1- how many configmaps exist

```
controlplane $ kubectl get configmaps --all-namespaces
NAMESPACE           NAME                                                 DATA   AGE
default             kube-root-ca.crt                                     1      42h
kube-node-lease     kube-root-ca.crt                                     1      42h
kube-public         cluster-info                                         2      42h
kube-public         kube-root-ca.crt                                     1      42h
kube-system         canal-config                                         6      42h
kube-system         coredns                                              1      42h
kube-system         extension-apiserver-authentication                   6      42h
kube-system         kube-apiserver-legacy-service-account-token-tracking 1      42h
kube-system         kube-proxy                                           2      42h
kube-system         kube-root-ca.crt                                     1      42h
kube-system         kubeadm-config                                       1      42h
kube-system         kubelet-config                                       1      42h
local-path-storage  kube-root-ca.crt                                     1      42h
local-path-storage  local-path-config                                    4      42h
controlplane $
```

14 configmap

2- create web-app-configmap

```
Editor   Tab 1   +
apiVersion: v1
kind: ConfigMap
metadata:
  name: web-app-configmap
data:
  APP_COLOR: darkblue
~
```

3- create webapp-color

```
Editor   Tab 1   +
apiVersion: v1
kind: Pod
metadata:
  name: webapp-color
spec:
  containers:
    - name: nginx-color
      image: nginx
      envFrom:
        - configMapRef:
            name: web-app-configmap
```

4- secrets in the namespace

```
controlplane $ kubectl get secrets --all-namespaces
NAMESPACE     NAME                  TYPE                            DATA   AGE
kube-system   bootstrap-token-o43zse   bootstrap.kubernetes.io/token   5      42h
```

5- secrets in the bootstrap-token

```
controlplane $ kubectl get secrets --all-namespaces
NAMESPACE     NAME                  TYPE                            DATA   AGE
kube-system   bootstrap-token-o43zse   bootstrap.kubernetes.io/token   5      42h
```

The amount of secrets are 5 secrets which is specified by the data field

6- create a pod called db-pod

```
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  containers:
    - name: mysql
      image: mysql:5.7
~
~
```

7- check its status

```
Events:
  Type     Reason     Age                From               Message
  ----     ------     ----               ----               -------
  Normal   Scheduled  51s                default-scheduler  Successfully assigned default/db-pod to node01
  Normal   Pulling    50s                kubelet            Pulling image "mysql:5.7"
  Normal   Pulled     33s                kubelet            Successfully pulled image "mysql:5.7" in 17.501s (17.501s including wa
iting). Image size: 137909886 bytes.
  Normal   Created    18s (x3 over 32s)  kubelet            Created container mysql
  Normal   Started    18s (x3 over 32s)  kubelet            Started container mysql
  Normal   Pulled     18s (x2 over 31s)  kubelet            Container image "mysql:5.7" already present on machine
  Warning  BackOff    3s (x4 over 30s)   kubelet            Back-off restarting failed container mysql in pod db-pod_default(28beb
e88-72cc-4d46-b3f6-949c692ee8b1)
```

Container mysql:5.7 already exists on the machine

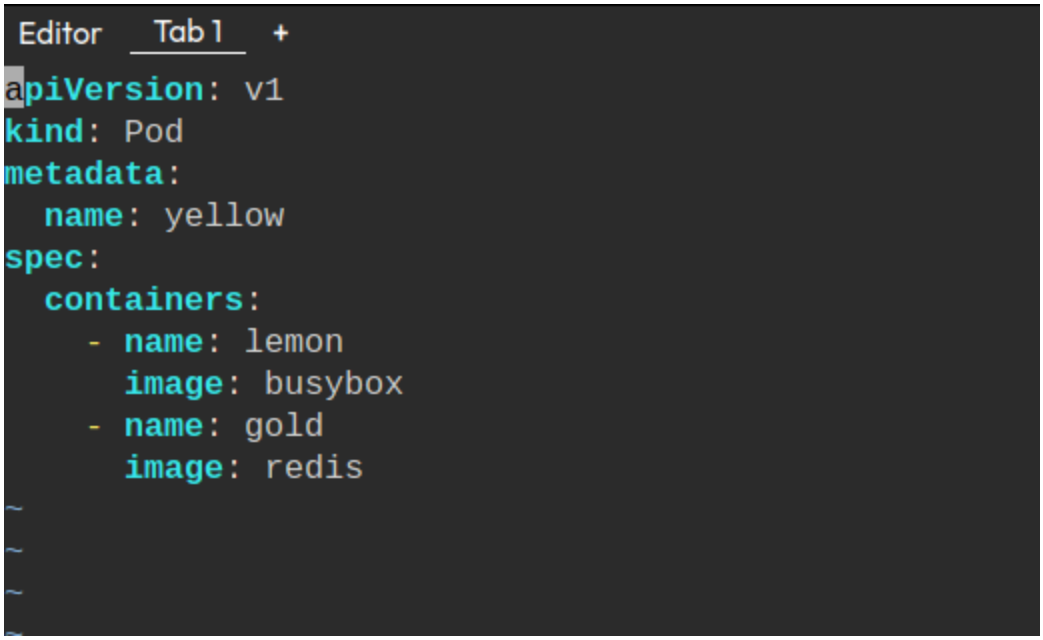
8- create db-secret

```
controlplane $ kubectl create secret generic db-secret --from-literal=MYSQL_DATABASE=sql01   --from-literal=MYSQL_USER=user   --fr
om-literal=MYSQL_PASSWORD=password --from-literal=MYSQL_ROOT_PASSWORD=password123
secret/db-secret created
```

9- attach db-secret to db-pod

```
Editor   Tab 1   +
apiVersion: v1
kind: Pod
metadata:
  name: db-pod
spec:
  containers:
    - name: mysql
      image: mysql:5.7
      envFrom:
        - secretRef:
            name: db-secret
~
~
~
```

10- create a multicontianer pod



11- use bysbox as initContainer with sleep 20

```
apiVersion: v1
kind: Pod
metadata:
  name: red
spec:
  initContainers:
    - name: lemon
      image: busybox
      command: ["/bin/sh","-c"]
      args:
        - sleep 20
  containers:
    - name: gold
      image: redis
```

12- creating pod with said specifications

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: print-envars-greeting
spec:
  containers:
    - name: print-env-container
      image: bash
      command: ["bash","-c"]
      args:
        - echo "$GREETING $COMPANY $GROUP"
        - sleep 3600
      env:
        - name: GREETING
          value: Welcome to
        - name: COMPANY
          value: DevOps
        - name: GROUP
          value: Industries
~
```

13- check the logs of the container

```
controlplane $ kubectl logs print-envars-greeting
Welcome to DevOps Industries
controlplane $
```

14- create an nginx image with curl and startupprobe

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-test
spec:
  containers:
    - name: nginx-pod
      image: nginx
      startupProbe:
        exec:
          command:
            - curl
            - localhost
        periodSeconds: 5
        failureThreshold: 3
~
```

15- creating a pod that checks nginx at root

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: probe-test
spec:
  containers:
    - name: nginx-pod
      image: nginx
      livenessProbe:
        httpGet:
          path: /
          port: 80
        periodSeconds: 5
        failureThreshold: 3
~
~
```

16- what happens to the pod

```
db-pod                1/1     Running            0               26m
live-probe-test       1/1     Running            0               12s
print-envars-greeting 0/1     CrashLoopBackOff   6 (5m1s ago)    10m
```

The pod is running because the liveness check is valid and it can httpget the / at port 80

17- edit to /test.html

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: live-probe-test
spec:
  containers:
    - name: nginx-pod
      image: nginx
      livenessProbe:
        httpGet:
          path: /test.html
          port: 80
        periodSeconds: 5
        failureThreshold: 3
~
```

18- checking the status after

```
live-probe-test   1/1     Running   1 (5s ago)   20s
controlplane $ kubectl get pod live-probe-test  -w
NAME              READY   STATUS    RESTARTS     AGE
live-probe-test   1/1     Running   1 (7s ago)   22s
live-probe-test   1/1     Running   2 (2s ago)   32s
live-probe-test   1/1     Running   3 (2s ago)   47s
live-probe-test   0/1     CrashLoopBackOff   3 (1s ago)   61s
```

After editing it crashes because it doesnt fir the live probe test

19- creating a nodejs container with a readiness probe

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: ready-probe-test
spec:
  containers:
    - name: node-pod
      image: learnk8s/knote-js:1.0.0
      readinessProbe:
        httpGet:
          path: /health
          port: 3000
        failureThreshold: 3
~
~
```

Using https://learnk8s.io/deploying-nodejs-kubernetes#deploying-containerised-applications

20- the server doesnt have a health endpoint at port 3000

```
Events:
  Type     Reason     Age              From               Message
  ----     ------     ----             ----               -------
  Normal   Scheduled  32s              default-scheduler  Successfully assigned default/ready-probe-test to node01
  Normal   Pulling    32s              kubelet            Pulling image "learnk8s/knote-js:1.0.0"
  Normal   Pulled     12s              kubelet            Successfully pulled image "learnk8s/knote-js:1.0.0" in 19.133s (19.133s including waiting). Image size: 86757
514 bytes.
  Normal   Created    12s              kubelet            Created container node-pod
  Normal   Started    12s              kubelet            Started container node-pod
  Warning  Unhealthy  2s (x5 over 12s)  kubelet            Readiness probe failed: Get "http://192.168.1.10:3000/health": dial tcp 192.168.1.10:3000: connect: connectio
n refused
```