CKAD Practice Exam: Pod Design Solution Guide

Intermediate 20m Bookmark

Check 1: Create and Manage Deployments Potential Solution

```
# Create a deployment named webapp in the zap namespace. Use the
nginx:1.17.8 image and set the number of replicas initially to 2
kubectl -n zap create deployment --image=nginx:1.17.8 --replicas=2 webapp
# Next, scale the current deployment up from 2 to 4.
kubectl -n zap scale deployment --replicas=4 webapp
# Finally, update the deployment to use the newer nginx:1.19.0 image.
kubectl -n zap set image deployment webapp nginx=nginx:1.19.0
```

Commentary

The above solution uses all imperative commands (`create`, `scale`, `set`) to accomplish the task. If you forget how to use the `create`, `scale`, or `set` commands, they have useful examples built into their help output, e.g. `kubectl scale --help`. The help output uses the convention of identifying resources with resource_kind/resource_name, e.g. `deployment/webapp`. This is equivalent to the commands above that have separated the kind and the name. The kubectl cheat sheet also includes examples of all the `create`, `scale`, and `set` commands needed to complete this task.

The `create` command is well worth knowing because of its versatility in creating a variety of resources. You may also use it in `dry-run` mode to generate manifest files as in

```
kubectl -n zap create deployment --image=nginx:1.17.8 --replicas=2 webapp
--dry
```

accomplish the scaling and changing image. Similarly, the following command can combine file editing and `apply` into one after initially creating the deployment:

```
kubectl -n zap edit deployment webapp
```

When editing manifest files, you can use `kubectl explain` to get more information about any available fields. In this task, the `replicas:` and `image:` fields are provided in `--dry-run` output and are self-explanatory.

Content to review

- Kubernetes Pod Design for Application Developers: Deployments - Managing Pods with Deployments

Suggested documentation bookmark(s)

- kubectl Cheat Sheet

Check 2: Create Pod Labels Potential Solution

```
# Add an additional label app=cloudacademy to all pods currently running
in the gzz namespace that have the label env=prod
kubectl -n gzz label pods --selector env=prod app=cloudacademy
```

Commentary

The potential solution combines a selector to select only Pods with the `env=prod` label and the `kubectl label` command to apply the new `app=cloudacademy` label. This is the most succint solution and checking the `kubectl label --help` output could help you arrive at it. The `--selector` (or `-l`) option is available for a variety of `kubectl` commands to make filterir bectl Cheat

```
kubectl -n gzz get pods --show-labels
```

and using `kubectl edit` to edit the `metadata.labels` mapping. This approach would use up much more valuable exam time however.

Content to review

- Kubernetes Pod Design for Application Developers: Labels, Selectors, and Annotations - Working With Pod Labels, Selectors, and Annotations

Suggested documentation bookmark(s)

- kubectl Cheat Sheet

Check 3: Rollback Deployment Potential Solution

```
# The nginx container running within the cloudforce deployment in the fre
namespace needs to be updated to use the nginx:1.19.0-perl image. Perform
this deployment update and ensure that the command used to perform it is
recorded in the tracked rollout history.
kubectl -n fre set image deployment cloudforce nginx=nginx:1.19.0-perl
# To manage the deployment history, use the annotate command to create a
message.
kubectl -n fre annotate deployment cloudforce kubernetes.io/change-
cause="set image to nginx:1.19.0-perl" --overwrite=true
```

Commentary

The `kubectl set image` command image command above is purpose-built for this task. The `nginx=nginx:1.19.0-perl` option specifies the Pod container (`nginx`) being set to the given image (`nginx:1.19.0-perl`). The annotate command is required to store the change message and make it available in the rollout history (view it with `kubectl -n fre rollout history deployment cloudforce`). The help output (`kubectl annotate --help`) provid es the conve

Alternatively, you could use

```
kubectl -n fre edit deployment cloudforce
```

and edit the spec.template.spec.containers.image value.

Content to review

- Kubernetes Pod Design for Application Developers: Deployments - Managing Pods with Deployments

Suggested documentation bookmark(s)

- Deployments
- kubectl Cheat Sheet

Check 4: Configure Pod AutoScaling Potential Solution

```
# A deployment named eclipse has been created in the xx1 namespace. This
deployment currently consists of 2 replicas. Configure this deployment to
autoscale based on CPU utilisation. The autoscaling should be set for a
minimum of 2, maximum of 4, and CPU usage of 65%.
kubectl -n xx1 autoscale deployment --min=2 --max=4 --cpu-percent=65
eclipse
```

Commentary

The autoscale command may not be used frequently and you may have to pick it out from the output of `kubectl` or use tab completion to list all the available commands after typing kubectl. From there `kubectl autoscale --help` provides examples that only need a bit of massaging to perform the task. The kubectl Cheat Sheet also includes an autoscale example. Autoscaling creates a HorizontalPodAutoscaler resource that you can view with `kubectl -n xx1 get h`

Suggested documentation bookmark(s)

- Horizontal Pod Autoscaler Walkthrough
- kubectl Cheat Sheet

Check 5: Create CronJob Potential Solution

```
# Create a cronjob named matrix in the saas namespace. Use the
radial/busyboxplus:curl image and set the schedule to */10 * * * *. The
job should run the following command: curl www.google.com
kubectl -n saas create cronjob --image=radial/busyboxplus:curl --
schedule='*/10 * * * *' matrix -- curl www.google.com
```

Commentary

The create command supports CronJobs and options are available for setting the image and schedule making it an efficient choice for this task. Running `kubectl create cronjob --help` provides a similar example to illustrate how to add the command to run after `--`. The kubectl Cheat Sheet also has a similar example.

For this task the schedule is provided in cron syntax already. If a task requires you to construct the syntax for the schedule, e.g. "Create a CronJob that runs every hour", the cron schedule syntax is explained in the official documentation.

If you prefer working with manifest files, the CronJob documentation has a sample you can customize and apply with `kubectl apply -f manifest.yaml`.

Content to review

- Kubernetes Pod Design for Application Developers: Jobs and CronJobs - U

Skip to content Press option + Q to open this menu

Sugge

Check 6: Filter and Sort Pods Potential Solution

```
# Get a list of all pod names running in the rep namespace which have
their colour label set to either orange, red, or yellow. The returned pod
name list should contain only the pod names and nothing else. The pods
names should be ordered by the cluster IP address assigned to each pod.
The resulting pod name list should be saved out to the file
/home/ubuntu/pod001
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP -o jsonpath='{range .items[*]}{.metadata.name}
{"\n"}{end}' > /home/ubuntu/pod001
```

Commentary

To arrive at the final answer you can build it in steps. Starting with the `--selector` query along with `--show-labels` to confirm only the desired pods are selected:

```
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
show-labels
```

See `kubectl get --help` if you need to review the options available for `get`. The labels and selectors documentations has examples of the operations supported by selectors. In this case using the set operator `in` is best to choose from a set of allowed colours.

Next you add an option to sort by (`--sort-by`) the Pod IPs. To sort, you need to specify a JSONPath expression. The syntax can be tricky. The kubectl Cheat Sheet has several examples of sorting but none to sort by IP address. Don't get caught up in the semantics of what it means to sort by an IP address. Use the sorting that is built into `kubectl`, which in this case will treat the addresses as strings (lexicographic order). To construct the JSONPath expression for a Pod's IP, you have a couple options. First is to output an example Pod in YAML format:

```
kube
```

Alternatively, you can traverse the kubectl explain output starting with:

```
kubectl explain pod
```

and adding .status (`kubectl explain pod.status`) to find the key name for the IP address. The IP is determined by the cluster and not something you declare in the Pod's spec, so it will be in the Pod's status object. The `explain` field syntax for the Pod IP (`pod.status.podIP`) is a JSONPath expression that can be used for --sort-by except you omit the type (`pod`). in the explain argument.

With the JSONPath expression ready, you can add the `--sort-by` option and alson include `-o wide` to output the IP addresses for confirmation:

```
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP -o wide
```

You need to output only the Pod names for the task. This can be accomplished using the JSONPath output option with an appropriate JSONPath to the Pod name (resource names are always available in the `metadata` object):

```
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP -o jsonpath='{.items[*].metadata.name}'
```

The `.items[*]` is needed for the output because the `get` response is returned in an object with an `items` array containing all of the Pods (use `-o json` to see the returned object). The kubectl Cheat Sheet has `-o jsonpath` examples with `.items[*]` to reference during the exam. The Pod names are output on a single line, however, to massage the data into the desired format it is possible to use a range function as shown in the JSONPath support documentation:

```
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP -o jsonpath='{range .items[*]}{.metadata.name}
{"\n"}{end}'
```

```
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP -o custom-columns="NAME:.metadata.name"
```

It may be easier to use linux commands during the exam unless you are very familiar with JSONPath or custom columns to get the desired result, however. Some examples are:

```
# cut the first column of the earlier table output and remove the column
heading
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP | cut -d' ' -f1 | tail +2
# grep the pod name and only output the matching characters
kubectl -n rep get pods --selector 'colour in (orange,red,yellow)' --
sort-by=.status.podIP | grep -o -e "pod[0-9]*"
```

Or you could manually construct the output in the exam's notepad and paste it into the desired file using `vim`. Do whatever will get the desired result in the least amount of time.

Content to review

- Kubernetes Pod Design for Application Developers: Labels, Selectors, and Annotations - Working With Pod Labels, Selectors, and Annotations

Suggested documentation bookmark(s)

- Labels and Selectors
- JSONPath Support
- kubectl Cheat Sheet

Mark it or miss it!
Make sure to mark this content as completed; otherwise, it will not be displayed as such.

Mark as completed

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

HELP

Help Center