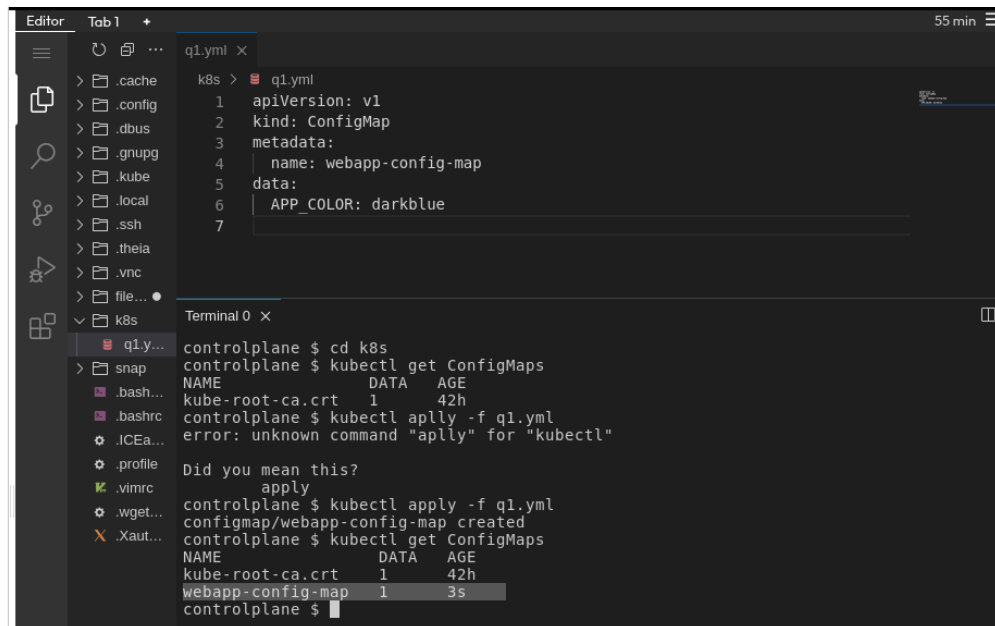


1. How many ConfigMaps exist in the cluster?

```
Terminal 0 X
controlplane $ cd k8s
controlplane $ kubectl get ConfigMaps
NAME          DATA   AGE
kube-root-ca.crt 1       42h
controlplane $
```

2. Create a new ConfigMap Use the spec given below. → ConfigName Name: webapp-config-map → Data: APP_COLOR=darkblue



The screenshot shows a code editor with a file named `q1.yml` containing the following YAML content:

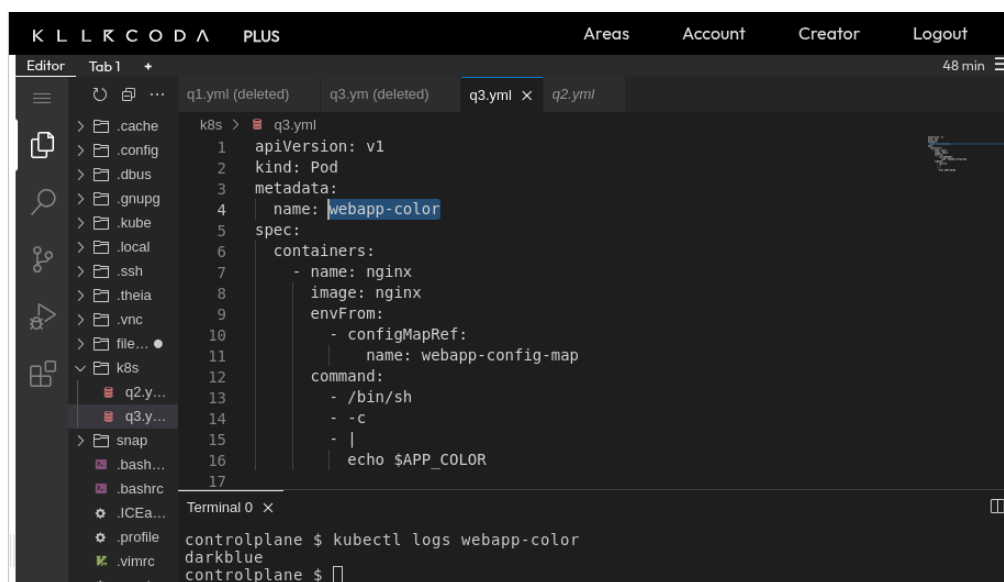
```
1 apiVersion: v1
2 kind: ConfigMap
3 metadata:
4   name: webapp-config-map
5 data:
6   APP_COLOR: darkblue
7
```

Below the editor is a terminal window showing the following commands and output:

```
controlplane $ cd k8s
controlplane $ kubectl get ConfigMaps
NAME          DATA   AGE
kube-root-ca.crt 1       42h
controlplane $ kubectl aply -f q1.yml
error: unknown command "aply" for "kubectl"

Did you mean this?
  apply
controlplane $ kubectl apply -f q1.yml
configmap/webapp-config-map created
controlplane $ kubectl get ConfigMaps
NAME          DATA   AGE
kube-root-ca.crt 1       42h
webapp-config-map 1       3s
controlplane $
```

3. Create a webapp-color POD with nginx image and use the created ConfigMap



The screenshot shows a code editor with a file named `q3.yml` containing the following YAML content:

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: webapp-color
5 spec:
6   containers:
7     - name: nginx
8       image: nginx
9       envFrom:
10        - configMapRef:
11          name: webapp-config-map
12   command:
13     - /bin/sh
14     - -c
15     - |
16       echo $APP_COLOR
17
```

Below the editor is a terminal window showing the following commands and output:

```
controlplane $ kubectl logs webapp-color
darkblue
controlplane $
```

4. How many Secrets exist in the cluster?

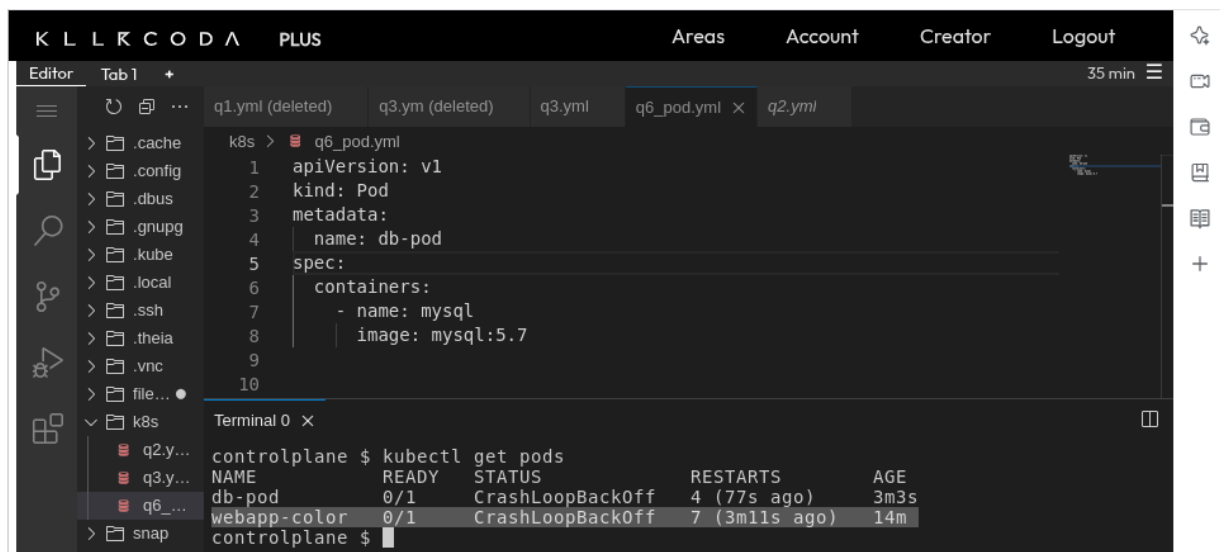
5. How many secrets are defined in the default-token secret?

Answer off 4 & 5

```
Terminal 0 x
controlplane $ kubectl get secrets --all-namespaces
NAMESPACE   NAME                               TYPE                               DATA   AGE
kube-system  bootstrap-token-o43zse             bootstrap.kubernetes.io/token      5       42h
controlplane $ kubectl get secrets
No resources found in default namespace.
controlplane $
```

6. create a POD called db-pod with the image mysql:5.7 then check the POD status

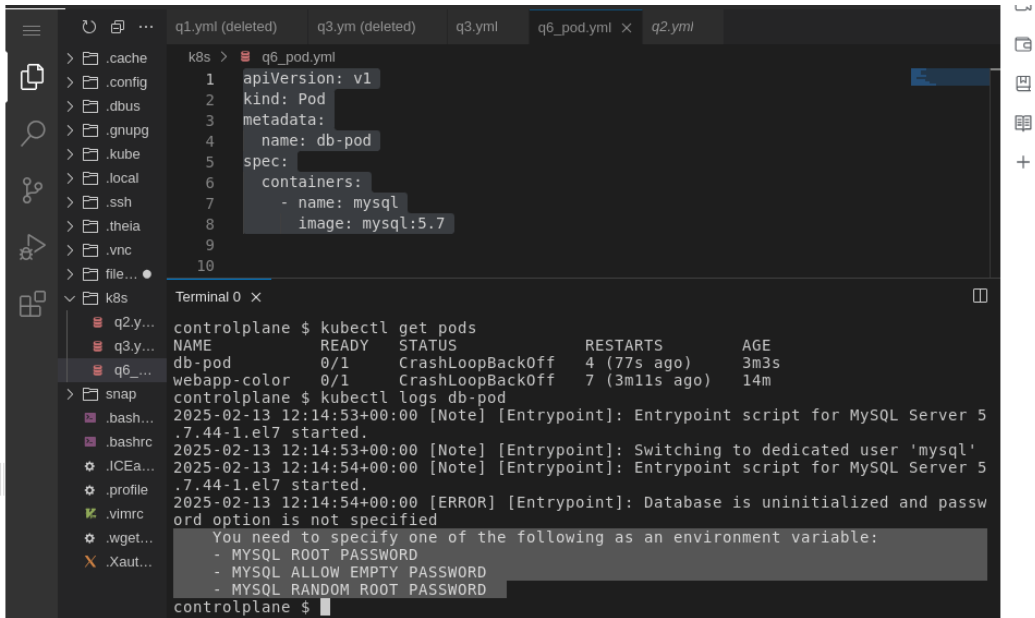
->Db-pod crashed



The screenshot shows a code editor with a file explorer on the left and a terminal window at the bottom. The file explorer shows a directory structure with files like .cache, .config, .dbus, .gnupg, .kube, .local, .ssh, .theia, .vnc, file..., k8s, q2.y..., q3.y..., q6_..., and snap. The terminal window shows the following output:

```
controlplane $ kubectl get pods
NAME      READY   STATUS    RESTARTS   AGE
db-pod    0/1     CrashLoopBackOff   4 (77s ago)    3m3s
webapp-color 0/1     CrashLoopBackOff   7 (3m11s ago)  14m
controlplane $
```

7. why is the db-pod status not ready? -> missing environment variables like MYSQL_ROOT_PASSWORD



The screenshot shows a code editor with a file explorer on the left and a terminal window at the bottom. The file explorer shows a directory structure with files like .cache, .config, .dbus, .gnupg, .kube, .local, .ssh, .theia, .vnc, file..., and k8s. The k8s directory is expanded, showing files q1.yml (deleted), q3.yml (deleted), q3.yml, q6_pod.yml, and q2.yml. The q6_pod.yml file is selected, and its content is displayed in the editor. The content is a Kubernetes Pod manifest for a MySQL database pod named db-pod. The manifest specifies the image as mysql:5.7. The terminal window shows the output of the command `kubectl get pods`, which lists the db-pod with a status of CrashLoopBackOff. The terminal also shows the output of the command `kubectl logs db-pod`, which displays the MySQL server startup logs. The logs indicate that the MySQL server is failing to start because the root password is not specified. The error message is: "You need to specify one of the following as an environment variable: - MYSQL_ROOT_PASSWORD - MYSQL_ALLOW_EMPTY_PASSWORD - MYSQL_RANDOM_ROOT_PASSWORD".

```
k8s > q6_pod.yml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: db-pod
5  spec:
6    containers:
7      - name: mysql
8        image: mysql:5.7
9
10

Terminal 0 X
controlplane $ kubectl get pods
NAME      READY   STATUS              RESTARTS   AGE
db-pod    0/1     CrashLoopBackOff    4 (77s ago) 3m3s
webapp-color 0/1     CrashLoopBackOff    7 (3m11s ago) 14m
controlplane $ kubectl logs db-pod
2025-02-13 12:14:53+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5
.7.44-1.el7 started.
2025-02-13 12:14:53+00:00 [Note] [Entrypoint]: Switching to dedicated user 'mysql'
2025-02-13 12:14:54+00:00 [Note] [Entrypoint]: Entrypoint script for MySQL Server 5
.7.44-1.el7 started.
2025-02-13 12:14:54+00:00 [ERROR] [Entrypoint]: Database is uninitialized and passw
ord option is not specified
You need to specify one of the following as an environment variable:
- MYSQL_ROOT_PASSWORD
- MYSQL_ALLOW_EMPTY_PASSWORD
- MYSQL_RANDOM_ROOT_PASSWORD
controlplane $
```

8. Create a new secret named db-secret with the data given below: → Secret Name: db-secret → Secret 1: MYSQL_DATABASE=sql01 → Secret 2: MYSQL_USER=user1 → Secret 3: MYSQL_PASSWORD=password → Secret 4: MYSQL_ROOT_PASSWORD=password123

```
k8s > q8.yml
1  apiVersion: v1
2  kind: Secret
3  metadata:
4    name: db-secret
5  data:
6    MYSQL_DATABASE: c3FsMDE=
7    MYSQL_USER: dXNlcjE=
8    MYSQL_PASSWORD: cGFzc3dvcmQ=
9    MYSQL_ROOT_PASSWORD: cGFzc3dvcmQxMjM=

Terminal 0 X
controlplane $ echo sql01 | base64
c3FsMDEK
controlplane $ echo user1 | base64
dXNlcjEK
controlplane $ echo password | base64
cGFzc3dvcmQK
controlplane $ echo password123 | base64
cGFzc3dvcmQxMjMK
controlplane $ kubectl apply -f q8.yml
Error from server (BadRequest): error when creating "q8.yml": Secret in version "v1"
cannot be handled as a Secret: json: cannot unmarshal string into Go struct field
Secret.data of type map[string][]uint8
controlplane $ kubectl apply -f q8.yml
error: resource mapping not found for name: "db-secret" namespace: "" from "q8.yml"
: no matches for kind "Secret" in version "apps/v1"
ensure CRDs are installed first
controlplane $ kubectl apply -f q8.yml
error: resource mapping not found for name: "db-secret" namespace: "" from "q8.yml"
: no matches for kind "Secret" in version "apps/v1"
ensure CRDs are installed first
controlplane $ kubectl apply -f q8.yml
secret/db-secret created
controlplane $
```

9. Configure db-pod to load environment variables from the newly created secret. Delete and recreate the pod if required.

```
k8s > q6_pod.yml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: db-pod
5  spec:
6    containers:
7      - name: mysql
8        image: mysql:5.7
9        envFrom:
10         - secretRef:
11           name: db-secret

Terminal 0 X
controlplane $ kubectl delete pod dp-pod
Error from server (NotFound): pods "dp-pod" not found
controlplane $ kubectl delete pod db-pod
pod "db-pod" deleted
controlplane $ kubectl apply -f q6 pod.yml
pod/db-pod created
controlplane $
```

10. Create a multi-container pod with 2 containers. → Name: yellow → Container 1 Name: lemon → Container 1 Image: busybox → Container 2 Name: gold → Container 2 Image: redis

The screenshot shows a code editor with a file explorer on the left. The file explorer shows a directory structure with files like .cache, .config, .dbus, .gnupg, .kube, .local, .ssh, .theia, .vnc, file..., and k8s. The k8s directory is expanded, showing files q1.yml (deleted), q3.yml (deleted), q3.yml, q6_pod.yml, q10.yml, q8.yml, and q2.yml. The q10.yml file is selected, and its content is displayed in the editor. The content is a Kubernetes Pod manifest with the following details: apiVersion: v1, kind: Pod, metadata: name: multi-con, spec: containers: - name: lemon, image: busybox, - name: gold, image: redis. Below the editor, a terminal window is open, showing the command 'kubectl apply -f q10.yml' being executed, resulting in 'pod/multi-con created' and 'controlplane \$'.

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: multi-con
5 spec:
6   containers:
7     - name: lemon
8       image: busybox
9     - name: gold
10      image: redis
```

```
controlplane $ kubectl apply -f q10.yml
pod/multi-con created
controlplane $
```

11. Create a pod red with redis image and use an initContainer that uses the busybox image and sleeps for 20 seconds

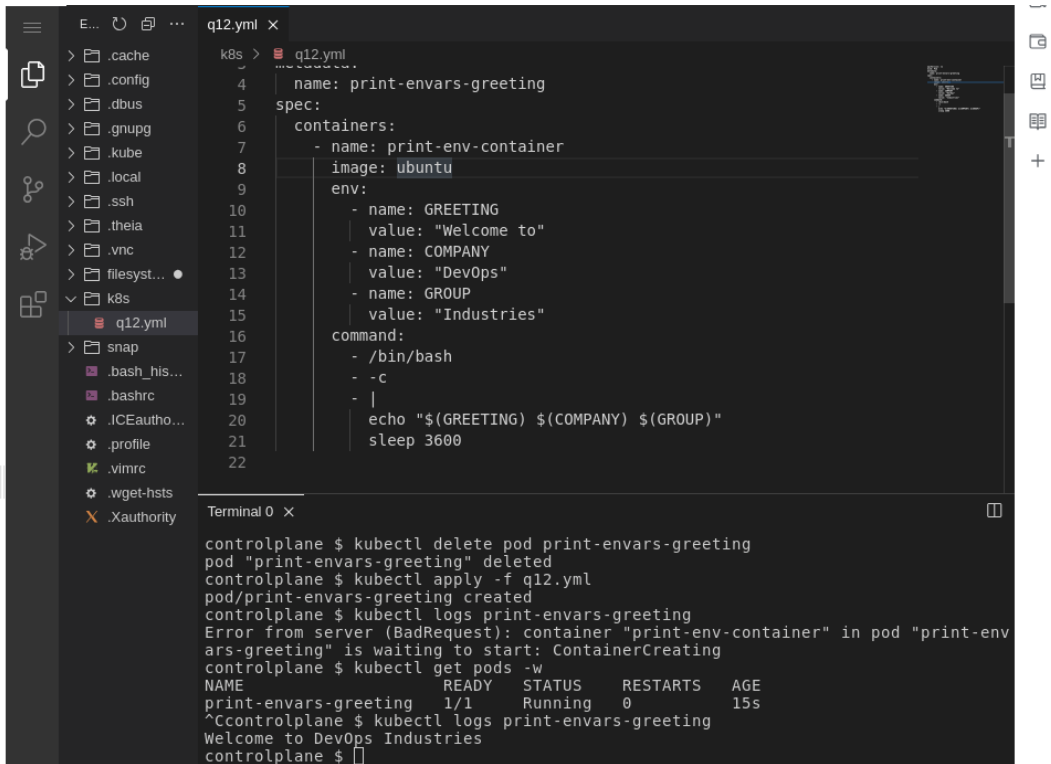
The screenshot shows a code editor with a file explorer on the left. The file explorer shows a directory structure with files like .cache, .config, .dbus, .gnupg, .kube, .local, .ssh, .theia, .vnc, file..., and k8s. The k8s directory is expanded, showing files q10..., q11..., q2.y..., q3.y..., q6_..., and q8.y.... The q11.yml file is selected, and its content is displayed in the editor. The content is a Kubernetes Pod manifest with the following details: apiVersion: v1, kind: Pod, metadata: name: red, spec: initContainers: - name: busybox, image: busybox, command: - /bin/sh, - -c, - |, sleep 20, containers: - name: redis, image: redis. Below the editor, a terminal window is open, showing the command 'kubectl apply -f q11.yml' being executed, resulting in 'pod/red created' and 'controlplane \$'.

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: red
5 spec:
6   initContainers:
7     - name: busybox
8       image: busybox
9       command:
10         - /bin/sh
11         - -c
12         - |
13           sleep 20
14   containers:
15     - name: redis
16       image: redis
```

```
controlplane $ kubectl apply -f q11.yml
pod/red created
controlplane $
```

12. Create a pod named print-envs-greeting, Configure spec as, the container name should be print-env-container and use bash image, Create three environment variables: → GREETING and its value should be “Welcome to” → COMPANY and its value should be “DevOps” → GROUP and its value should be “Industries” Use

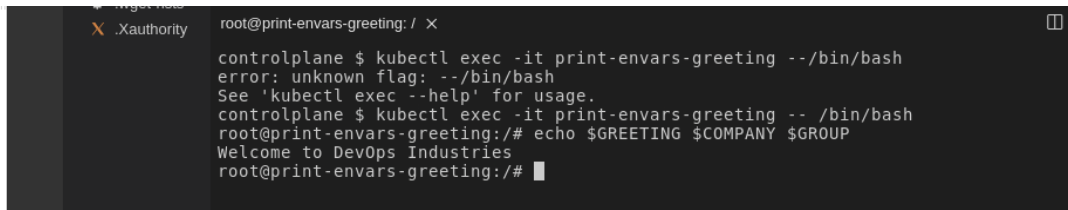
command to echo ["\$(GREETING) \$(COMPANY) \$(GROUP)"] message and sleep the container 3600.



```
q12.yml
k8s > q12.yml
4   name: print-envvars-greeting
5   spec:
6     containers:
7     - name: print-env-container
8       image: ubuntu
9       env:
10        - name: GREETING
11          value: "Welcome to"
12        - name: COMPANY
13          value: "DevOps"
14        - name: GROUP
15          value: "Industries"
16      command:
17        - /bin/bash
18        - -c
19        - |
20          echo "$(GREETING) $(COMPANY) $(GROUP)"
21          sleep 3600
22

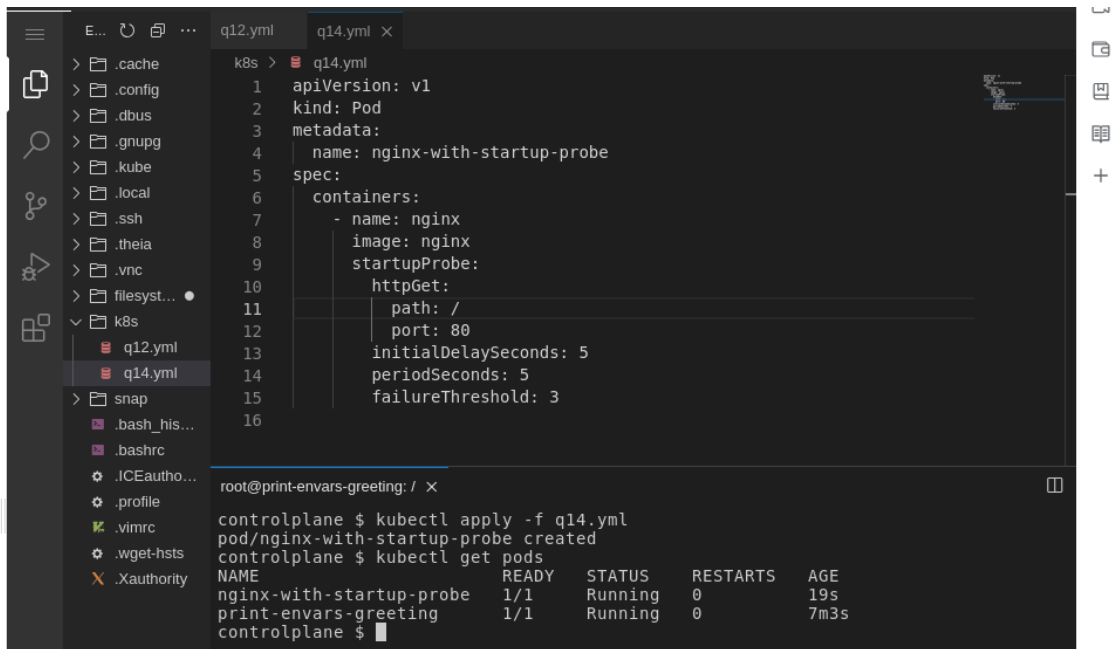
Terminal 0 X
controlplane $ kubectl delete pod print-envvars-greeting
pod "print-envvars-greeting" deleted
controlplane $ kubectl apply -f q12.yml
pod/print-envvars-greeting created
controlplane $ kubectl logs print-envvars-greeting
Error from server (BadRequest): container "print-env-container" in pod "print-envvars-greeting" is waiting to start: ContainerCreating
controlplane $ kubectl get pods -w
NAME                READY   STATUS    RESTARTS   AGE
print-envvars-greeting 1/1     Running   0           15s
^C
controlplane $ kubectl logs print-envvars-greeting
Welcome to DevOps Industries
controlplane $
```

13. You can check the output using command.



```
root@print-envvars-greeting: / #
controlplane $ kubectl exec -it print-envvars-greeting -- /bin/bash
error: unknown flag: -- /bin/bash
See 'kubectl exec --help' for usage.
controlplane $ kubectl exec -it print-envvars-greeting -- /bin/bash
root@print-envvars-greeting: /# echo $GREETING $COMPANY $GROUP
Welcome to DevOps Industries
root@print-envvars-greeting: /#
```

14. Create a pod with a container running the nginx image. → Configure a startupProbe that checks if Nginx is ready using curl. → Set the probe to check every 5 seconds with a failure threshold of 3. → What happens if the container takes longer to start than expected?



```
q14.yml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nginx-with-startup-probe
5  spec:
6    containers:
7      - name: nginx
8        image: nginx
9        startupProbe:
10         httpGet:
11           path: /
12         port: 80
13         initialDelaySeconds: 5
14         periodSeconds: 5
15         failureThreshold: 3
16

root@print-envvars-greeting: / X
controlplane $ kubectl apply -f q14.yml
pod/nginx-with-startup-probe created
controlplane $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-with-startup-probe            1/1     Running   0           19s
print-envvars-greeting              1/1     Running   0           7m3s
controlplane $
```

The startupProbe will keep checking every 5 seconds after the initial delay of 5 seconds. If the Nginx service is not ready within the failureThreshold -> 3 failures, the container will be considered unhealthy, and Kubernetes will mark it as failed. Kubernetes will not send traffic to the container until the startupProbe succeeds. If the probe fails, Kubernetes will try to restart the pod.

15. Deploy an Nginx pod with a livenessProbe that checks /

```
Editor Tab 1 + 43 min
q12.yml q14.yml q15.yml X
k8s > q15.yml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx-with-liveness-probe
5 spec:
6   containers:
7     - name: nginx
8       image: nginx
9       ports:
10        - containerPort: 80
11       livenessProbe:
12         httpGet:
13           path: /
14           port: 80
15         initialDelaySeconds: 5
16         periodSeconds: 5
17         failureThreshold: 3
18
root@print-envvars-greeting: / X
controlplane $ kubectl apply -f q15.yml
pod/nginx-with-liveness-probe created
controlplane $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
nginx-with-liveness-probe          1/1     Running   0           4s
nginx-with-startup-probe           1/1     Running   0          3m36s
print-envvars-greeting             1/1     Running   0          10m
controlplane $
```

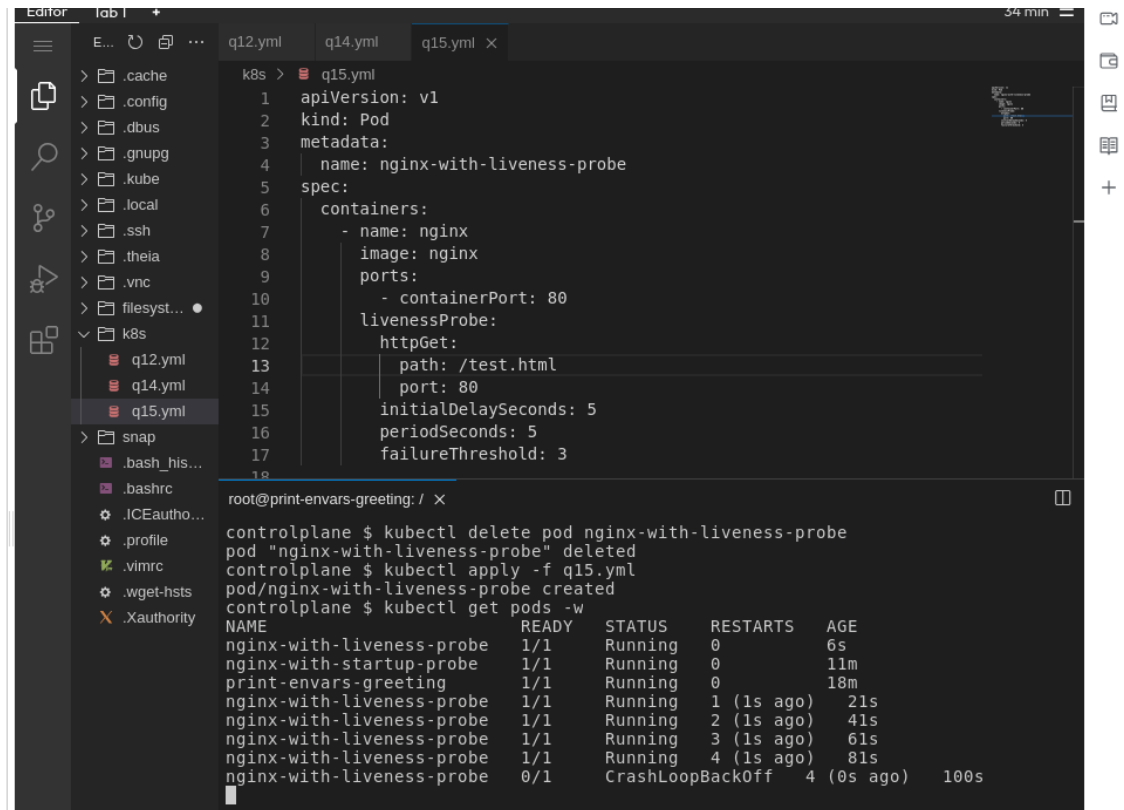
```
q12.yml q14.yml q15.yml X
k8s > q15.yml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx-with-liveness-probe
5 spec:
6   containers:
7     - name: nginx
8       image: nginx
9       ports:
10        - containerPort: 80
11       livenessProbe:
12         httpGet:
13           path: /dskdndsda
14           port: 80
15         initialDelaySeconds: 5
16         periodSeconds: 5
17         failureThreshold: 3
18
root@print-envvars-greeting: / X
controlplane $ kubectl delete pod nginx-with-liveness-probe
pod "nginx-with-liveness-probe" deleted
controlplane $ kubectl apply -f q15.yml
pod/nginx-with-liveness-probe created
controlplane $ kubectl get pods -w
NAME                                READY   STATUS    RESTARTS   AGE
nginx-with-liveness-probe          1/1     Running   0           4s
nginx-with-startup-probe           1/1     Running   0          6m43s
print-envvars-greeting             1/1     Running   0          13m
nginx-with-liveness-probe          1/1     Running   1 (1s ago)  22s
nginx-with-liveness-probe          1/1     Running   2 (0s ago)  36s
nginx-with-liveness-probe          1/1     Running   3 (1s ago)  52s
nginx-with-liveness-probe          0/1     CrashLoopBackOff   3 (0s ago)  66s
nginx-with-liveness-probe          1/1     Running   4 (28s ago)  94s
```

16. What happens to the pod?

The liveness probe will continue to pass every 5 seconds, and the pod will remain in the Running state.

Kubernetes will consider the pod healthy and continue serving traffic to it.

17. Edit the livenessProbe inside the pod to /test.html.



The screenshot shows a terminal window with a file explorer on the left. The file explorer shows a directory structure with files like .cache, .config, .dbus, .gnupg, .kube, .local, .ssh, .theia, .vnc, filesys..., k8s, q12.yml, q14.yml, q15.yml, snap, .bash_his..., .bashrc, .ICEAutho..., .profile, .vimrc, .wget-hsts, and .Xauthority. The terminal window shows the following commands and output:

```
k8s > q15.yml
1  apiVersion: v1
2  kind: Pod
3  metadata:
4    name: nginx-with-liveness-probe
5  spec:
6    containers:
7      - name: nginx
8        image: nginx
9        ports:
10       - containerPort: 80
11       livenessProbe:
12         httpGet:
13           path: /test.html
14           port: 80
15         initialDelaySeconds: 5
16         periodSeconds: 5
17         failureThreshold: 3
18

root@print-envvars-greeting: / X
controlplane $ kubectl delete pod nginx-with-liveness-probe
pod "nginx-with-liveness-probe" deleted
controlplane $ kubectl apply -f q15.yml
pod/nginx-with-liveness-probe created
controlplane $ kubectl get pods -w
NAME                                READY    STATUS    RESTARTS   AGE
nginx-with-liveness-probe           1/1      Running   0           6s
nginx-with-startup-probe             1/1      Running   0          11m
print-envvars-greeting               1/1      Running   0          18m
nginx-with-liveness-probe            1/1      Running   1 (1s ago)  21s
nginx-with-liveness-probe            1/1      Running   2 (1s ago)  41s
nginx-with-liveness-probe            1/1      Running   3 (1s ago)  61s
nginx-with-liveness-probe            1/1      Running   4 (1s ago)  81s
nginx-with-liveness-probe            0/1      CrashLoopBackOff 4 (0s ago)  100s
```

18. What happens to the pod after the edit?

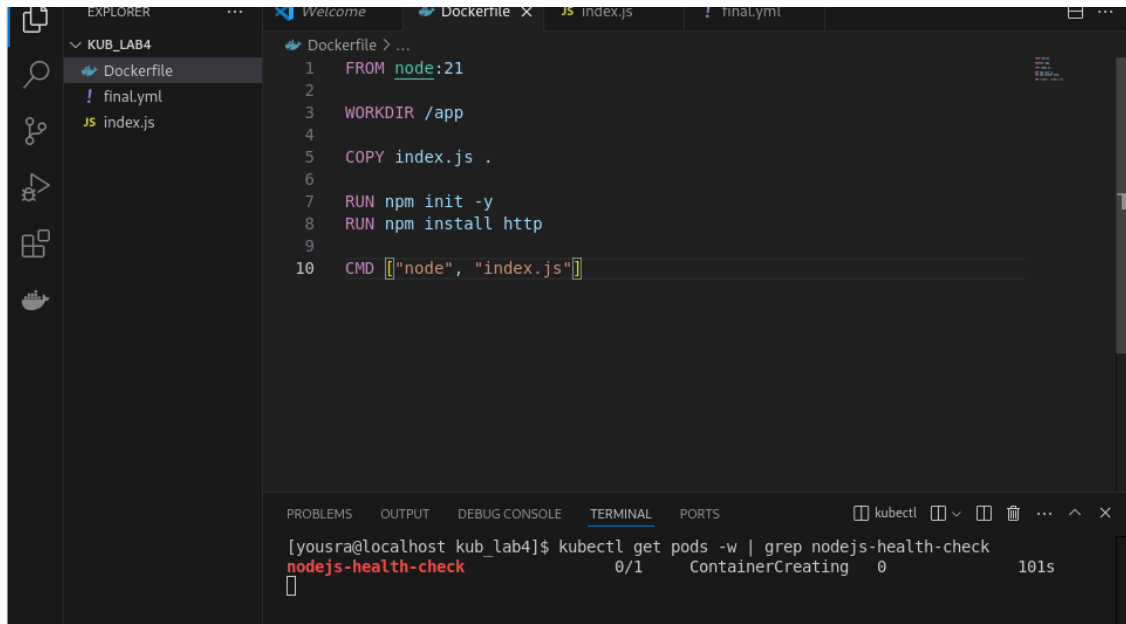
/test.html doesn't exist or the probe path is incorrect, the livenessProbe will fail, and the pod will keep restarting until it either passes the probe or is manually corrected.

If the path /test.html is valid (or if you create that file), the probe will succeed, and the pod will remain healthy.

19. Create a pod running a simple Node.js web server.

20. Use a readinessProbe to check the HTTP endpoint (/health).

21. Test what happens when the application is not ready.S

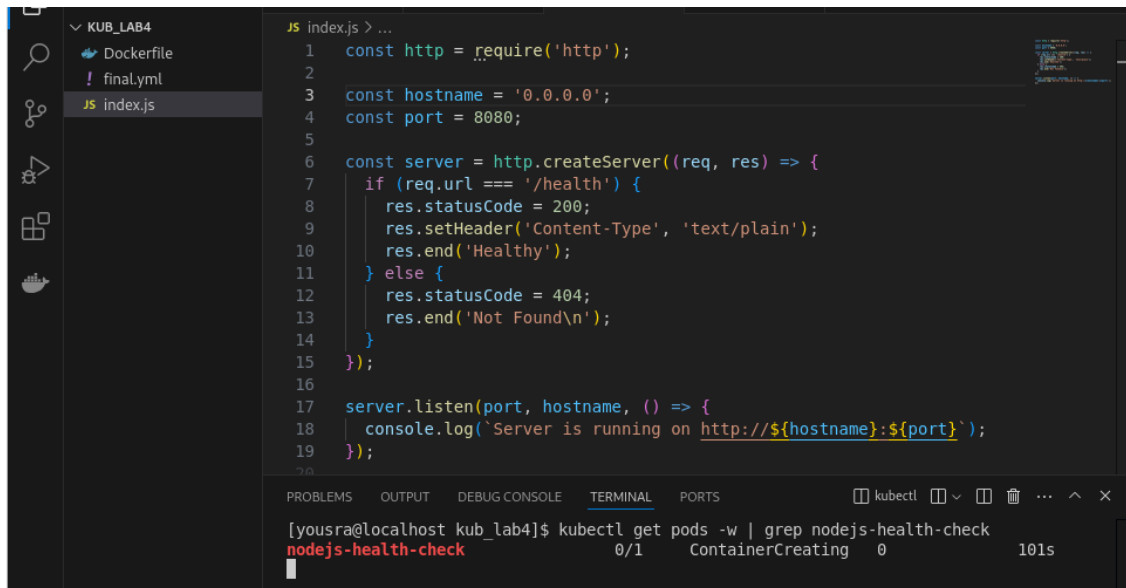


The image shows a VS Code editor window with the Explorer sidebar on the left displaying the file structure for 'KUB_LAB4', including 'Dockerfile', 'final.yml', and 'index.js'. The main editor area shows the 'Dockerfile' with the following content:

```
1 FROM node:21
2
3 WORKDIR /app
4
5 COPY index.js .
6
7 RUN npm init -y
8 RUN npm install http
9
10 CMD ["node", "index.js"]
```

The bottom panel shows the 'TERMINAL' tab with the following command and output:

```
[yousra@localhost kub_lab4]$ kubectl get pods -w | grep nodejs-health-check
nodejs-health-check      0/1      ContainerCreating      0      101s
```



The image shows a VS Code editor window with the Explorer sidebar on the left displaying the file structure for 'KUB_LAB4', including 'Dockerfile', 'final.yml', and 'index.js'. The main editor area shows the 'index.js' file with the following content:

```
1 const http = require('http');
2
3 const hostname = '0.0.0.0';
4 const port = 8080;
5
6 const server = http.createServer((req, res) => {
7   if (req.url === '/health') {
8     res.statusCode = 200;
9     res.setHeader('Content-Type', 'text/plain');
10    res.end('Healthy');
11  } else {
12    res.statusCode = 404;
13    res.end('Not Found\n');
14  }
15 });
16
17 server.listen(port, hostname, () => {
18   console.log(`Server is running on http://${hostname}:${port}`);
19 });
```

The bottom panel shows the 'TERMINAL' tab with the following command and output:

```
[yousra@localhost kub_lab4]$ kubectl get pods -w | grep nodejs-health-check
nodejs-health-check      0/1      ContainerCreating      0      101s
```

EXPLORER

KUB_LAB4

Dockerfile

final.yml

index.js

! final.yml

```
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nodejs-health-check
5 spec:
6   containers:
7     - name: nodejs
8       image: yousra314/my-nodejs-health-check
9       ports:
10        - containerPort: 8080
11       readinessProbe:
12         httpGet:
13           path: /health
14           port: 8080
15       initialDelaySeconds: 5
16       periodSeconds: 5
17       failureThreshold: 3
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

kubectl

pod/nodejs-health-check configured

```
[yousra@localhost kub_lab4]$ kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
deployment-1-6d845b97db-fm9ft	1/1	Running	3 (59m ago)	4d2h
deployment-1-6d845b97db-mlr6s	1/1	Running	3 (59m ago)	4d2h
deployment-1-6d845b97db-vvkqv	1/1	Running	3 (59m ago)	4d2h
elasticsearch-lm445	0/1	ImagePullBackOff	0	3d2h
my-pod	1/1	Running	3 (59m ago)	4d5h
my-pod-1	1/1	Running	3 (59m ago)	4d5h
my-pod-nginx	1/1	Running	3 (59m ago)	4d2h
nginx-7d746df889-6b9w2	1/1	Running	1 (59m ago)	3d1h
nginx-7d746df889-q6kz4	1/1	Running	1 (59m ago)	3d1h
nginx-deployment-7dd57fbd7-865nl	1/1	Running	3 (59m ago)	4d1h
nginx-deployment-7dd57fbd7-nw7tc	1/1	Running	3 (59m ago)	4d1h
nginx-deployment-7dd57fbd7-wrqch	1/1	Running	3 (59m ago)	4d1h
nginx-hwhrt	1/1	Running	1 (59m ago)	3d2h
nginx-pod	1/1	Running	1 (59m ago)	3d1h
nodejs-health-check	0/1	Running	16 (3m12s ago)	59m
redis	1/1	Running	3 (59m ago)	4d3h
replica-set-1-f7qdz	1/1	Running	3 (59m ago)	4d2h
replica-set-1-vqdqz	1/1	Running	3 (59m ago)	4d2h
replica-set-1-xsg7f	1/1	Running	3 (59m ago)	4d2h
test-nginx-pod	1/1	Running	1 (59m ago)	3d2h
web-app-64cd7668-k5t44	1/1	Running	1 (59m ago)	3d2h
web-app-64cd7668-qdbz7	1/1	Running	1 (59m ago)	3d2h
web-statefulset-0	1/1	Running	1 (59m ago)	3d2h
web-statefulset-1	1/1	Running	1 (59m ago)	3d2h

```
[yousra@localhost kub_lab4]$ kubectl get pods -w | grep nodejs-health-check
nodejs-health-check 1/1 Running 16 (3m19s ago) 59m
```

OUTLINE

TIMELINE

Ln 15, Col 31

Spaces: 2

UTF-8

LF

YAML