# Linux Foundation

# CKA

## Certified Kubernetes Administrator (CKA)

# Latest Version: 6.0

## Question: 1

You have a Kubernetes cluster with a Deployment named 'web-app' that runs a web application. You need to set up a mechanism to automatically scale the deployment based on the CPU utilization of the pods. The scaling should be triggered when the average CPU utilization across all pods reaches 70%. You should set the minimum and maximum replicas to 2 and 5 respectively.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:
**Solution (Step by Step) :**
1. Create a Horizontal Pod Autoscaler (HPA):
- Use "kubectl create hpa' command to create an HPA resource.
- Specify the name of the HPA, the Deployment to scale, the target CPU utilization (70%), and the minimum and maximum replicas.
kubectl create hpa web-app-hpa --min=2 --max=5 --cpu-utilization-percentage=70 --target-ref=Deployment/web-app
2. Verify the HPA Creation:
- Use 'kubectl get hpa' command to check if the HPA was created successfully. You should see an HPA named 'web-app-hpa' with the configured settings.
3. Monitor the Scaling Behavior:
- You can use the ‘kubectl get pods -l command to monitor the number of pods running as the CPU utilization changes.
- When the average CPU utilization across the pods reaches 70%, the HPA will automatically scale up the Deployment to add more pods.
- Conversely, when the CPU utilization falls below the threshold, the HPA will scale down the Deployment to reduce the number of pods.
Ensure that the 'metrics-server' is installed in your cluster to enable CPU utilization monitoring.,

## Question: 2

You have a Deployment named 'my-app' with 5 replicas running an application container. You need to implement a rolling update strategy that allows for a maximum of 2 pods to be unavailable at any given time during the update process. Additionally, you want to ensure that the update process is triggered automatically whenever a new image is pushed to the Docker Hub repository 'my.org/my-app:latest'.

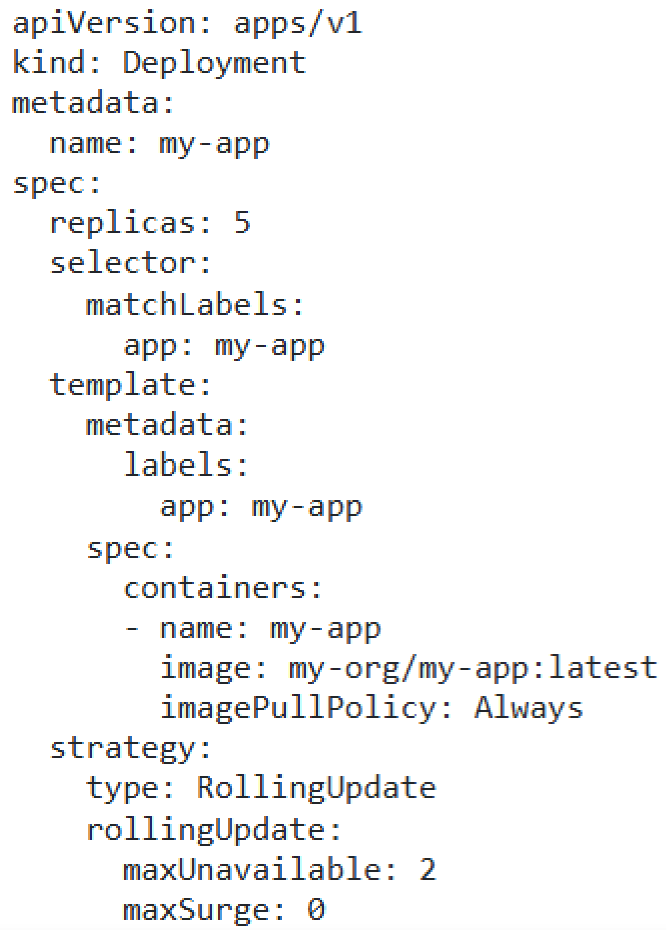
A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:

**Solution (Step by Step) :**

1. Update the Deployment YAML:
- Update the 'replicas' to 5.
- Define 'maxUnavailable: 2' and ‘maxSurge: 0' in the 'strategy.rollingUpdate’ section to control the rolling update process.
- Configure a 'strategy.type' to ‘RollingUpdate' to trigger a rolling update when the deployment is updated.
- Add a 'spec.template.spec.imagePullPolicy: Always' to ensure that the new image is pulled even if it exists in the pod's local cache.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
spec:
  replicas: 5
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
      - name: my-app
        image: my-org/my-app:latest
        imagePullPolicy: Always
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 0
```

2. Create the Deployment:
- Apply the updated YAML file using 'kubectl apply -f my-app.yamP
3. Verify the Deployment:
- Check the status of the deployment using 'kubectl get deployments my-app' to confirm the rollout and updated replica count.
4. Trigger the Automatic Update:
- Push a new image to the 'my.org/my-app:latest' Docker Hub repository.
5. Monitor the Deployment:
- Use 'kubectl get pods -l app=my-app' to monitor the pod updates during the rolling update process. You will observe that up to 2 pods are terminated at a time, while new pods with the updated image are created.
6. Check for Successful Update:

- Once the deployment is complete, use 'kubectl describe deployment my-app' to see that the 'updatedReplicas' field matches the 'replicas' field, indicating a successful update.

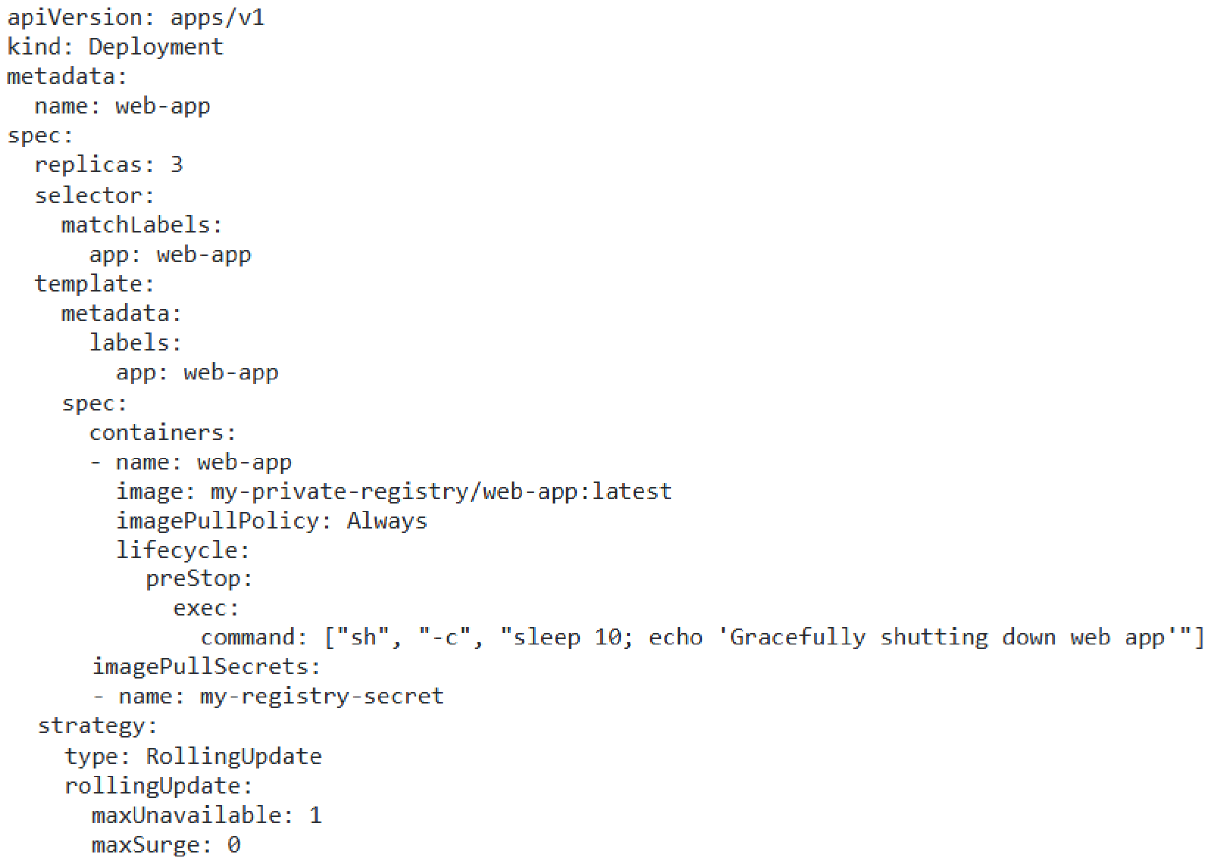
## Question: 3

You are running a Deployment named ‘web-app' with 3 replicas of a web application container. The container image is hosted in a private registry accessible via a secret named 'my-registry-secret'. You need to implement a rolling update strategy that allows for a maximum of one pod to be unavailable at any given time during the update process. Additionally, you need to configure a 'pre-stop' hook for the container that gracefully shuts down the web application before it is terminated.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:
**Solution (Step by Step) :**
1. Update the Deployment YAML:
- Update the 'replicas' to 3.
- Define 'maxUnavailable: 1' and 'maxSurge: 0' in the 'strategy.rollingUpdate' section to control the rolling update process.
- Configure a 'strategy.type' to 'RollingUpdate' to trigger a rolling update when the deployment is updated.
- Add a Always' to ensure that the new image is pulled even if it exists in the pod's local cache.
- Add a hook to define a script that gracefully shuts down the web application before the pod is terminated.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: web-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: web-app
  template:
    metadata:
      labels:
        app: web-app
    spec:
      containers:
      - name: web-app
        image: my-private-registry/web-app:latest
        imagePullPolicy: Always
        lifecycle:
          preStop:
            exec:
              command: ["sh", "-c", "sleep 10; echo 'Gracefully shutting down web app'"]
      imagePullSecrets:
      - name: my-registry-secret
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
```

2. Create the Deployment:
- Apply the updated YAML file using 'kubectl apply -f web-app.yamP
3. Verify the Deployment:
- Check the status of the deployment using 'kubectl get deployments web-app' to confirm the rollout and updated replica count.
4. Trigger the Automatic Update:
- Push a new image to the 'my-private-registry/web-app:latest' private registry.
5. Monitor the Deployment:
- Use "kubectl get pods -l app=web-app' to monitor the pod updates during the rolling update process. You will observe that one pod is terminated at a time, while one new pod with the updated image is created.
6. Check for Successful Update:
- Once the deployment is complete, use 'kubectl describe deployment web-app' to see that the 'updatedReplicas' field matches the ‘replicas' field, indicating a successful update.

## Question: 4

You have a Deployment named 'database-deployment' with 2 replicas of a database container. You want to implement a rolling update strategy that allows for a maximum of one pod to be unavailable at any given time. However, you also want to ensure that the database container is restarted automatically when it encounters a failure or crashes. This restart policy should be applied only to the database containers within the deployment and not affect any other containers in the same pod.

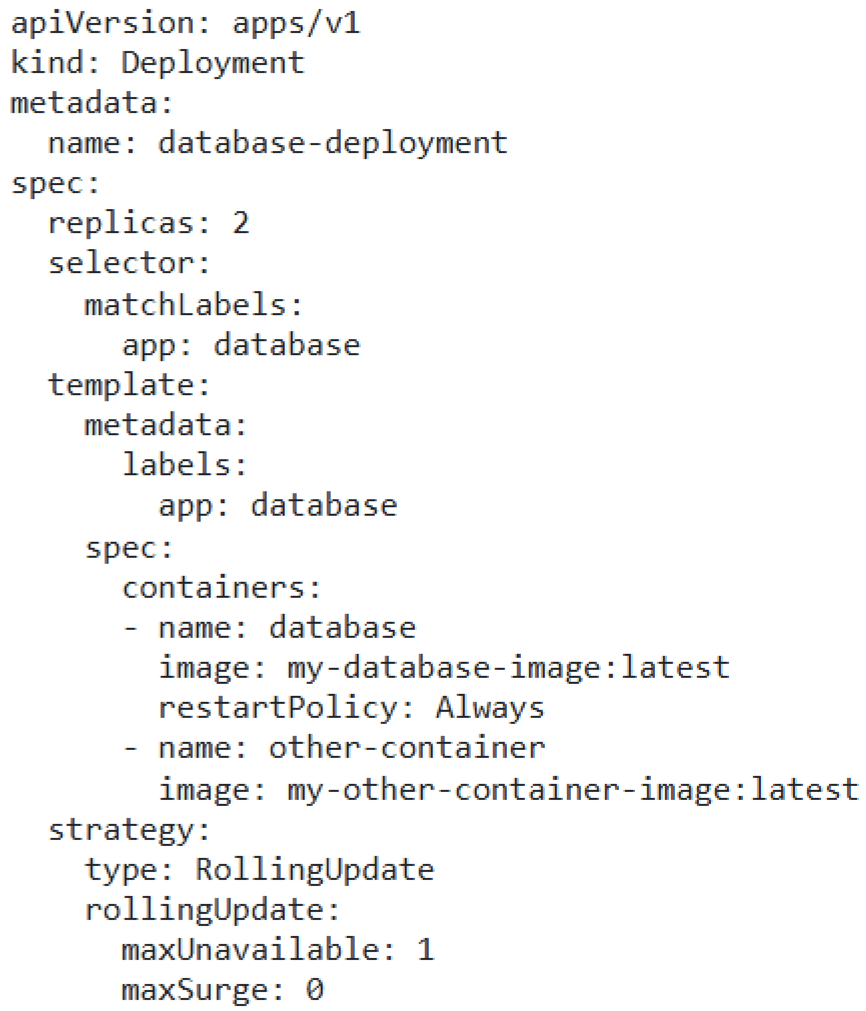
A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:

**Solution (Step by Step) :**

1. Update the Deployment YAML:
- Update the 'replicas' to 2.
- Define 'maxUnavailable: 1' and ‘maxSurge: 0' in the •strategy.rollingUpdate' section to control the rolling update process.
- Configure a 'strategy.type' to 'RollingUpdate' to trigger a rolling update when the deployment is updated.
- Set 'spec.template.spec.containers[0].restartPolicy: Always' to ensure the database container restarts automatically when it encounters a failure.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: database-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: database
  template:
    metadata:
      labels:
        app: database
    spec:
      containers:
      - name: database
        image: my-database-image:latest
        restartPolicy: Always
      - name: other-container
        image: my-other-container-image:latest
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 1
      maxSurge: 0
```

2. Create the Deployment:
- Apply the updated YAML file using 'kubectl apply -f database-deployment.yaml'

3. Verify the Deployment:

- Check the status of the deployment using ‘kubectl get deployments database-deployment’ to confirm the rollout and updated replica count.
4. Test the Restart Policy:
- Trigger a failure in the 'database' container within one of the pods. This can be done by sending a SIGKILL signal to the container or by simulating a crash within the container itself.
- Observe that the 'database' container will be restarted automatically while other containers in the same pod will remain unaffected.
5. Trigger the Automatic Update:
- Push a new image to the 'my-database-image:latest• registry.
6. Monitor the Deployment:
- Use "kubectl get pods -l app=database' to monitor the pod updates during the rolling update process.
7. Check for Successful Update:
- Once the deployment is complete, use ‘kubectl describe deployment database-deployment' to see that the
'updatedReplicas' field matches the 'replicas' field, indicating a successful update.

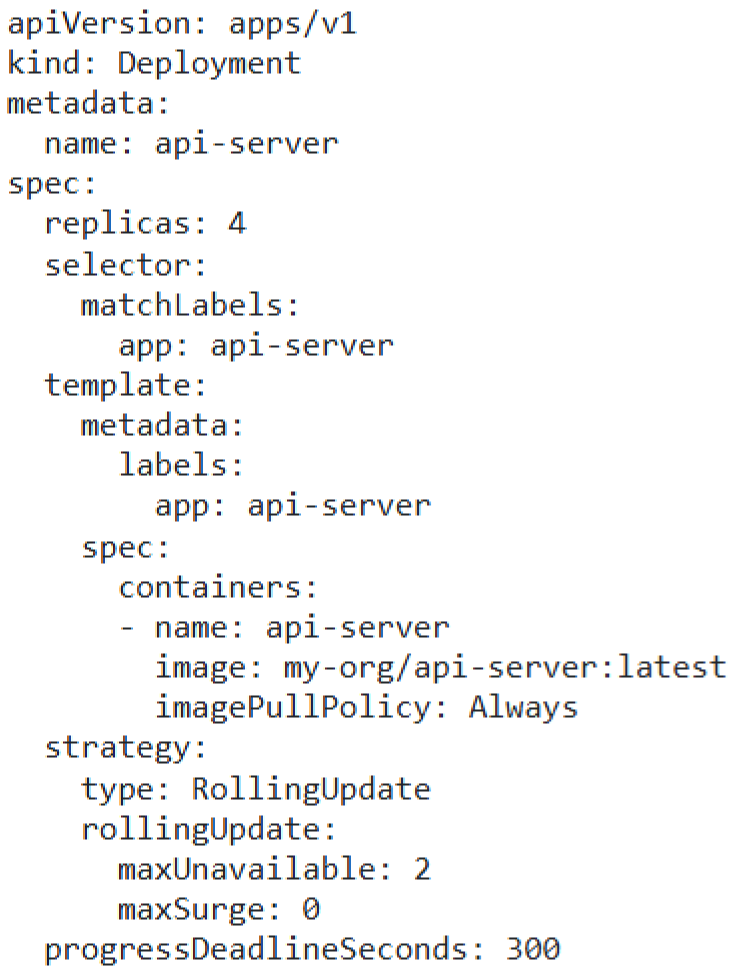
## Question: 5

You have a Deployment named 'api-server' with 4 replicas of an API server container. You need to implement a rolling update strategy that allows for a maximum of 2 pods to be unavailable at any given time. You also want to ensure that the update process is triggered automatically whenever a new image is pushed to the Docker Hub repository "my.org/api-server:latest'. Furthermore, you want to ensure that the update process is completed within a specified timeout of 5 minutes. If the update fails to complete within the timeout, the deployment should revert to the previous version.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:
**Solution (Step by Step) :**
1. Update the Deployment YAML:
- Update the 'replicas' to 4.
- Define 'maxUnavailable: 2' and •maxSurge: in the 'strategy.rollingUpdate' section to control the rolling update process.
- Configure a 'strategy.type' to to trigger a rolling update when the deployment is updated.
- Set 'spec.template.spec.containers[0].imagePullPolicy: Always' to ensure that the new image is pulled even if it exists in the pod's local cache.
- Add a 'spec.progressDeadlineSeconds: 300' to set a timeout of 5 minutes for the update process.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: api-server
spec:
  replicas: 4
  selector:
    matchLabels:
      app: api-server
  template:
    metadata:
      labels:
        app: api-server
    spec:
      containers:
      - name: api-server
        image: my-org/api-server:latest
        imagePullPolicy: Always
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 0
  progressDeadlineSeconds: 300
```

2. Create the Deployment:
- Apply the updated YAML file using 'kubectl apply -f api-server.yaml'
3. Verify the Deployment:
- Check the status of the deployment using "kubectl get deployments api-server' to confirm the rollout and updated replica count.
4. Trigger the Automatic Update:
- Push a new image to the 'my.org/api-server:latest' Docker Hub repository.
5. Monitor the Deployment:
- Use Vxubectl get pods -l app=api-server' to monitor the pod updates during the rolling update process.
6. Observe Rollback if Timeout Exceeds:
- If the update process takes longer than 5 minutes to complete, the deployment will be rolled back to the previous version. This can be observed using 'kubectl describe deployment api-server’ and checking the 'updatedReplicas' and 'availableReplicas' fields.

## Question: 6

You have a Deployment named 'worker-deployment' with 10 replicas of a worker container. You need to implement a rolling update strategy that allows for a maximum of 3 pods to be unavailable at any given time during the update process. You also want to ensure that the update process is completed within a specified timeout of 10 minutes. If the update fails to complete within the timeout, the deployment should revert to the previous version. Additionally, you want to implement a pause functionality to temporarily halt the rolling update process.

A. See the solution below with Step by Step Explanation.

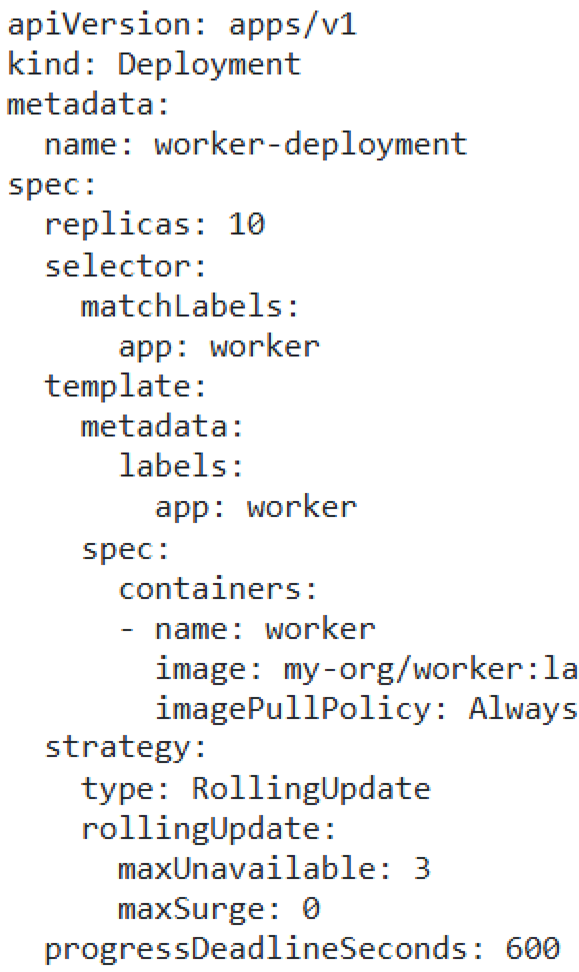
**Answer: A**

Explanation:

**Solution (Step by Step) :**

1. Update the Deployment YAML:

- Update the 'replicas' to 10.
- Define 'maxUnavailable: 3' and 'maxSurge: 0' in the 'strategy.rollingUpdate' section to control the rolling update process.
- Configure a 'strategy.type' to 'RollingUpdate' to trigger a rolling update when the deployment is updated.
- Set Always' to ensure that the new image is pulled even if it exists in the pod's local cache.
- Add a 'spec.progressDeadlineSeconds: 600' to set a timeout of 10 minutes for the update process.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: worker-deployment
spec:
  replicas: 10
  selector:
    matchLabels:
      app: worker
  template:
    metadata:
      labels:
        app: worker
    spec:
      containers:
      - name: worker
        image: my-org/worker:latest
        imagePullPolicy: Always
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 3
      maxSurge: 0
  progressDeadlineSeconds: 600
```

2. Create the Deployment:

- Apply the updated YAML file using 'kubectl apply -f worker-deployment.yaml'

3. Verify the Deployment:
- Check the status of the deployment using 'kubectl get deployments worker-deployment' to confirm the rollout and updated replica count.
4. Trigger the Automatic Update:
- Push a new image to the 'my.org/worker:latest' Docker Hub repository.
5. Monitor the Deployment:
- Use "kubectl get pods -l app=worker' to monitor the pod updates during the rolling update process.
6. Pause the Rolling Update:
- To pause the rolling update process, use the following command:

```
bash
kubectl rollout pause deployment worker-deployment
```

7. Resume the Rolling Update:
- To resume the rolling update process, use the following command:

```
bash
kubectl rollout resume deployment worker-deployment
```

8. Observe Rollback if Timeout Exceeds:
- If the update process takes longer than 10 minutes to complete, the deployment will be rolled back to the previous version. This can be observed using ‘kubectl describe deployment worker-deployment' and checking the 'updatedReplicas' and 'availableReplicas" fields.

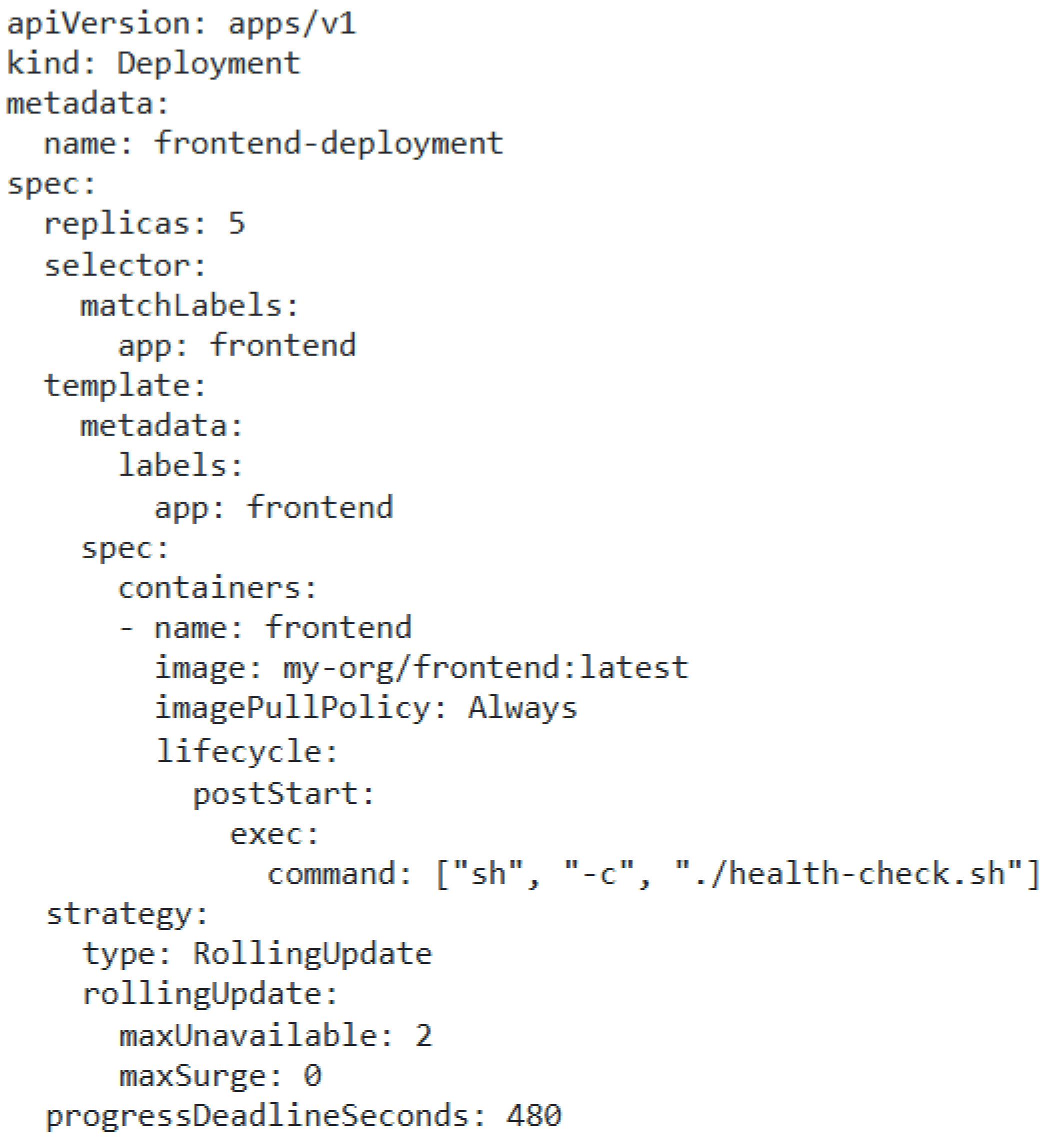
## Question: 7

You have a Deployment named 'frontend-deployment' with 5 replicas of a frontend container. You need to implement a rolling update strategy that allows for a maximum of 2 pods to be unavailable at any given time. You also want to ensure that the update process is completed within a specified timeout of 8 minutes. If the update fails to complete within the timeout, the deployment should revert to the previous version. Additionally, you want to configure a 'post-start' hook for the frontend container that executes a health check script to verify the application's readiness before it starts accepting traffic.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:
**Solution (Step by Step) :**
1. Update the Deployment YAML:
- Update the 'replicas' to 5.
- Define 'maxUnavailable: 2' and 'maxSurge: 0' in the 'strategy.rollingUpdate' section to control the rolling update process.
- Configure a 'strategy.type' to 'RollingUpdate' to trigger a rolling update when the deployment is updated.
- Set Always' to ensure that the new image is pulled even if
it exists in the pod's local cache.
- Add a 'spec.progressDeadlineSeconds: 480' to set a timeout of 8 minutes for the update process.
- Add a 'spec.template.spec.containers[0].lifecycle.postStart' hook to define a script that executes a health check script before the container starts accepting traffic.



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
spec:
  replicas: 5
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
      - name: frontend
        image: my-org/frontend:latest
        imagePullPolicy: Always
        lifecycle:
          postStart:
            exec:
              command: ["sh", "-c", "./health-check.sh"]
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxUnavailable: 2
      maxSurge: 0
  progressDeadlineSeconds: 480
```

2. Create the Deployment:
- Apply the updated YAML file using 'kubectl apply -f frontend-deployment.yaml'
3. Verify the Deployment:
- Check the status of the deployment using 'kubectl get deployments frontend-deployment' to confirm the rollout and updated replica count.
4. Trigger the Automatic Update:
- Push a new image to the 'my.org/frontend:latest' Docker Hub repository.
5. Monitor the Deployment:
- Use 'kubectl get pods -l app=frontend' to monitor the pod updates during the rolling update process.
6. Observe Rollback if Timeout Exceeds:
- If the update process takes longer than 8 minutes to complete, the deployment will be rolled back to the previous version. This can be observed using 'kubectl describe deployment frontend-deployment' and checking the 'updatedReplicas' and 'availableReplicas' fields.,

## Question: 8

You have a Deployment named 'web-app' running a web application with two pods. The web application is configured to access a database with the connection string stored in a ConfigMap named 'db- config'. You need to update the database connection string in the ConfigMap without restarting the pods.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:

**Solution (Step by Step) :**

1. Update the ConfigMap:

- Modify the 'db-config' ConfigMap to include the new connection string. This can be done using 'kubectl patch' or 'kubectl edit' commands. For instance, using 'kubectl patch':

```
kubectl patch configmap db-config -p '{"data": {"db-connection-string": "new-connection-string"}}'
```

2. Verify the Updated ConfigMap:

- Confirm the changes were applied to the ConfigMap by checking its data with 'kubectl get configmap db- config -o yaml':

```
kubectl get configmap db-config -o yaml
```

This should show the updated 'db-connection-string' value.

3. Observe the Pods:

- Monitor the pods in the 'web-app' Deployment. Since ConfigMaps are mounted as volumes, the updated connection string will automatically be available to the pods without any manual restarts. You can check the pods using get pods -l app=web-app'.

```
kubectl get pods -l app=web-app
```

4. Confirm Application Functionality:

- Verify that the web application is now using the updated database connection by performing relevant actions within the application, such as querying data or performing other operations.

## Question: 9

You have a Deployment named 'mysql-deployment' running a MySQL database server. You need to store the MySQL root password securely using a Secret. This password should be used by the database server when it starts.

A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:

**Solution (Step by Step) :**

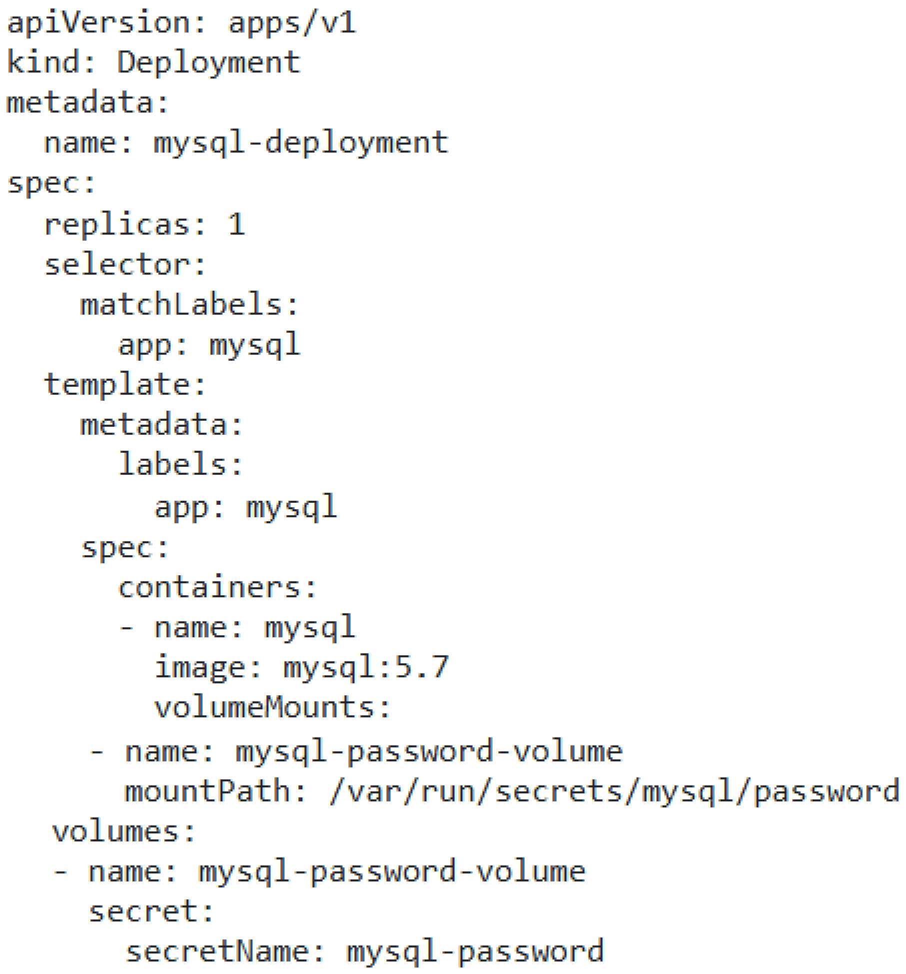
1. Create the Secret:

- Create a Secret named 'mysql-password’ to store the root password.
- Use the 'kubectl create secret generic' command with the '--from-literal' flag to create a generic Secret with a key-value pair:

kubectl create secret generic mysql-password —from-literal=mysql-root password="your_strong_password"

2. Modify the Deployment:

- Update the 'mysql-deployment' Deployment's Pod template to mount the •mysql-password' Secret as a volume.
- Use 'volumeMounts' to specify where the Secret should be mounted within the container, and 'volumes' to define the Secret as a volume source.
- Update the MySQL server's configuration (e.g., the ‘my.cnf file) to read the password from the mounted volume.



```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mysql-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mysql
  template:
    metadata:
      labels:
        app: mysql
    spec:
      containers:
      - name: mysql
        image: mysql:5.7
        volumeMounts:
    - name: mysql-password-volume
      mountPath: /var/run/secrets/mysql/password
  volumes:
  - name: mysql-password-volume
    secret:
      secretName: mysql-password
```

3. Apply the Changes:

- Apply the modified Deployment YAML using 'kubectl apply -f mysql-deployment.yamP.

4. Restart the MySQL Pod:

- Restart the MySQL pod for it to read the password from the mounted volume. This can be achieved using 'kubectl delete pod’.

5. Verify the Password:

- Connect to the MySQL database using the provided password and confirm it works correctly.

## Question: 10

You have a Deployment named 'nginx-deployment• running an Nginx server. The Nginx configuration file is stored in a ConfigMap named 'nginx-config'. You need to dynamically update the Nginx configuration file without restarting the Nginx pods.

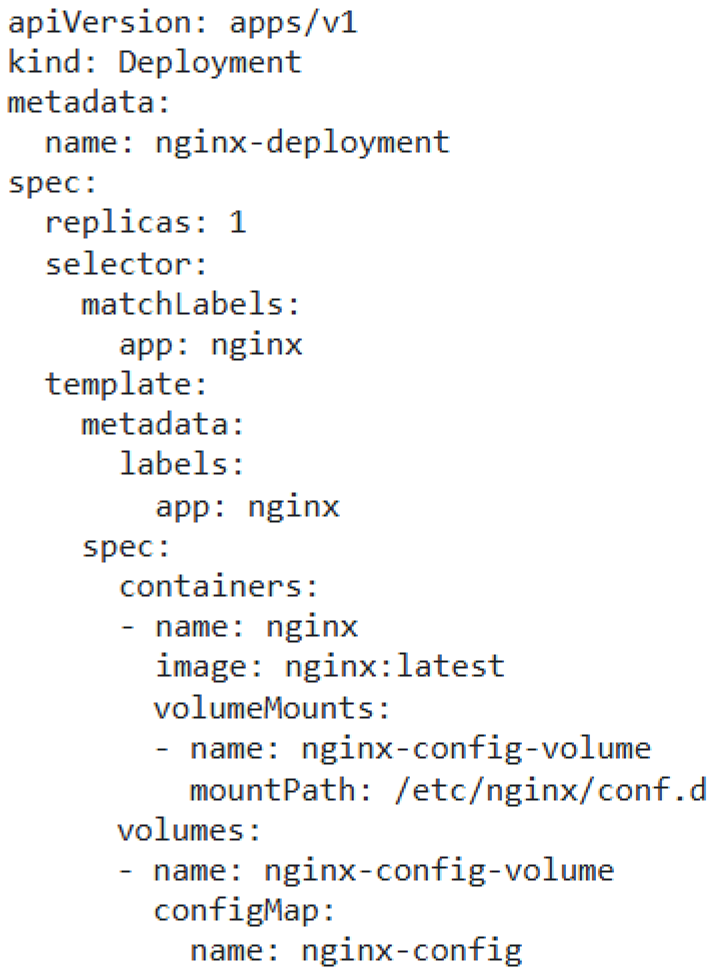
A. See the solution below with Step by Step Explanation.

**Answer: A**

Explanation:
**Solution (Step by Step) :**
1 . Create the ConfigMap (if not already existing):
- Define a ConfigMap named 'nginx-config' containing the Nginx configuration file. For example, create a file 'nginx.conf with the desired configuration and then create the ConfigMap using 'kubectl create configmap nginx-config --from-file=nginx.conf:

kubectl create configmap nginx-config --from-file=nginx.conf

2. Configure Nginx Deployment:
- Modify the 'nginx-deployment' Deployment to mount the 'nginx-config' ConfigMap as a volume.
- Use 'volumeMounts' to specify where the ConfigMap should be mounted (e.g., '/etc/nginx/conf.d/' ) and 'volumes' to define the ConfigMap as a volume source.
- Update the Nginx container's configuration to use the mounted configuration file (e.g., 'nginx -g daemon off;').



```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:latest
        volumeMounts:
        - name: nginx-config-volume
          mountPath: /etc/nginx/conf.d
      volumes:
      - name: nginx-config-volume
        configMap:
          name: nginx-config
```

3. Update the ConfigMap:
- Modify the ‘nginx-config' ConfigMap with the new configuration content. This can be done using ‘kubectl patch' or 'kubectl edit':
kubectl patch configmap nginx-config -p '{"data": {"nginx.conf": "new_nginx_configuration"}}'
4. Observe Nginx Pods:
- Monitor the Nginx pods in the 'nginx-deployment' Deployment. Since the ConfigMap is mounted as a volume, Nginx will automatically reload the configuration file without restarting the pod.
5. Verify the Update:
- Use 'kubectl logs' or 'kubectl exec' to examine the Nginx pod's logs and confirm that the new configuration is being used.