar to Config Maps. However, they are meant for sensitive information like passwords, keys, tokens, etc.

Kubernetes creates and uses some secrets automatically (e.g. for accessing the API from a pod), but you can also create your own easily.

Using secrets is quite straightforward. You reference them in a pod and can then use them either as files from volumes or as environment variables in your pod. Keep in mind that each container in your pod that needs to access the secret needs to request it explicitly. There's no implicit sharing of secrets inside the pod.

There's also a special type of secret called imagePullSecrets. Using these you can pass a Docker (or other) container image registry login to the Kubelet, so it can pull a private image for your pod.

Secrets are kept in a tmpfs and only on nodes that run pods that use those secrets. The tmpfs keeps secrets from coming to rest on the node. However, they are (currently) transmitted to and from the API server in plain text, thus, be sure to have SSL/TLS protected connections between user and API server, but also between API server and Kubelets (Giant Swarm clusters do come with both enabled by default).

Daemon Sets

A daemon set ensures that an instance of a specific pod is running on all (or a selection of) nodes in a cluster. It creates pods on each added node and garbage collects pods when nodes are removed from the cluster.

As the name suggests you can use daemon sets for running daemons (and other tools) that need to run on all nodes of a cluster. These can be things like cluster storage daemons (e.g. Quobyte, glusterd, ceph, etc.), log collectors (e.g. fluentd or logstash), or monitoring daemons (e.g. Prometheus Node Exporter, collectd, New Relic agent, etc.)

The simplest use case is deploying a daemon to all nodes. However, you might want to split that up to multiple daemon sets for example if you have a cluster with nodes of varying hardware, which might need

adaptation in the memory and/or cpu requests you might include for the daemon.

In other cases you might want different logging, monitoring, or storage solutions on different nodes of your cluster. For these cases where you want to deploy the daemons only to a specific set of nodes instead of all, you can use a node selector to specify a subset of nodes for the daemon set. Note that for this to work you need to have labeled your nodes accordingly.

There are four ways to communicate with your daemons:

- Push: The pods are configured to push data to a service, so they do not have clients that need to find them.
- NodeIP and known port: The pods use a hostPort and clients can access them via this port on each NodeIP (in the range of nodes they are deployed to).
- DNS: The pods can be reached through a headless service by either using the endpoints resource or getting multiple A records from DNS.
- Service: The pods are reachable through a standard service. Clients can access a daemon on a random node using that service. Note that this option doesn't offer a way to reach a specific node.

Currently, you cannot update a daemon set. The only way to semi-automatically update the pods is to delete the daemon set with the --cascade=false option, so that the pods will be left on the nodes. Then you create a new daemon set with the same pod selector, but the updated template. The new daemon set will recognize the old pods, but not update them automatically. You will need to force the creation of pods with the new template, by manually deleting the old pods from the nodes.

Jobs

Unlike the typical pod that you use for long-running processes, jobs let you manage pods that are supposed to terminate and not be restarted. A job creates one or more pods and ensures a specified number of them terminate with success.

You can use jobs for the typical batch-job (e.g. a backup of a database), but also for workers that need to work off a certain queue (e.g. image or video converters).

There are three kinds of jobs:

- Non-parallel jobs
- Parallel jobs with a fixed completion count
- Parallel jobs with a work queue

For non-parallel jobs usually only one pod gets started and the job is considered complete once the pod terminates successfully. If the pod fails another one gets started in its place.

For parallel jobs with a fixed completion count the job is complete when there is one successful pod for each value between 1 and the number of completions specified.

For parallel jobs with a work queue, you need to take care that no pod terminates with success unless the work queue is empty. That is even if the worker did its job it should only terminate successfully if it knows that all its peers are also done. Once a pod exits with success, then all other pods should also be exited or in the process of exiting.

For parallel jobs you can define the requested parallelism. By default it is set to 1 (only a single pod at any time). If parallelism is set to 0, the job is basically paused until it is increased.

Keep in mind that parallel jobs are not designed to support use cases that need closely-communicating parallel processes like for example in scientific computations, but rather for working off a specific amount of work that can be parallelized.

Scheduled Jobs

A Scheduled Job manages time based Jobs, namely:

- Once at a specified point in time
- Repeatedly at a specified point in time

One ScheduledJob object is like one line of a crontab (cron table) file. It runs a job periodically on a given schedule, written in Cron format.

Further Reading

- Understanding Basic Kubernetes Concepts IV - Secrets and ConfigMaps
- Understanding Basic Kubernetes Concepts V - Daemon Sets and Jobs
- Namespaces Reference Documentation
- Node Selectors Reference Documentation
- ConfigMaps Reference Documentation
- Secrets Reference Documentation
- Daemon Sets Reference Documentation
- Jobs Reference Documentation
- Scheduled Jobs Reference Documentation