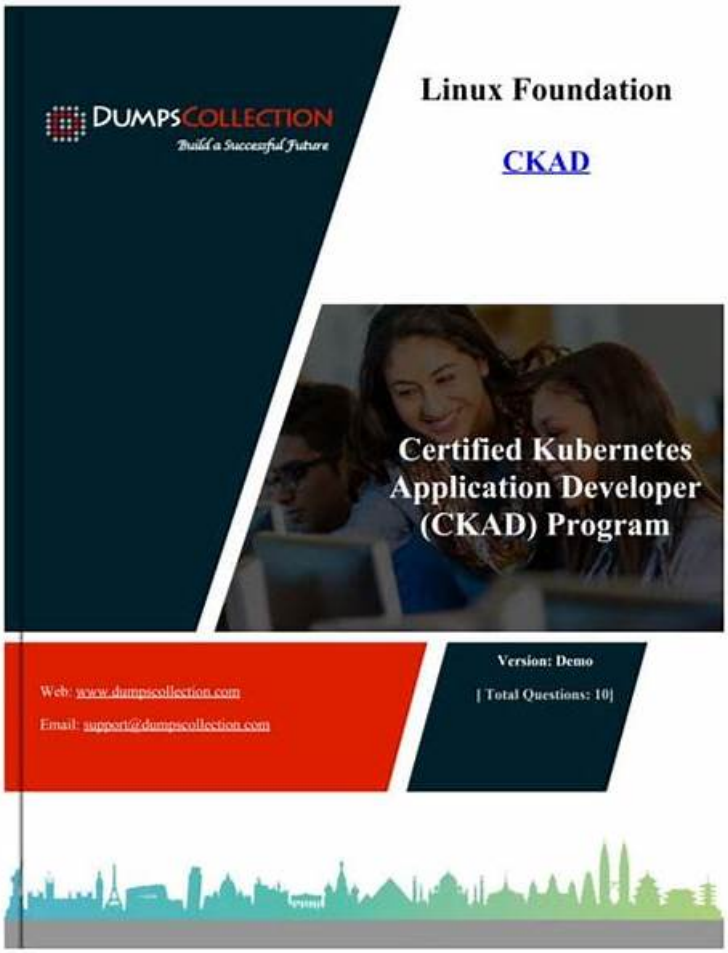
Linux Foundation CKAD인증시험덤프자료 & CKAD시험문제집



2026 ExamPassdump 최신 CKAD PDF 버전 시험 문제집과 CKAD 시험 문제 및 답변 무료 공유: https://drive.google.com/open?id=1_jleR_DQekGqlGJJk7mtBPZMBf31Mkml

ExamPassdump덤프를 IT국제인증자격증 시험대비자료중 가장 퍼펙트한 자료로 거듭날수 있도록 최선을 다하고 있습니다. Linux Foundation CKAD 덤프에는Linux Foundation CKAD시험문제의 모든 범위와 유형을 포함하고 있어 시험적중율이 높아 구매한 분이 모두 시험을 패스한 인기덤프입니다.만약 시험문제가 변경되어 시험에서 불합격 받으신다면 덤프비용 전액 환불해드리기에 안심하셔도 됩니다.

CKAD 자격증 시험은 Kubernetes 애플리케이션을 배포, 구성 및 관리할 수 있는 능력을 평가하는 실습 중심 시험입니다. 시험은 엄격한 시간 제한 내에서 완료해야 하는 일련의 성능 기반 작업으로 구성되어 있으며, 프로덕션 환경에서 Kubernetes를 작업할 때 개발자가 직면하는 실제 문제를 반영합니다. 시험은 Kubernetes API와 함께 작업하는 능력, Kubernetes 객체를 생성하고 관리하는 능력, 그리고 Kubernetes 애플리케이션을 작업할 때 발생하는 일반적인 문제를 해결하는 능력을 평가하기 위해 설계되었습니다.

>> Linux Foundation CKAD인증시험 덤프자료 <<

최신버전 CKAD인증시험 덤프자료 덤프는 Linux Foundation Certified Kubernetes Application Developer Exam 시험패스의 지름길

현재 많은 IT인사들이 같은 생각하고 있습니다. 그것은 바로Linux Foundation CKAD인증시험자격증 취득으로 하여 IT업계의 아주 중요한 한걸음이라고 말입니다.그만큼Linux Foundation CKAD인증시험의 인기는 말 그대로 하늘을 찌르고 있습니다,

CKAD 인증은 Kubernetes 및 컨테이너 화 분야에서 경력을 발전시키려는 개발자에게 귀중한 자격 증명입니다. 인증은 후보자가 Kubernetes 기반 애플리케이션을 설계, 구축 및 배포하는 데 필요한 기술과 지식을 가지고 있으며 Kubernetes를 효과적으로 사용하여 컨테이너화 된 응용 프로그램을 관리 할 수 있음을 보여줍니다. CKAD 인증은 업계 리더에 의해 인정되며 구직 시장의 공인 전문가에게 경쟁력있는 우위를 제공합니다.

최신 Kubernetes Application Developer CKAD 무료샘플문제 (Q97-Q102):

질문 # 97
You have a ConfigMap named 'my-app-config' that stores environment variables for your application. You want to dynamically update tne values in the ConfigMap without restarting the pods. How would you achieve this using a Kubernetes Patch?

정답：

설명：
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Get the Existing ConfigMap Data:
bash
kubectl get configmap my-app-config -o yaml > my-app-config.yaml
2. Modify the YAML File:
- Open 'my-app-config.yaml and update the values in the 'data' section as required- For example, if you want to change the value of 'DATABASE_HOST to Sdb.new.example.coms:

3. Patch the ConfigMap: bash kubectl patch configmap my-app-config -p "S(cat my-app-config_yaml)" 4. Verify the Changes: bash kubectl get configmap my-app-config -o yaml 5. Observe the Updated Values: - The pods will automatically pick up the updated values without the need for restarting. - You can confirm this by checking the environment variables within the pod using 'kubectl exec -it - bash -c 'env" This method allows for dynamic updates to the ConfigMap without restarting the pods, making it a convenient way to manage environment variables in your Kubernetes applications.

질문 # 98
You have a custom resource definition (CRD) named that represents a database resource in your Kubernetes cluster. You want to create a custom operator that automates the creation and management of these database instances. The operator should handle the following:
- Creation: When a new 'database.example.com' resource is created, the operator should provision a new PostgreSQL database instance on the cluster-
- Deletion: When a 'database.example_com' resource is deleted, the operator should clean up the corresponding PostgreSQL database instance.
- Scaling: If the 'spec-replicas' field of the 'database-example.com' resource is updated, the operator should scale the number of database instances accordingly.
Provide the necessary Kubernetes resources, custom operator code, and steps to implement this operator. You should use the 'Operator Framework' to build and deploy this operator

정답：

설명：
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Create the CRD:

```yaml
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: databases.example.com
spec:
  group: example.com
  names:
    kind: Database
    plural: databases
    singular: database
  scope: Namespaced
  versions:
  - name: v1
    served: true
    storage: true
  subresources:
    status: {}
```

- Apply this YAML file to your cluster using ' kubectl apply -f database-crd.yamr. 2. Create the Operator Project: - IJse the Operator Framework' to initialize a new operator project bash operator-sdk init -domain example.com -repo example.com/database-operator --version VO.O. I -license apache2 - Replace 'example_com' with your desired domain name. 3. Define the Custom Resource: - Create a 'database_types.go' file in the 'api/vl' directory of your project. - Define the 'Database' resource as a custom resource struct Go package v1 import (metavl "k8s.iofapimachinery/pkg/apis/meta/v1" // DatabaseSpec defines the desired state of Database type DatabaseSpec struct { If Replicas specifies the number of database instances to run.

```
Replicas int32 `json:"replicas"`
```

// Password is the password for the database users.

```
Password string `json:"password"`
```

} // DatabaseStatus defines the observed state of Database type DatabaseStatus struct { // Replicas is the actual number of database instances running.

```
Replicas int32 `json:"replicas"`
```

// Ready indicates if the database is ready to accept connections.

```
Ready bool `json:"ready"`
```

}

```go
// +kubebuilder:object:root=true
// +kubebuilder:subresource:status

// Database is the Schema for the databases API
type Database struct {
metav1.TypeMeta `json:",inline"`
metav1.ObjectMeta `json:"metadata,omitempty"`

Spec DatabaseSpec `json:"spec,omitempty"`
Status DatabaseStatus `json:"status,omitempty"`
}

// +kubebuilder:object:root=true

// DatabaseList contains a list of Database
type DatabaseList struct {
metav1.TypeMeta `json:",inline"`
metav1.ListMeta `json:"metadata,omitempty"`
Items []Database `json:"items"`
}

func init() {
SchemeBuilder.Register(&Database{}, &DatabaseList{})
}
```

4. Implement the Controller Logic: - Create a 'database_controller.go' file in the 'controllers' directory- - Implement the logic for

creating, deleting, and scaling database instances.

```
go
package controllers

import (
"context"
"fmt"
appsv1 "k8s.io/api/apps/v1"
corev1 "k8s.io/api/core/v1"
"k8s.io/apimachinery/pkg/api/errors"
metav1 "k8s.io/apimachinery/pkg/apis/meta/v1"
"k8s.io/apimachinery/pkg/runtime"
"k8s.io/apimachinery/pkg/types"
"sigs.k8s.io/controller-runtime/pkg/client"
"sigs.k8s.io/controller-runtime/pkg/controller"
"sigs.k8s.io/controller-runtime/pkg/handler"
"sigs.k8s.io/controller-runtime/pkg/manager"
"sigs.k8s.io/controller-runtime/pkg/reconcile"
"sigs.k8s.io/controller-runtime/pkg/source"
"sigs.k8s.io/controller-runtime/pkg/webhook/admission"
)

// DatabaseReconciler reconciles a Database object
type DatabaseReconciler struct {
client.Client
Scheme runtime.Scheme
}

// +kubebuilder:rbac:groups=example.com,resources=databases,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=example.com,resources=databases/status,verbs=get;update;patch
// +kubebuilder:rbac:groups=apps,resources=deployments,verbs=get;list;watch;create;update;patch;delete
// +kubebuilder:rbac:groups=core,resources=services,verbs=get;list;watch;create;update;patch;delete

func (r DatabaseReconciler) Reconcile(ctx context.Context, req reconcile.Request) (reconcile.Result, error) {
log := r.Log.WithValues("database", req.NamespacedName)

// Fetch the Database instance
instance := &v1.Database{}
err := r.Get(ctx, req.NamespacedName, instance)
if err != nil {
if errors.IsNotFound(err) {
// Request object not found, could have been deleted after reconcile request.
// Owned objects are automatically garbage collected. For additional cleanup logic use finalizers.
// Return and don't requeue
return reconcile.Result{}, nil
}
// Error reading the object - requeue the request.
return reconcile.Result{}, err
}

// Check if the number of replicas needs to be updated
if instance.Spec.Replicas != instance.Status.Replicas {
// Scale the deployment to match the desired number of replicas
err = r.scaleDeployment(ctx, instance)
if err != nil {
return reconcile.Result{}, err
}
instance.Status.Replicas = instance.Spec.Replicas
}

// Set the status of the database instance
instance.Status.Ready = true

// Update the database instance status
err = r.Status().Update(ctx, instance)
if err != nil {
return reconcile.Result{}, err
}

return reconcile.Result{}, nil
}

func (r DatabaseReconciler) scaleDeployment(ctx context.Context, instance v1.Database) error {
// Create or update the Deployment
deployment := &appsv1.Deployment{
ObjectMeta: metav1.ObjectMeta{
Name: fmt.Sprintf("database-%s", instance.Name),
Namespace: instance.Namespace,
},

Spec: appsv1.DeploymentSpec{
Replicas: &instance.Spec.Replicas,
Selector: &metav1.LabelSelector{
MatchLabels: map[string]string{
"app": "database",
},
},
Template: corev1.PodTemplateSpec{
ObjectMeta: metav1.ObjectMeta{
```

```
Labels: map[string]string{
"app": "database",
},
},
Spec: corev1.PodSpec{
Containers: []corev1.Container{
{
Name: "database",
Image: "postgres:latest", // Use your desired PostgreSQL image
Env: []corev1.EnvVar{
{
Name: "POSTGRES_USER",
Value: "postgres",
},
{
Name: "POSTGRES_PASSWORD",
Value: instance.Spec.Password,
},
{
Name: "POSTGRES_DB",
Value: instance.Name,
},
},
Ports: []corev1.ContainerPort{
{
ContainerPort: 5432,
Name: "database",
},
},
},
},
},
},
}

// Create or update the deployment
if err := r.CreateOrUpdate(ctx, deployment, func() error {
return r.Client.Get(ctx, types.NamespacedName{Name: deployment.Name, Namespace: deployment.Namespace}, deployment)
}); err != nil {
return err
}

// Create or update the service
service := &corev1.Service{
ObjectMeta: metav1.ObjectMeta{
Name: fmt.Sprintf("database-%s", instance.Name),
Namespace: instance.Namespace,
},
Spec: corev1.ServiceSpec{
Selector: map[string]string{
"app": "database",
},
Ports: []corev1.ServicePort{
{
Port: 5432,
TargetPort: intstr.FromInt(5432),
Name: "database",
},
},
},
}

// Create or update the service
if err := r.CreateOrUpdate(ctx, service, func() error {
return r.Client.Get(ctx, types.NamespacedName{Name: service.Name, Namespace: service.Namespace}, service)
}); err != nil {
return err
}

return nil
}

func (r DatabaseReconciler) SetupWithManager(mgr manager.Manager) error {
return ctrl.NewControllerManagedBy(mgr).
For(&v1.Database{}).
Owns(&appsv1.Deployment{}).
Owns(&corev1.Service{}).
Complete(r)
}
```

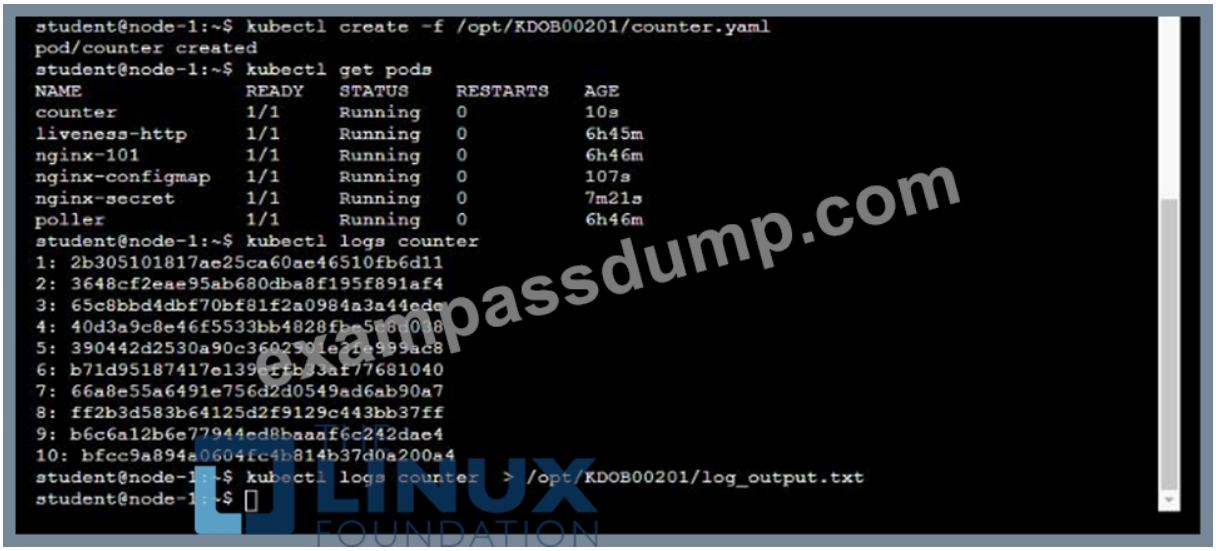
5. Build and Deploy the Operator: - Build the operator using the 'operator-sdk build' command: bash operator-sdk build example.com/database-operator:vO.O.I --local - Deploy the operator to your Kubernetes cluster: bash kubectl apply -f deploy/operator.yaml 6. Test the Operator: - Create a new 'database-example-com' resource:

- Apply the YAML file to your cluster: bash kubectl apply -f my-database.yaml - Verify that the operator creates a PostgreSQL database instance. - Test scaling the database by updating the 'spec.replicas' field of the 'database.example.com' resource. - Delete the 'database.example.com' resource and verify that the operator cleans up the database instance. This step-by-step guide demonstrates a basic example of a custom operator using the Operator Framework. You can Kustomize this operator further to handle more complex operations and integrate with other Kubernetes resources. ,

질문 # 99
Exhibit:

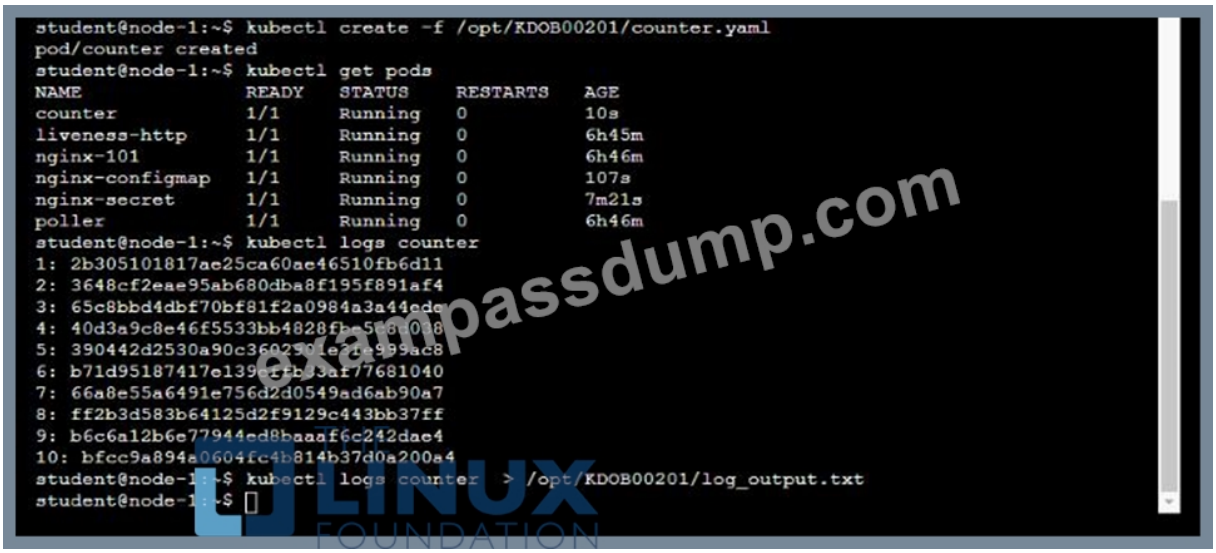
Context
You sometimes need to observe a pod's logs, and write those logs to a file for further analysis.
Task
Please complete the following;
* Deploy the counter pod to the cluster using the provided YAMLspec file at /opt/KDOB00201/counter.yaml
* Retrieve all currently available application logs from the running pod and store them in the file /opt/KDOB0020l/log_Output.txt, which has already been created

- A. Solution:




Readme
Web Terminal
THE LINUX FOUNDATION
student@node-1:~$ kubectl logs counter > /opt/KDOB00201/log_output.txt
student@node-1:~$ cat /opt/KDOB00201/log_output.txt
1: 2b305101817ae25ca60ae46510fb6d11
2: 3648cf2eae95ab680dba8f195f891af4
3: 65c8bbd4dbf70bf81f2a0984a3a44ede
4: 40d3a9c8e46f5533bb4828fbe5c8d038
5: 390442d2530a90c3602901e3fe999ac8
6: b71d95187417e139effb33af77681040
7: 66a8e55a6491e756d2d0549ad6ab90a7
8: ff2b3d583b64125d2f9129c443bb37ff
9: b6c6a12b6e77944ed8baaaf6c242dae4
10: bfcc9a894a0604fc4b814b37d0a200a4
11: 5493cd16a1790a5fb9512b0c9d4c5dd1
12: 03f169e93e6143438e6dfe4ecb3cc9ed
13: 764b37fe611373c42d0b47154041f6eb
14: 1a56fbe1896b0ee63941361662818 39e
15: ecc492eb17715de090c47345a98d98d3
16: 7974a6bec0fb44b6b8bbfc71aa3fbe74
17: 9ae01bef01748b12cc9f97a5f9f72dd
18: 23fb22ee34d4272e4c9e005f17745 1f
19: ec7e1a5d314da9a0ad45d53be 7deae
20: 0bccdd8ee02cd42029e8162 d1 197c
21: d6851ea43546216b95bcb81 997102
22: 7ed9a38ea8bf0d86206 1442af44
23: 29b8416ddc63dbfcb987 3c8198e9fe
24: 1f2062001df51a108ab25010f506716f
student@node-1:~$
THE LINUX FOUNDATION

- B. Solution:


student@node-1:~$ kubectl create -f /opt/KDOB00201/counter.yaml
pod/counter created
student@node-1:~$ kubectl get pods
NAME READY STATUS RESTARTS AGE
counter 1/1 Running 0 10s
liveness-http 1/1 Running 0 6h45m
nginx-101 1/1 Running 0 6h46m
nginx-configmap 1/1 Running 0 107s
nginx-secret 1/1 Running 0 7m21s
poller 1/1 Running 0 6h46m
student@node-1:~$ kubectl logs counter
1: 2b305101817ae25ca60ae46510fb6d11
2: 3648cf2eae95ab680dba8f195f891af4
3: 65c8bbd4dbf70bf81f2a0984a3a44ede
4: 40d3a9c8e46f5533bb4828fbe5c8d038
5: 390442d2530a90c3602901e3fe999ac8
6: b71d95187417e139effb33af77681040
7: 66a8e55a6491e756d2d0549ad6ab90a7
8: ff2b3d583b64125d2f9129c443bb37ff
9: b6c6a12b6e77944ed8baaaf6c242dae4
10: bfcc9a894a0604fc4b814b37d0a200a4
student@node-1:~$ kubectl logs counter > /opt/KDOB00201/log_output.txt
student@node-1:~$
THE LINUX FOUNDATION


student@node-1:~$ kubectl logs counter > /opt/KDOB00201/log_output.txt
student@node-1:~$ kubectl logs counter > /opt/KDOB00201/log_output.txt
student@node-1:~$ cat /opt/KDOB00201/log_output.txt

정답：B

질문 # 100
You have a microservice application that relies on a Redis cacne for data retrieval. Design a multi-container Pod that incorporates a Redis sidecar container to provide local caching within the Pod. Ensure that the main application container can access the Redis sidecar container within the same Pod Namespace Without needing to communicate with an external Redis cluster.

정답：

설명：
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Define the Pod YAML: Create a Pod YAML file that includes both the main application container and the Redis sidecar container.

```
kind: Pod
metadata:
  name: my-app-pod
spec:
  containers:
  - name: my-app
    image: my-app-image:latest
    ports:
    - containerPort: 8080
    env:
    - name: REDIS_HOST
      value: redis
  - name: redis
    image: redis:latest
    ports:
    - containerPort: 6379
```

2. Configure Environment Variables: Set an environment variable 'REDIS HOST within the main application container to point to the Redis sidecar containers hostname- In Kubernetes, containers within the same Pod can communicate with each other using their container names. 3. Connect Application to Redis: Modifry' the application code to connect to the Redis instance using the 'REDIS HOST environment variable. For example, using a Python application with the 'redis-py' library: python import redis r = redis-

Redis(host=os.environ.get('REDlS_HOST'), port=6379) # Perform Redis operations (e.g., r.set('key', 'value')) 4. Deploy the Pod: Apply the Pod YAML using 'kubectl apply -f my-app-pod.yamr 5. Verify Connectivity: Check the logs of the main application container to ensure it's successfully connecting to the Redis sidecar container Note: This approach provides local caching within the Pod, reducing external network calls and improving performance. It's important to consider potential data consistency issues if multiple Pods share the same Redis instance.

질문 # 101
You are running a web application with multiple services exposed via Kubernetes Ingress. The application has two distinct environments: 'staging' and 'production' , each with its own set of services and domain names. You need to configure Ingress rules to route traffic to the appropriate services based on the requested hostname and environment. For example, requests to 'staging.example.com' should be directed to the staging environment, while requests to 'example.com' should go to the production environment. Implement this configuration using Ingress rules.

정답：

설명：
See the solution below with Step by Step Explanation.
Explanation:
Solution (Step by Step) :
1. Create a Service for Each Environment:
- Define services for both 'staging' and 'production' environments, ensuring that the services for each environment are named appropriately. For example, 'staging-service' and .

2. Create an Ingress Resource: - Define an Ingress resource that maps the hostnames to the corresponding services.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-app-ingress
spec:
  rules:
  - host: staging.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: staging-service
            port:
              number: 80
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: production-service
            port:
              number: 80
```


3. Apply the Configuration: - Apply the service and ingress definitions using 'kubectl apply -f services.yaml' and 'kubectl apply -f ingress.yaml' respectively. 4. Test the Configuration: - Access 'staging.example.com' and 'example.com' in your browser to ensure that the traffic is directed to the correct services and environments. ,

질문 # 102

.....

www.exampassdump.com 〉에서□ CKAD □를 검색하고 무료로 다운로드하세요CKAD시험패스 인증덤프문제

- jessempzs009294.cosmicwiki.com, scrapbookmarket.com, cormacdbfi790550.blogspothub.com, majaujnr784968.newsbloger.com, andrewgubs652359.goabroadblog.com, aliviapfyj016679.ttblogs.com, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, myportal.utt.edu.tt, flynnualr021285.blog-mall.com, fanniefmnd102509.luwebs.com, jasonewof120428.blgwiki.com, Disposable vapes

2026 ExamPassdump 최신 CKAD PDF 버전 시험 문제집과 CKAD 시험 문제 및 답변 무료 공유: https://drive.google.com/open?id=1_jleR_DQekGqlGJJk7mtBPZMBf31Mkml