Version - **2025.01**

Last update : 2025/01/17 16:39

DOE304 - Working with Pods and Containers

Module content

LAB #1 - Application Configuration

1.1 - Overview

Application Configuration is the process of passing dynamic values to applications at runtime.

There are two ways of storing information in K8s:

- ConfigMaps,
- Secrets.

Data stored in ConfigMaps and Secrets can be passed to containers using :

- Environment variables,

- Configuration volumes.

1.2 - Creating a ConfigMap

To begin, create the file **myconfigmap.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi myconfigmap.yaml
root@kubemaster:~# cat myconfigmap.yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: my-configmap
data:
  key1: Hello, world!
  key2: |
    Test
    multiple lines
    more lines
```

Important: Note that the data is stored in **Key-values**. The first data item in the **data** section is **key1: Hello, world!**, while the second, **key2**, is in multiple lines.

Now create the ConfigMap :

```
root@kubemaster:~# kubectl create -f myconfigmap.yaml
```

```
configmap/my-configmap created
```

To view the contents of the ConfigMap, use the **kubectl describe** command:

```
root@kubemaster:~# kubectl describe configmap my-configmap
Name: my-configmap
Namespace: default
Labels: <none>
Annotations: <none>

Data
====
key1:
----
Hello, world!
key2:
----
Test
multiple lines
more lines


BinaryData
====

Events: <none>
```

1.3 - Creating a Secret

Now create the **mysecret.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi mysecret.yaml
root@kubemaster:~# cat mysecret.yaml
apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  secretkey1:
  secretkey2:
```

Important: Note that the secret keys have not yet been defined.

Now encrypt both keys using **base64** :

```
root@kubemaster:~# echo -n 'secret' | base64
c2VjcmV0

root@kubemaster:~# echo -n 'anothersecret' | base64
YW5vdGhlcnNlY3JldA==
```

Copy and paste the base64 strings into the mysecret.yaml file:

```
root@kubemaster:~# vi mysecret.yaml
root@kubemaster:~# cat mysecret.yaml
apiVersion: v1
kind: Secret
```

```
metadata:
  name: my-secret
type: Opaque
data:
  secretkey1: c2VjcmV0
  secretkey2: YW5vdGhlcnNlY3JldA==
```

Important: Replace the strings with the ones that YOU have created.

Now create the Secret :

```
root@kubemaster:~# kubectl create -f mysecret.yaml
secret/my-secret created
```

1.4 - Using ConfigMaps and Secret

Using Environment variables

Create the file **envpod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi envpod.yaml
root@kubemaster:~# cat envpod.yaml
apiVersion: v1
kind: Pod
```

```
metadata:
  name: envpod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'echo "configmap: $CONFIGMAPVAR secret: $SECRETVAR"']
    env:
    - name: CONFIGMAPVAR
      valueFrom:
        configMapKeyRef:
          name: my-configmap
          key: key1
    - name: SECRETVAR
      valueFrom:
        secretKeyRef:
          name: my-secret
          key: secretkey1
```

Important: Note that the **$CONFIGMAPVAR** will contain the value of **key1** from the **ConfigMap** and that the **$SECRETVAR** will contain the value of **secretkey1** from the **Secret**.

Now create the pod :

```
root@kubemaster:~# kubectl create -f envpod.yaml
pod/envpod created
```

Now check the pod logs:

```
root@kubemaster:~# kubectl logs envpod
```

```
configmap: Hello, world! secret: secret
```

Important: Note that the container in the pod can see the values of both variables.

Using Configuration Volumes

Create the file **volumepod.yaml** :

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi volumepod.yaml
root@kubemaster:~# cat volumepod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: volumepod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    volumeMounts:
    - name: configmap-volume
      mountPath: /etc/config/configmap
    - name: secret-volume
      mountPath: /etc/config/secret
  volumes:
```

```
  - name: configmap-volume
    configMap:
      name: my-configmap
  - name: secret-volume
    secret:
      secretName: my-secret
```

Now create the pod:

```
root@kubemaster:~# kubectl create -f volumepod.yaml
pod/volumepod created
```

Use the **kubectl exec** command to view the config data files in the container:

```
root@kubemaster:~# kubectl exec volumepod -- ls /etc/config/configmap
key1
key2

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/configmap/key1
Hello, world!root@kubemaster:~# [Enter]

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/configmap/key2
Test
multiple lines
more lines

root@kubemaster:~# kubectl exec volumepod -- ls /etc/config/secret
secretkey1
secretkey2

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/secret/secretkey1
secretroot@kubemaster:~# [Enter]

root@kubemaster:~# kubectl exec volumepod -- cat /etc/config/secret/secretkey2
```

```
anothersecretroot@kubemaster:~# [Enter]

root@kubemaster:~#
```

Lastly, delete the **envpod** and **volumepod** pods:

```
root@kubemaster:~# kubectl delete pod envpod volumepod
pod “envpod” deleted
pod “volumepod” deleted
```

LAB #2 - Container Resource Management

2.1 - Overview

Two important aspects of container resource management are :

- **Resource Requests**,
  - A Resource Request is used to define resources such as CPU and memory at the time of **scheduling**. In other words, if the Resource Request is for 5GB, the pod scheduler will look for a node with 5GB of available RAM. A Resource Request is not a limit, as the pod can use more or less memory.
- **Resource Limits**,
  - A Resource Limit allows you to set limits on resources such as CPU and memory. Different Container Runtimes react in different ways to a Resource Limit. For example, some will stop the container process if the limit is exceeded. In the case of Docker, if the CPU limit is exceeded, Docker will limit CPU usage. On the other hand, if the memory limit is exceeded, Docker will kill the container process.

For both types, memory requests and limits are generally expressed in **Mi**, while CPU requests and limits are expressed in 1/1000 of a processor. For example, 250m represents 250/1000 of a CPU or 1/4.

2.2 - Resource Requests

Create the file **bigrequestpod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi bigrequestpod.yaml
root@kubemaster:~# cat bigrequestpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: bigrequestpod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    resources:
      requests:
        cpu: "10000m"
        memory: "128Mi"
```

Create the pod:

```
root@kubemaster:~# kubectl create -f bigrequestpod.yaml
pod/bigrequestpod created
```

Now check the status of the created pod:

```
root@kubemaster:~# kubectl get pod bigrequestpod
NAME            READY   STATUS    RESTARTS   AGE
bigrequestpod   0/1     Pending   0          92s
```

Important: Note that the pod's status is **pending**. The pod will remain pending because neither **kubenode1** nor **kubenode2** are able to meet the **10000m** resource request.

2.3 - Resource Limits

Create the file **resourcepod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi resourcepod.yaml
root@kubemaster:~# cat resourcepod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: resourcepod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    resources:
      requests:
        cpu: "250m"
        memory: "128Mi"
      limits:
        cpu: "500m"
```

```
        memory: "256Mi"
```

Create the pod:

```
root@kubemaster:~# kubectl create -f resourcepod.yaml
pod/resourcepod created
```

Check pod status:

```
root@kubemaster:~# kubectl get pods
NAME                              READY   STATUS              RESTARTS       AGE
bigrequestpod                     0/1     Pending             0              20m
my-deployment-67b5d4bf57-6wcrq    1/1     Running             0              22h
myapp-deployment-689f9d59-c25f9   1/1     Running             0              7d
myapp-deployment-689f9d59-nn9sw   1/1     Running             0              7d
myapp-deployment-689f9d59-rnc4r   1/1     Running             0              7d
resourcepod                       1/1     Running             0              5m49s
```

Important: Note that the status of the **bigrequestpod** pod is **still pending**.

LAB #3 - Container Supervision

3.1 - Overview

Container supervision involves monitoring the health of containers to ensure robust applications and solutions by restarting broken containers. To accomplish this task, K8s uses *probes*.

There are several types of probe:

- **Liveness Probes**,
  - By default, K8s considers a container to be out of service only when it stops,
  - Liveness probes allow more sophisticated configuration of this mechanism.
- **Startup Probes**,
  - Similar to Liveness Probes, Startup Probes only intervene at container startup and stop when the application has started.
- **Readiness Probes**,
  - Similar to Startup Probes in that they only intervene at pod startup, Readiness Probes are responsible for blocking traffic to pods until all containers in the pod have passed met the Readiness Probes criteria.

3.2 - Liveness Probes

The exec Probe

Create the file **livenesspod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi livenesspod.yaml
root@kubemaster:~# cat livenesspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: livenesspod
spec:
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'while true; do sleep 3600; done']
    livenessProbe:
```

```
      exec:
        command: ["echo", "Hello, world!"]
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important: In the file above, if the command **echo “Hello, World!”** returns a exit code of 0, the container will be considered healthy. The Liveness Probe will start 5 seconds after the container is started, thanks to the **initialDelaySeconds** directive. The probe will then run every 5 seconds using the **periodSeconds** directive.

Create the pod:

```
root@kubemaster:~# kubectl create -f livenesspod.yaml
pod/livenesspod created
```

Check the pod status:

```
root@kubemaster:~# kubectl get pod livenesspod
NAME           READY   STATUS    RESTARTS   AGE
livenesspod    1/1     Running   0          90s
```

Important: Note that the pod is healthy and in a **running** status.

The httpGet Probe

Create the file **livenesspodhttp.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi livenesspodhttp.yaml
root@kubemaster:~# cat livenesspodhttp.yaml
apiVersion: v1
kind: Pod
metadata:
  name: livenesspodhttp
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
    livenessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important: In the above file, if the **GET /** command executes without error, the container will be considered healthy. The Liveness Probe will start 5 seconds after the container is started, thanks to the **initialDelaySeconds** directive. The probe will then run every 5 seconds using the **periodSeconds** directive.

Create the pod:

```
root@kubemaster:~# kubectl create -f livenesspodhttp.yaml
pod/livenesspodhttp created
```

Check the pod status:

```
root@kubemaster:~# kubectl get pod livenesspodhttp
NAME              READY   STATUS    RESTARTS   AGE
livenesspodhttp   1/1     Running   0          52s
```

Important: Note that the pod is healthy and in the **running** status.

3.3 - Startup Probes

Create the **startuppod.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi startuppod.yaml
root@kubemaster:~# cat startuppod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: startuppod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
    startupProbe:
      httpGet:
        path: /
```

```
      port: 80
    failureThreshold: 30
    periodSeconds: 10
```

Important: In the above file, the Startup Probe will wait a maximum of 30 seconds for the application to start, thanks to the **failureThreshold** directive. The probe will run every 10 seconds, thanks to the **periodSeconds** directive.

Create the pod:

```
root@kubemaster:~# kubectl create -f startuppod.yaml
pod/startuppod created
```

Check the pod status:

```
root@kubemaster:~# kubectl get pod startuppod
NAME           READY   STATUS    RESTARTS   AGE
livenesspod    1/1     Running   0          90s
```

Important: Note that the pod is healthy and in the **running** status.

3.4 - Readiness Probes

Create the **readinesspod.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi readinesspod.yaml
root@kubemaster:~# cat readinesspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: readinesspod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
    readinessProbe:
      httpGet:
        path: /
        port: 80
      initialDelaySeconds: 5
      periodSeconds: 5
```

Important: In the above file, if the **GET /** command executes without error, the container will be considered in a READY state. The Readiness Probe will start 5 seconds after container startup, thanks to the **initialDelaySeconds** directive. The probe will then run every 5 seconds using the **periodSeconds** directive.

Create the pod and check its status:

```
root@kubemaster:~# kubectl create -f readinesspod.yaml;kubectl get pod readinesspod;sleep 1;kubectl get pod
readinesspod;sleep 3;kubectl get pod readinesspod;sleep 3;kubectl get pod readinesspod;sleep 3;kubectl get pod
readinesspod;sleep 3;kubectl get pod readinesspod
pod/readinesspod created
```

```
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Pending   0          0s
NAME           READY   STATUS              RESTARTS   AGE
readinesspod   0/1     ContainerCreating   0          1s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          4s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          7s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   0/1     Running   0          10s
NAME           READY   STATUS    RESTARTS   AGE
readinesspod   1/1     Running   0          13s
```

Important: Note that the pod has a Running status 4 seconds after startup. On the other hand, the pod only switches to READY after 13 seconds when the Readiness Probe is successful.

LAB #4 - Restart Policy Management

4.1 - Overview

K8s can restart containers in the event of problems. There are three restart policies:

- **Always**,
  - Always is the default policy,
  - Always restarts a container regardless of the exit code when the container is stopped.
- **OnFailure**,
  - OnFailure restarts a container only if it exits with a exit code other than 0, or if a Liveness Probe has reported poor container health. In the opposite case, where the container has completed its task and exits with a exit code of 0, the policy does not restart it.

- **Never**,
  - Never is the opposite of Always. The container is never restarted in the event of a container shutdown, whatever the cause.

4.2 - Always

Create the file **alwayspod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi alwayspod.yaml
root@kubemaster:~# cat alwayspod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: alwayspod
spec:
  restartPolicy: Always
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10']
```

Create the pod and check its status:

```
root@kubemaster:~# kubectl create -f alwayspod.yaml;kubectl get pod alwayspod;sleep 9;kubectl get pod
alwayspod;sleep 9;kubectl get pod alwayspod;sleep 9;kubectl get pod alwayspod
pod/alwayspod created
NAME        READY   STATUS              RESTARTS   AGE
alwayspod   0/1     ContainerCreating   0          0s
NAME        READY   STATUS    RESTARTS   AGE
```

```
alwayspod   1/1     Running   0          9s
NAME        READY   STATUS    RESTARTS     AGE
alwayspod   1/1     Running   1 (6s ago)   19s
NAME        READY   STATUS      RESTARTS      AGE
alwayspod   0/1     Completed   1 (15s ago)   28s
```

Important: Note that the pod has been restarted.

4.3 - OnFailure

Create the **onfailure.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi onfailure.yaml
root@kubemaster:~# cat onfailure.yaml
apiVersion: v1
kind: Pod
metadata:
  name: onfailure
spec:
  restartPolicy: OnFailure
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10']
```

Create the pod and check its status:

```
root@kubemaster:~# kubectl create -f onfailure.yaml;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure
pod/onfailure created
NAME        READY   STATUS    RESTARTS   AGE
onfailure   0/1     Pending   0          0s
NAME        READY   STATUS    RESTARTS   AGE
onfailure   1/1     Running   0          9s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          19s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          28s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          37s
NAME        READY   STATUS      RESTARTS   AGE
onfailure   0/1     Completed   0          46s
```

Important: Note that the pod has not been restarted.

Now delete the onfailure pod:

```
root@kubemaster:~# kubectl delete pod onfailure
pod “onfailure” deleted
```

Then modify the **onfailure.yaml** file by adding the string **this is a bad command** :

```
root@kubemaster:~# vi onfailure.yaml
root@kubemaster:~# cat onfailure.yaml
apiVersion: v1
```

```
kind: Pod
metadata:
  name: onfailure
spec:
  restartPolicy: OnFailure
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10;this is a bad command']
```

Create the pod and check its status:

```
root@kubemaster:~# kubectl create -f onfailure.yaml;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod onfailure;sleep 9;kubectl get pod
onfailure;sleep 9;kubectl get pod onfailure
pod/onfailure created
NAME        READY   STATUS    RESTARTS   AGE
onfailure   0/1     Pending   0          0s
NAME        READY   STATUS    RESTARTS   AGE
onfailure   1/1     Running   0          9s
NAME        READY   STATUS    RESTARTS     AGE
onfailure   1/1     Running   1 (5s ago)   18s
NAME        READY   STATUS   RESTARTS      AGE
onfailure   0/1     Error    1 (14s ago)   27s
NAME        READY   STATUS   RESTARTS      AGE
onfailure   0/1     Error    1 (23s ago)   36s
NAME        READY   STATUS    RESTARTS      AGE
onfailure   1/1     Running   2 (21s ago)   46s
```

Important: Note that the pod has been restarted due to the error.

4.4 - Never

Create the file **never.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi never.yaml
root@kubemaster:~# cat never.yaml
apiVersion: v1
kind: Pod
metadata:
  name: never
spec:
  restartPolicy: Never
  containers:
  - name: busybox
    image: busybox
    command: ['sh', '-c', 'sleep 10;this is a bad command']
```

Create the pod and check its status:

```
root@kubemaster:~# kubectl create -f never.yaml;kubectl get pod never;sleep 9;kubectl get pod never;sleep
9;kubectl get pod never;sleep 9;kubectl get pod never;sleep 9;kubectl get pod never;sleep 9;kubectl get pod never
pod/never created
NAME    READY   STATUS              RESTARTS   AGE
never   0/1     ContainerCreating   0          0s
NAME    READY   STATUS    RESTARTS   AGE
never   1/1     Running   0          9s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          18s
```

```
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          27s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          36s
NAME    READY   STATUS   RESTARTS   AGE
never   0/1     Error    0          45s
```

Important: Note that the pod has not been restarted.

LAB #5 - Creating Multi-container Pods

5.1 - Overview

It's always best to put only one container in a pod. The exception to this rule is when two or more pods need to interact in order to fulfill their respective roles. The other containers are called **sidecars** or **helpers**. The interaction is called **Cross-Container Interaction**.

This interaction takes the form of sharing :

- the same network space,
    - containers can communicate on all ports, even if the ports are not exposed to the cluster,
- the same storage space,
    - containers can share the same volumes.

5.2 - Implementation

Start by creating the file **multicontainerpod.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi multicontainerpod.yaml
root@kubemaster:~# cat multicontainerpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: multicontainerpod
spec:
  containers:
  - name: nginx
    image: nginx
  - name: redis
    image: redis
  - name: couchbase
    image: couchbase
```

Important: Note that the file will create three containers - **nginx**, **redis** and **couchbase**.

Next, create the pod:

```
root@kubemaster:~# kubectl create -f multicontainerpod.yaml
pod/multicontainerpod created
```

Check the pod status:

```
root@kubemaster:~# kubectl get pod multicontainerpod
NAME                READY   STATUS              RESTARTS   AGE
```

```
multicontainerpod   0/3     ContainerCreating   0          65s
```

Important: Note that there are currently 0 of 3 pods in a READY state.

Wait a few minutes and check the pod status again:

```
root@kubemaster:~# kubectl get pod multicontainerpod
NAME                READY   STATUS    RESTARTS   AGE
multicontainerpod   3/3     Running   0          16m
```

Important: Note that there are currently 3 of 3 pods in a READY state.

Now create the **helper.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi helper.yaml
root@kubemaster:~# cat helper.yaml
apiVersion: v1
kind: Pod
metadata:
  name: helperpod
spec:
  containers:
  - name: busybox1
```

```
    image: busybox
    command: ['sh', '-c', 'while true; do echo logs data > /output/output.log; sleep 5; done']
    volumeMounts:
      - name: sharedvol
        mountPath: /output
  - name: helper
    image: busybox
    command: ['sh', '-c', 'tail -f /input/output.log']
    volumeMounts:
      - name: sharedvol
        mountPath: /input
  volumes:
  - name: sharedvol
    emptyDir: {}
```

Important: Note that this file will create a pod containing two containers - **busybox1** and **helper**. Each container shares an identical volume called **sharedvol**. In the ***busybox1** container this volume is mounted on **/output** while in the **helper** container, the same volume is mounted on **/input**.

Create the **helper** pod:

```
root@kubemaster:~# kubectl create -f helper.yaml
pod/helperpod created
```

View the **helper** container logs in the **helperpod** pod:

```
root@kubemaster:~# kubectl logs helperpod -c helper
logs data
```

Important: Note that the **busybox1** container has written the **logs data** string to the **/output/output.log** file every 5 seconds by executing the **while true; do echo logs data > /output/output.log; sleep 5; done** command. The **helper** container executes the **tail -f /input/output.log** command. The **helper** container log therefore contains the **logs data** string from the **output.log** file, as this file is shared between the two containers.

LAB #6 - Init containers

6.1 - Overview

An Init Container is a container that runs only once at pod startup. If several Init Containers exist, they run in sequence. An Init container must complete its execution before the next Init container, or the application if the Init container concerned is the last, can run. The purpose of an Init container is to execute code that doesn't need to be in the application's containers in order to make the latter lighter, for example:

- Securely isolate sensitive data such as passwords, to prevent them being compromised if an application container is compromised,
- inject data into a shared volume,
- wait for other K8s resources to be created.

6.2 - Implementation

Start by creating the **initpod.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi initpod.yaml
root@kubemaster:~# cat initpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: initpod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
  initContainers:
  - name: delay
    image: busybox
    command: ['sleep', '30']
```

Important: Note that the **delay** container will delay the creation of the **nginx** container for 30 seconds.

Create the **initpod** pod:

```
root@kubemaster:~# kubectl create -f initpod.yaml
pod/initpod created
```

Check the pod status:

```
root@kubemaster:~# kubectl get pod initpod
NAME      READY   STATUS     RESTARTS   AGE
initpod   0/1     Init:0/1   0          6s
```

Important: Note that the pod's **STATUS** is **Init:0/1**.

Wait at least 30 seconds, then execute the last command again:

```
root@kubemaster:~# kubectl get pod initpod
NAME      READY   STATUS    RESTARTS   AGE
initpod   1/1     Running   0          79s
```

Important: Note that the pod's **STATUS** is **Running**.

LAB #7 - Scheduling

7.1 - Overview

Scheduling is the process of assigning pods to nodes. This process is performed by the **Scheduler**, a component of the **Control Plane**.

The Scheduler makes its decision based on the following criteria:

- the resources available on the nodes, based on **Resource Requests**,
- configurations of **nodeSelectors** using **Node Labels**,
- **nodeName** instructions that force the choice of one node over another.

7.2 - Implementation

Start by viewing the cluster nodes:

```
root@kubemaster:~# kubectl get nodes
NAME                        STATUS   ROLES           AGE   VERSION
kubemaster.ittraining.loc   Ready    control-plane   11d   v1.25.0
kubenode1.ittraining.loc    Ready    <none>          11d   v1.25.0
kubenode2.ittraining.loc    Ready    <none>          11d   v1.25.0
```

nodeSelector

Assign the label **mylabel=thisone** to node **kubenode1.ittraining.loc** :

```
root@kubemaster:~# kubectl label nodes kubenode1.ittraining.loc mylabel=thisone
node/kubenode1.ittraining.loc labeled
```

Now create the file **nodeselector.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi nodeselector.yaml
root@kubemaster:~# cat nodeselector.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodeselector
spec:
  nodeSelector:
    mylabel: "thisone"
  containers:
  - name: nginx
    image: nginx:1.19.1
```

Important: Note the **nodeSelector** entry.

Create the **nodeselector** pod:

```
root@kubemaster:~# kubectl create -f nodeselector.yaml
pod/nodeselector created
```

Note the location of the **nodeselector** pod:

```
root@kubemaster:~# kubectl get pod nodeselector -o wide
NAME           READY   STATUS    RESTARTS   AGE   IP               NODE                        NOMINATED NODE
READINESS GATES
nodeselector   1/1     Running   0          66s   192.168.239.21   kubenode1.ittraining.loc   <none>
<none>
```

Important: Note that the nodeselector pod has been scheduled on the **kubenode1** node.

nodeName

Now create the **nodename.yaml** file:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi nodename.yaml
```

```
root@kubemaster:~# cat nodename.yaml
apiVersion: v1
kind: Pod
metadata:
  name: nodename
spec:
  nodeName: kubenode2.ittraining.loc
  containers:
    - name: nginx
      image: nginx:1.19.1
```

Important: Note that the pod will be scheduled on **kubenode2.ittraining.loc** thanks to the use of **nodeName**.

Create the **nodename** pod:

```
root@kubemaster:~# kubectl create -f nodename.yaml
pod/nodename created
```

Note the location of the **nodename** pod:

```
root@kubemaster:~# kubectl get pod nodename -o wide
NAME       READY   STATUS    RESTARTS   AGE   IP               NODE                        NOMINATED NODE
READINESS GATES
nodename   1/1     Running   0          67s   192.168.150.25   kubenode2.ittraining.loc   <none>           <none>
```

Important: Note that the pod has been scheduled on **kubenode2.ittraining.loc** thanks to the use of **nodeName**.

LAB #8 - DaemonSets

8.1 - Overview

A DaemonSet:

- creates a copy of a pod on all available nodes,
- creates a copy of a pod on any new node added to the cluster,
- respects Node Labels constraints.

8.2 - Implementation

Start by cleaning up the cluster:

```
root@kubemaster:~# kubectl delete --all pods --namespace=default
pod “alwayspod” deleted
pod “bigrequestpod” deleted
pod “helperpod” deleted
pod “initpod” deleted
pod “liveness-pod” deleted
pod “livenesspodhttp” deleted
pod “multicontainerpod” deleted
pod “my-deployment-67b5d4bf57-6wcrq” deleted
pod “myapp-deployment-689f9d59-c25f9” deleted
pod “myapp-deployment-689f9d59-nn9sw” deleted
pod “myapp-deployment-689f9d59-rnc4r” deleted
pod “never” deleted
pod “nodename” deleted
pod “nodeselector” deleted
pod “onfailure” deleted
pod “readinesspod” deleted
```

```
pod “resourcepod” deleted
pod “startuppod” deleted

root@kubemaster:~# kubectl delete --all deployments --namespace=default
deployment.apps “my-deployment” deleted
deployment.apps “myapp-deployment” deleted
```

Then create the file **daemonset.yaml**:

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubemaster:~# vi daemonset.yaml
root@kubemaster:~# cat daemonset.yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: mydaemonset
spec:
  selector:
    matchLabels:
      app: mydaemonset
  template:
    metadata:
      labels:
        app: mydaemonset
    spec:
      containers:
      - name: nginx
        image: nginx:1.19.1
```

Create the DaemonSet **mydaemonset** :

```
root@kubemaster:~# kubectl create -f daemonset.yaml
daemonset.apps/mydaemonset created
```

Check the status of the **DaemonSet**:

```
root@kubemaster:~# kubectl get daemonset
NAME          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR   AGE
mydaemonset   2         2         2       2            2           <none>          37s
```

Now you can see that there is a pod on each node:

```
root@kubemaster:~# kubectl get pods -o wide
NAME                READY   STATUS    RESTARTS   AGE   IP               NODE                        NOMINATED NODE
READINESS GATES
mydaemonset-hmdhp   1/1     Running   0          38s   192.168.239.26   kubenode1.ittraining.loc   <none>
<none>
mydaemonset-kmf4z   1/1     Running   0          38s   192.168.150.30   kubenode2.ittraining.loc   <none>
<none>
```

Important: Note that there is no pod on **kubemaster**. This is because the kubemaster has the **no taint** flag set, which prevents pods from being scheduled on it.

LAB #9 - Static Pods

9.1 - Presentation

A Static Pod is :

- a pod that is controlled by the **kubelet** on the relevant node instead of by the K8s API,
  - this type of pod can be created even if there is no Control Plane,
  - if the Control Plane exists, a **Mirror Pod** is created in the Control Plane to represent the static pod, making it easier to check its status. However, the pod cannot be changed or managed from the Control Plane,
- a pod created using a yaml file located in a **specific** path on the node concerned,
  - for a cluster installed with **kubeadm**, the default “specific” path in each **worker** is **/etc/kubernetes/manifests**. Note that it is possible to modify this path.

9.2 - Implementation

Connect to kubenode1 and become the **root** user:

```
root@kubemaster:~# ssh -l trainee 192.168.56.3
trainee@192.168.56.3's password: trainee
Linux kubenode1.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sun Sep 4 13:01:18 2022 from 192.168.56.2
trainee@kubenode1:~$ su -
Password: fenestros
root@kubenode1:~#
```

Create the file **/etc/kubernetes/manifests/mystaticpod.yaml** :

```
root@kubenode1:~# mkdir /etc/kubernetes/manifests
```

To do: Copy the content from **here** **and paste it into your file.**

```
root@kubenode1:~# vi /etc/kubernetes/manifests/mystaticpod.yaml
root@kubenode1:~# cat /etc/kubernetes/manifests/mystaticpod.yaml
apiVersion: v1
kind: Pod
metadata:
  name: mystaticpod
spec:
  containers:
  - name: nginx
    image: nginx:1.19.1
```

Important: Note that kubelet will see that the file has been created and then continue with the pod creation.

Restart the **kubelet** service to start the static pod **immediately** without waiting:

```
root@kubenode1:~# systemctl restart kubelet
```

Return to the **kubemaster** and note the presence of a mirrored pod:

```
root@kubenode1:~# exit
logout
trainee@kubenode1:~$ exit
logout
Connection to 192.168.56.3 closed.

root@kubemaster:~# kubectl get pods
```

```
NAME                                READY   STATUS    RESTARTS   AGE
mydaemonset-hmdhp                   1/1     Running   0          32m
mydaemonset-kmf4z                   1/1     Running   0          32m
mystaticpod-kubenode1.ittraining.loc   1/1     Running   0          3m40s
```

Now delete the static pod :

```
root@kubemaster:~# kubectl delete pod mystaticpod-kubenode1.ittraining.loc
pod “mystaticpod-kubenode1.ittraining.loc” deleted
```

Important: Note that the deletion seems to have been successful.

View the running pods:

```
root@kubemaster:~# kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
mydaemonset-hmdhp                   1/1     Running   0          45m
mydaemonset-kmf4z                   1/1     Running   0          45m
mystaticpod-kubenode1.ittraining.loc   1/1     Running   0          19s
```

Important: Note that the pod **mystaticpod-kubenode1.ittraining.loc** has returned. In fact, the previous deletion only deleted the mirror, which was then regenerated.

To delete the static pod, connect to **kubenode1** :

```
root@kubemaster:~# ssh -l trainee kubenode1
trainee@kubenode1's password: trainee
Linux kubenode1.ittraining.loc 4.9.0-19-amd64 #1 SMP Debian 4.9.320-2 (2022-06-30) x86_64
```

```
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Thu Sep 15 17:51:03 2022 from 192.168.56.2

trainee@kubenode1:~$ su -
Password: fenestros

root@kubenode1:~# rm -f /etc/kubernetes/manifests/mystaticpod.yaml

root@kubenode1:~# systemctl restart kubelet

root@kubenode1:~# exit
logout

trainee@kubenode1:~$ exit
logout
Connection to kubenode1 closed.

root@kubemaster:~#
```
