Application Configuration And Context

Summary

In the implementation phase, a good development practice is to separate the configuration from the source code. This increased the portability of an application as it can cover multiple customer use cases. Kubernetes has 2 resources to pass data to an application: Configmaps and Secrets.

ConfigMaps

ConfigMaps are objects that store non-confidential data in key-value pairs. A Configmap can be consumed by a pod as an environmental variable, configuration files through a volume mount, or as command-line arguments to the container.

To create a Configmap use the `kubectl create configmap` command, with the following syntax:

```
# create a Configmap
# NAME - required; set the name of the configmap resource
# FLAGS - optional; define  extra configuration parameters for the configmap
kubectl create configmap NAME [FLAGS]

# Some of the widely used FLAGS are:
--from-file - set path to file with key-value pairs
--from-literal - set key-value pair from command-line
```

For example, to create a Configmap to store the background color for a front-end application, the following command can be used:

```
# create a Configmap to store the color value
kubectl create configmap test-cm --from-literal=color=yellow
```

Secrets

Secrets are used to store and distribute sensitive data to the pods, such as passwords or tokens. Pods can consume secrets as environment variables or as files from the volume mounts to the pod. It is noteworthy, that Kubernetes will encode the secret values using base64.

To create a Secret use the `kubectl create secret generic` command, with the following syntax:

```
# create a Secret
# NAME - required; set the name of the secret resource
# FLAGS - optional; define  extra configuration parameters for the secret
kubectl create secret generic NAME [FLAGS]

# Some of the widely used FLAGS are:
--from-file - set path to file with the sensitive key-value pairs
--from-literal - set key-value pair from command-line
```

For example, to create a Secret to store the secret background color for a front-end application, the following command can be used:

```
# create a Secret to store the secret color value
kubectl create secret generic test-secret --from-literal=color=blue
```

Namespaces

A Kubernetes cluster is used to host hundreds of applications, and it is required to have separate execution environments across teams and business verticals. This functionality is provisioned by the **Namespace** resources. A Namespace provides a logical separation between multiple applications and associated resources. In a nutshell, it provides the **application context**, defining the *environment* for a group of Kubernetes resources that relate to a project, such as the amount of CPU, memory, and access. For example, a `project-green` namespace includes any resources used to deploy the Green Project. These resources construct the application context and can be managed collectively to ensure a successful deployment of the project.

Each team or business vertical is allocated a separate Namespace, with the desired amount of CPU, memory, and access. This ensures that the application is managed by the owner team and has enough resources to execute successfully. This also eliminates the "noisy neighbor" use case, where a team can consume all the available resources in the cluster if no Namespace boundaries are set.

To create a Namespace we can use the `kubectl create namespace` command, with the following syntax:

```
# create a Namespace
# NAME - required; set the name of the Namespace
kubectl create ns NAME
```

For example, to create a `test-udacity` Namespace, the following command can be used:

```
# create a `test-udacity` Namespace
kubectl create ns test-udacity

# get all the pods in the `test-udacity` Namespace
kubectl get po -n test-udacity
```

Demo - Application Configuration

Summary

This demo showcases how Configmap and Secret resources can be created using literal values from the command line.

Demo - Application Context

Summary

This demo is a step-by-step guide on how to create a Namespace resource and retrieve resources from a specific Namespace.

New terms

- **Configmap** - a resource to store non-confidential data in key-value pairs.
- **Secret** - a resource to store confidential data in key-value pairs. These are base64 encoded.
- **Namespace** - a logical separation between multiple applications and associated resources.

Further reading

Explore Kubernetes resources to pass configuration to pods:

- Kubernetes Configmap
- Kubernetes Secrets
- Kuebrnetes Namespaces