CKAD Practice Exam: Services & Networking Solution Guide

Intermediate 20m 5/5 Bookmark

Check 1: Create and Configure a Basic Pod Potential Solution

```
# Create a Pod in the red Namespace with the following configuration: The
Pod is named basic, the Pod uses the nginx:stable-alpine-perl image for
its only container, restart the Pod only OnFailure, and ensure port 80 is
open to TCP traffic
kubectl run -n red --image=nginx:stable-alpine-perl --restart=OnFailure -
-port=80 basic
```

Commentary

The above solution uses the `run` command to create the Pod. The options for configuring the pod can be seen by using tab completion (the host runs `source<(kubectl completion bash)` automatically to enable completions but the command is always available in the kubectl Cheat Sheet) or consulting the help output of `kubectl run --help`. The help output also has examples that can be combined to construct the required command. Ports default to referring to the TCP protocol.

Alternatively, you can achieve the result using a Pod manifest file. To generate a manifest file as a starting point, you could:

- Copy in a manifest from the official documentation. There are many examples to choose from. Try to choose one that is going to have all the fields you need.
- Use a dry-run, e.g. `kubectl -n cre run --image=nginx:stable-alpine-perl basic --dry-run=client -o yaml` The `--`
i

This option can be quite tedious for Pods due to their being hundreds of lines of output.

For the manifest file options you would need to edit the file using a command-line editor, such as `vim`. In the real exam, you could also use the exam's notepad function to edit the manifest before pasting it into a file in the exam terminal (for example entering vim manifest.yaml to open an empty file, pressing *i* to enter insert mode, pasting the text, pressing escape, and finally entering *:wq* to write the file and quit). Practicing with `vim` is highly recommended before taking the exam.

To know what fields to add to the manifest, you could look at the documentation examples, manifests of existing pods, or output from `kubectl explain` (for example `kubectl explain pod.spec`). Using manifest files is necessary sometimes - but for basic tasks, adding options to the `kubectl run` command with the help of tab completions will probably be the fastest, which is very important in the exam.

Content to review

- Introduction to Kubernetes - Pods
- Kubernetes Pod Design for Application Developers: Definition Basics - Reviewing Pod Definition Basics

Suggested documentation bookmark(s)

- Configure Pods and Containers
- kubectl Cheat Sheet

Check 2: Expose Pod Potential Solution

```
# Expose a Pod in the red Namespace with the following configuration: The
Service name is cloudacademy, the Service port is 8080, the Target port
is 8(
kube(                                                                        -
targe
```

options for exposing a pod can be seen by using tab completion (the host runs `source<(kubectl completion bash)` automatically to enable completions but the command is always available in the kubectl Cheat Sheet) or consulting the help output of `kubectl expose --help`. The help output also has examples that can be combined to construct the required command.

Alternatively, you can achieve the result using a Service manifest file. To generate a manifest file as a starting point, you could:

- Copy a service manifest from the official documentation.
- Use a dry-run, e.g. `kubectl expose pod basic -n red --name=cloudacademy-svc --port=8080 --target-port=80 --dry-run=client -o yaml`

For the manifest file options you would need to edit the file using a command-line editor, such as `vim`. In the real exam, you could also use the exam's notepad function to edit the manifest before pasting it into a file in the exam terminal (for example entering vim manifest.yaml to open an empty file, pressing *i* to enter insert mode, pasting the text, pressing escape, and finally entering *:wq* to write the file and quit). Practicing with `vim` is highly recommended before taking the exam.

To know what fields to add to the manifest, you could look at the documentation examples, or manifests of existing services. Using manifest files is necessary sometimes - but for basic tasks, adding options to the `kubectl expose` command with the help of tab completions will probably be the fastest, which is very important in the exam.

Content to review

- Introduction to Kubernetes - Services

Suggested documentation bookmark(s)

- S

Check 3: Expose Deployment Potential Solution

```
# Expose a deployment as a NodePort based service using the following
settings: The Service name is cloudforce-svc, the Service type is
NodePort, the Service port is 80, and the NodePort is 32080
kubectl expose deployment -n ca1 cloudforce --name=cloudforce-svc --
port=80 --type=NodePort
kubectl patch -n ca1 svc cloudforce-svc --patch '{"spec": {"ports":
[{"port": 80, "nodePort": 32080}]}}'
```

Commentary

The above solution involve 2 steps:

Step 1: Create a NodePort based Service

```
kubectl expose deployment -n ca1 cloudforce --name=cloudforce-svc --
port=80 --type=NodePort
```

This first step uses the `expose` command to first create a service for the Deployment. The options for exposing a deployment can be seen by using tab completion (the host runs `source<(kubectl completion bash)` automatically to enable completions but the command is always available in the kubectl Cheat Sheet) or consulting the help output of `kubectl expose --help`. The help output also has examples that can be combined to construct the required command.

Alternatively, you can achieve the result using a Service manifest file. To generate a manifest file as a starting point, you could:

- Copy a service manifest from the official documentation
- Use a dry-run, e.g. `kubectl expose deployment basic -n cal1 --name=cloudacademy-svc --port=8080 --target-port=80 --dry-run=client -o yaml`

function to edit the manifest before pasting it into a file in the exam terminal (for example entering vim manifest.yaml to open an empty file, pressing *i* to enter insert mode, pasting the text, pressing escape, and finally entering *:wq* to write the file and quit). Practicing with `vim` is highly recommended before taking the exam.

To know what fields to add to the manifest, you could look at the documentation examples, or manifests of existing services. Using manifest files is necessary sometimes - but for basic tasks, adding options to the `kubectl expose` command with the help of tab completions will probably be the fastest, which is very important in the exam.

Step2: Patch the NodePort based Service

```
kubectl patch -n ca1 svc cloudforce-svc --patch '{"spec": {"ports":
[{"port": 80, "nodePort": 32080}]}}'
```

This second step uses the `patch` command to override the auto generated NodePort. When using the `expose` command to create a NodePort based Service, Kubernetes will randomly choose and assign a NodePort for you. The quickest way to override this value is to patch it.

Content to review

- Introduction to Kubernetes - Services

Suggested documentation bookmark(s)

- Services
- kubectl Cheat Sheet

Check 4: Fix Networking Issue Potential Solution

```
# Exa                                                             e
sele
kube
kubectl describe svc -n skynet t1-svc
kubectl edit svc -n skynet t2-svc
# Check the t2-svc Service Endpoints are now correctly registered
kubectl get ep -n skynet t2-svc
# Curl the updated t2-svc Service and save the HTTP response
kubectl run client -n skynet --image=appropriate/curl -it --rm --
restart=Never -- curl http://t2-svc:8080 > /home/ubuntu/svc-output.txt
```

Commentary

The above solution involves 3 main steps:

Step 1: Determine the misconfiguration, examine the Deployment, Service, and Endpoints

```
kubectl describe deployments.apps -n skynet t2
kubectl describe svc -n skynet t2-svc
kubectl get ep -n skynet t2-svc
```

This first 2 commands both use the `describe` command to output the current configuration for both the Deployment and the Service. The Service selector labels should match the same labels declared and used within the Deployment - currently they don't, and this means that the Service is unable to forward traffic downstream as there are no valid Endpoints currently registered.

Step 2: Update the Service - change the label selector to be `app: t2`

```
kubectl edit svc -n skynet t2-svc
```

The second step uses the `edit` command to perform the selector label correction within the `t2-svc` Service.

Check that the Endpoints are now correctly registered:

```
kubectl get ep -n skynet t2-svc
```

Step 3 the
HTTP

The third and final step requires you to copy and paste the provided command, executing it to call the updated `t2-svc` Service.

Content to review

- kubectl edit

Suggested documentation bookmark(s)

- Services
- kubectl Cheat Sheet

Check 5: Secure Pod Networking Potential Solution

```
# Confirm that pod2 traffic sent to pod1 is as stated currently blocked
pod1IP=$(kubectl get pods pod1 -n sec1 -o jsonpath='{.status.podIP}')
kubectl -n sec1 exec -it pod2 -- ping $pod1IP
# Examine the pod labels for pod1 (receiver) and pod2 (sender)
kubectl get pods -n sec1 --show-labels
# Examine the existing NetworkPolicy
kubectl -n sec1 describe netpol netpol1
# Edit and update the NetworkPolicy with correct ingress pod selector
label
kubectl edit -n sec1 netpol netpol1
# Confirm that pod2 can now send traffic to pod1
pod1IP=$(kubectl get pods pod1 -n sec1 -o jsonpath='{.status.podIP}')
kubectl -n sec1 exec -it pod2 -- ping $pod1IP
```

Commentary

The above solution involves 4 main steps:

Step 1: Confirm that `pod2` traffic sent to `pod1` is as stated currently blocked

```
pod1IP=$(kubectl get pods pod1 -n sec1 -o jsonpath='{.status.podIP}')
kubectl -n sec1 exec -it pod2 -- ping $pod1IP
```

This st hen
exec ir ıld fail

which ones are being used

```
kubectl get pods -n sec1 --show-labels
kubectl -n sec1 describe netpol netpol1
```

This step highlights that there is a mismatch in terms of the ingress labels being used in the NetworkPolicy.

Step 3. Edit and update the NetworkPolicy with the correct ingress pod selector label

```
kubectl edit -n sec1 netpol netpol1
```

This step uses the `edit` command to perform the ingress pod selector label correction within the network policy

Step 4: Confirm that `pod2` traffic sent to `pod1` is now allowed

```
pod1IP=$(kubectl get pods pod1 -n sec1 -o jsonpath='{.status.podIP}')
kubectl -n sec1 exec -it pod2 -- ping $pod1IP
```

This step first grabs the pod IP address assigned by the cluster to `pod1`. We then exec into `pod2` and perform a ping command to `pod1`. The ping requests should now work as traffic is permitted from `pod2` to `pod1`.

Content to review

- kubectl edit
- kubectl exec

Suggested documentation bookmark(s)

- Network Policies
- kubectl Cheat Sheet

Skip to content Press option + Q to open this menu

Mark as completed

Did you like this Resource? Report an issue

About the author

Cloud Academy

Instructor

Students	Labs	Courses	Learning paths
2,626	83	9	18

Digital skills are built at the intersection of knowledge, experience, and context. The fundamental building blocks of the training templates in our Library meet teams wherever they are along the cloud maturity curve, imparting the knowledge and experience needed to take them to the next level. Our training platform is intuitive and scalable. Most importantly, all training is easily customizable, which enables organizations to provide context and guidance for teams of any size. Teams leveraging Cloud Academy hit the ground running.

Covered topics

Deployment Compute DevOps Kubernetes

About Circus Street

COMMUNITY

Join Discord Channel

HELP

Help Center