Welcome to Cert007 - Your Ultimate IT Certification Partner

- Real Exam Questions
- Free Updates
- Expert Support
- Instant Access
- Money-Back Guarantee

Visit us at **https://www.cert007.com/** for more information

Exam : **EX280**

Title : Red Hat Certified OpenShift Administrator exam

Version : DEMO

1.You are tasked with deploying a highly available application in OpenShift. Create a Deployment using YAML to deploy the nginx container with three replicas, ensuring that it runs successfully. Verify that the Deployment is active, all replicas are running, and the application can serve requests properly. Provide a complete walkthrough of the process, including necessary commands to check deployment status.

Answer:

See the Solution below.

Solution:

1. Create a Deployment YAML file named nginx-deployment.yaml with the following content:

apiVersion: apps/v1
kind: Deployment
metadata:
name: nginx-deployment
labels:
app: nginx
spec:
replicas: 3
selector:
matchLabels:
app: nginx
template:
metadata:
labels:
app: nginx
spec:
containers:
- name: nginx image: nginx:latest ports:
- containerPort: 80

2. Deploy the file using the command: kubectl apply -f nginx-deployment.yaml
3. Check the status of the deployment: kubectl get deployments
kubectl get pods
4. Test the application by exposing the Deployment:
kubectl expose deployment nginx-deployment --type=NodePort --port=80 kubectl get svc
5. Use the NodePort and cluster IP to confirm that the application is serving requests.

Explanation:

Deployments provide a scalable and declarative way to manage applications. YAML manifests ensure the configuration is consistent, while NodePort services expose the application for testing. Verifying replicas ensures that the application is running as expected and resilient.

2.Your team requires an application to load specific configuration data dynamically during runtime. Create a ConfigMap to hold key-value pairs for application settings, and update an existing Deployment to use this ConfigMap. Provide a complete YAML definition for both the ConfigMap and the updated Deployment, and demonstrate how to validate that the configuration is applied correctly.

Answer:

See the Solution below.

Solution:
1. Create a ConfigMap YAML file named app-config.yaml:
apiVersion: v1
kind: ConfigMap
metadata:
name: app-config
data:
APP_ENV: production
APP_DEBUG: "false"
2. Apply the ConfigMap using: kubectl apply -f app-config.yaml
3. Update the Deployment YAML to reference the ConfigMap:
apiVersion: apps/v1
kind: Deployment
metadata:
name: app-deployment
spec:
replicas: 1
selector:
matchLabels:
app: my-app
template:
metadata:
labels:
app: my-app
spec:
containers:
- name: app-container image: nginx:latest env:
- name: APP_ENV valueFrom:
configMapKeyRef: name: app-config key: APP_ENV
- name: APP_DEBUG valueFrom:
configMapKeyRef: name: app-config key: APP_DEBUG
4. Apply the updated Deployment: kubectl apply -f app-deployment.yaml
5. Verify the pod environment variables: kubectl exec -it <pod-name> -- env | grep APP
Explanation:
ConfigMaps decouple configuration data from the application code, enabling environment-specific settings without altering the deployment logic. Using environment variables from ConfigMaps ensures flexibility and reduces maintenance complexity.

3.Your cluster requires an application to be exposed to external users. Use the OpenShift CLI to expose an application running on the nginx Deployment as a service, making it accessible via NodePort. Provide step-by-step instructions, including testing the accessibility of the service from the host machine.
Answer:
See the Solution below.
Solution:

1. Expose the Deployment as a NodePort service:
kubectl expose deployment nginx-deployment --type=NodePort --port=80
2. Retrieve the service details: kubectl get svc
3. Identify the NodePort and access the application using <Node-IP>:<NodePort> in a browser or curl:
curl http://<Node-IP>:<NodePort>
Explanation:
NodePort services allow external access to applications for testing or specific use cases. This ensures that developers and testers can interact with the application from outside the cluster without requiring advanced ingress configurations.

4.Your organization needs a shared storage solution for an application running on OpenShift. Configure a PersistentVolume (PV) and a PersistentVolumeClaim (PVC), and update an existing Deployment to use this storage. Include a demonstration of how to validate that the storage is mounted correctly in the application pods.
Answer:
See the Solution below.
Solution:
1. Create a PersistentVolume YAML file:
apiVersion: v1
kind: PersistentVolume
metadata:
name: shared-pv
spec:
capacity:
storage: 1Gi
accessModes:
- ReadWriteOnce hostPath:
path: /data/shared-pv
2. Create a PersistentVolumeClaim YAML file:
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
name: shared-pvc
spec:
accessModes:
- ReadWriteOnce resources:
requests: storage: 1Gi
3. Update the Deployment YAML to use the PVC:
volumes:
- name: shared-storage persistentVolumeClaim:
claimName: shared-pvc
containers:
- name: app-container image: nginx:latest volumeMounts:
- mountPath: "/usr/share/nginx/html"

name: shared-storage
4. Apply the updated Deployment and verify:
kubectl exec -it <pod-name> -- ls /usr/share/nginx/html
Explanation:
Persistent volumes and claims abstract storage allocation in Kubernetes. By binding PVs to PVCs, applications can use persistent storage seamlessly across deployments, ensuring data persistence beyond pod lifecycles.

5.Configure a Role-based Access Control (RBAC) setup to allow a user named dev-user to list all pods in the test-project namespace. Provide YAML definitions for the Role and RoleBinding, and demonstrate how to verify the permissions.
Answer:
See the Solution below.
Solution:
1. Create a Role YAML file:
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
namespace: test-project
name: pod-reader
rules:
- apiGroups: [""]
resources: ["pods"]
verbs: ["get", "list"]
2. Create a RoleBinding YAML file:
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
name: read-pods-binding
namespace: test-project
subjects:
- kind: User
name: dev-user
roleRef:
kind: Role
name: pod-reader
apiGroup: rbac.authorization.k8s.io
3. Apply the Role and RoleBinding:
kubectl apply -f role.yaml
kubectl apply -f rolebinding.yaml
4. Verify the user’s permissions:
kubectl auth can-i list pods --as dev-user -n test-project
Explanation:
RBAC provides fine-grained access control, enabling administrators to assign specific permissions to

users or groups. Verification ensures the intended access level is configured.