DevOps and Security Lab 4 - Kubernetes Basics

The lab goal is to get familiar with the main Kubernetes manifests, then try them on practice and deploy application on Kubernetes cluster as well.

> Individual lab. **Choose this lab flavor if you don't work with Kubernetes before and have to learn it from the beginning.**
>
> While working on some tasks, keep in mind that you might wipe your created objects after the task completion before to go the next task in order do not mess some logically conflicted objects.
>
> We are going to work within the local cluster (minikube, k3s, MicroK8s...) but nothing prevents you from deploying a cluster on a real cloud like AWS EKS or On-premise bare-metal solution.

Task 1 - Preparation

1.1. Choose an application that has such characteristics as (*points with "star" are not required but recommended*):

- any non-confidential data like in JSON, XML or TOML format
- application configuration values (or environment variables)
- any secrets (like authorization credentials)
- availability from outside (on Internet) *
- has any health checks or/and status endpoints *

It could be any yours application or open-source project that meets the requirements. You are even able to work with different applications in different lab tasks but try to avoid it. However, it's **much better** to work with the single project but it's acceptable to involve only the required application functionality regarding the particular task requirements. Put the git link to your chosen project into report and provide a brief project description for what it is and how it works.

1.2. Get familiar with Kubernetes (k8s) and concepts.

1.3. Install and set up the necessary tools:

- kubectl
- minikube or its alternative

1.4. Get access to Kubernetes Dashboard.

Task 2 - k8s Nodes

> We are going to start to learn Kubernetes from kubectl command line that is dedicated to work with k8s cluster.

1. After installing all required lab tools and starting `minikube` cluster, use `kubectl` commands to `get` and `describe` your cluster nodes.
2. Get the more detailed information about the particular node (it's fine if you have one node in the cluster).

3. Get the OS and CPU information.
4. Put the results into report.

Task 3 - k8s Pod

> `Pod` is one of the simplest and basic k8s unit. It's like an abstraction over Docker containers. Inside pods containers run your application.

1. Figure out the necessary `Pod` spec fields.
2. Write a `Pod` spec for your chosen application, deploy the application and run a pod.
3. With `kubectl`, get the pods, pod logs, describe pod, go into pod shell.
4. Make sure that your app is working correctly inside Pod.
5. Put the results into report.

Task 4 - k8s Service

> We use k8s `Services` to make an application accessible from outside the Kubernetes virtual network, provide an compatible IP address and link to DNS name to pod, to route internal/external traffic within pods.

1. Figure out the necessary `Service` spec fields.
2. Write a `Service` spec for your pod(s) and deploy the `Service`.
3. With `kubectl`, get the `Services` and `describe` them.
4. Make sure that pods can communicate between each other using DNS names, check pods addresses.
5. Delete any Pod, recreate it and check addresses again. Make sure that traffic is routed to the new Pod correctly.
6. Learn about `Loadbalancer` and `NodePort`.
7. Deploy an external `Service` to access your application from outside, e.g., from your local host.
8. Put the results into report.

Bonus: learn how `labels` are used in Kubernetes. Create and provide a PoC of scenarious where no any pods math the label spec and where at least one or multiple pods math the label spec.

Task 5 - k8s Deployment

> `Deployment` is Kubernetes manifest to manage Pods automatically by Controller. It helps to release, scale and upgrade Pods.

1. Figure out the necessary `Deployment` spec fields.
2. Make sure that you wiped previous `Pod` manifests. Write a `Deployment` spec for your pod(s) and deploy the application.
3. With `kubectl`, get the `Deployments` and `describe` them.
4. Update your `Deployment` manifest to scale your application to three replicas.
5. Access pod shell and logs using `Deployment` labels.
6. Make any application configuration change in your `Deployment` yaml and try to update the application. Monitor what are happened with pods (`--watch`.
7. Rollback to previous application version using `Deployment`.
8. Set up `requests` and `limits` for CPU and Memory for your application pods. Provide a PoC that it works properly. Explain and show what is happened when app reaches the CPU usage limit? And memory limit?

9. Put the results into report.

Bonus: based on the last exercise with application update, learn and show a PoC how `ReplicaSets` are involved in updating process through `Deployments`.

Task 6 - k8s Secrets

> `Secret` manifest is a quite similar to `configMap`. However, we use `Secret` to work with confidential application data. Kubernetes encode secrets in Base64 format.

1. Figure out the necessary `Secret` spec fields.
2. Create and apply a new `Secret` manifest. For example, it could be login and password to login to your app or something else.
3. With `kubectl`, get and describe your secret(s).
4. Decode your secret(s).
5. Update your `Deployment` to reference to your secret as environment variable.
6. Make sure that you are able to see your secret inside pod.
7. Put the results into report.

Bonus: create a secret from json file. Hint: use configMap logic for that. Repeat steps 3 - 6.

Task 7 - k8s configMap

> `ConfigMap` is Kubernetes manifest to store application configuration setting in two ways: key-value pairs as environment variables and text (usually JSON) data as dedicated file into container filesystem. We usually use configMap to store non sensitive data as plain text.

1. Figure out the necessary `configMap` spec fields.
2. Modify your `Deployment` manifest to set up some app configuration via environment variables.
3. Create a new `configMap` manifest. In data spec, put some app configuration as key-value pair (it could be the same as in previous exercise). In the Deployment.Pod spec add the connection to key-value pair from `configMap` yaml file.
4. Create a new file like `config.json` file and put some json data into.
5. Create a new one `configMap` manifest. Connect `configMap` yaml file with `config.json` file to read the data from it.
6. Update your `Deployment` to add `Volumes` and `VolumeMounts`.
7. With `kubectl`, check the `configMap` details. Make sure that you see the data as plain text.
8. Check the filesystem inside app container to show the loaded file data on the specified path.
9. Put the results into report.

Task 8 - k8s Namespace

> `Namespace` Kubernetes Manifest is designed for different projects and deployment environments isolation. With `Namespaces` we can separate App 1 deployment from App 2 deployment, manage (and isolate) cluster resources for them, define users list to have access either to App 1 or to App 2 deployment. Using `Namespaces`, we also can define a different environments like DEV, TEST, STAGE. In that way, `Namespaces` is a required feature for a real Kubernetes production clusters.

1. Figure out the necessary `Namespace` spec fields.
2. Create a two different Namespaces in your k8s cluster.

3. Using `kubectl`, get and describe your Namespaces.
4. Deploy two different applications in two different Namespaces with kubectl. By the way, it's acceptable even just to deploy the same objects (same previous app) in the different Namespaces but with different resources names.
5. With `kubectl`, get and describe pods from different Namespaces witn `-n` flag.
6. Can you see and can you connect to the resources from different Namespaces?
7. Put the results into report.

Bonus: install and show a practical example of https://github.com/ahmetb/kubectx ***kubens*** *usage.*

Bonus - Rancher

On real complicated production Kubernetes clusters, it's very important to have a centralized GUI based system to easily access, control and monitor k8s deployments. Install any k8s container management platform and show several actions related to cluster control in practice.

Rancher is one of the most popular solution.