

In above screen shot for curl – curl <http://192.168.1.2:30008> where 192.168.1.2 is IP of Node

kubectl create service clusterip redis --tcp=6379:6379 --dry-run=client -o yaml

kubectl create service nodeport nginx --tcp=80:80 --node-port=30080 --dry-run=client -o yaml

Both the above commands have their own challenges. While one of it cannot accept a selector the other cannot accept a node port. I would recommend going with the `kubectl expose` command. If you need to specify a node port, generate a definition file using the same command and manually input the nodeport before creating the service

When we use –expose, it will also create service..

The major difference between sudo -i and sudo -s is:

* sudo -i gives you the root environment, i.e. your ~/.bashrc is ignored.
* sudo -s gives you the user's environment, so your ~/.bashrc is respected.

kubectl get po nginx --v=7

kubectl get po nginx --v=8

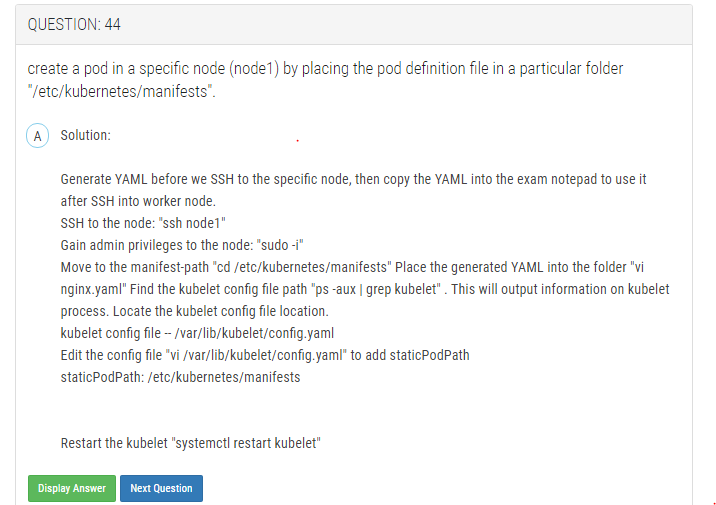
kubectl get po nginx --v=9

>> kubectl logs hello-jobs\*

kubectl get pv --sort-by=spec.capacity.storage > /opt/pvlist.txt

kubectl create job nodeversion --image=node -- node -v

24 svc DNS



<https://prabhatsharma.in/blog/cka-practice-test-1/>

Here is the first set:

1. Create a node that has an SSD and label it as such.
   1. Create a pod that is only scheduled on SSD nodes.
2. Create 2 pod definitions: the second pod should be scheduled to run anywhere the first pod is running - 2nd pod runs alongside the first pod.
3. Create a deployment running nginx version 1.12.2 that will run in 2 pods
   1. Scale this to 4 pods.
   2. Scale it back to 2 pods.
   3. Upgrade this to 1.13.8
   4. Check the status of the upgrade
   5. How do you do this in a way that you can see the history of what happened?
   6. Undo the upgrade
   7. Expose the service on port 80
4. Create a pod that uses a scratch disk.
   1. Change the pod to mount a path on the host.
5. Taint a node and run a Jenkins Pod on that specified node only.
6. Create a pod that has a liveness check
7. Use the utility nslookup to look up the DNS records of the service and pod.
8. Find which Pod is taking max CPU
9. List all PersistentVolumes sorted by their name
10. Create a daemon set
    1. Change the update strategy to do a rolling update but delaying 30 seconds between pod updates
11. Create a static pod
12. Create a busybox container without a manifest. Then edit the manifest.
13. Create a pod that uses secrets
    1. Create a secret
    2. Pull secrets from environment variables
    3. Pull secrets from a volume
    4. Dump the secrets out via kubectl to show it worked
14. Create a job that runs every 3 minutes and prints out the current time.
15. Create a job that runs 20 times, 5 containers at a time, and prints “Hello parallel world”
16. Create a horizontal autoscaling group that starts with 2 pods and scales when CPU usage is over 50%.
17. Create a custom resource definition - CRD
    1. Display it in the API with curl
18. Create a networking policy such that only pods with the label access=granted can talk to it.
    1. Create a nginx pod and attach this policy to it.
    2. Create a busybox pod and attempt to talk to nginx - should be blocked
    3. Attach the label to busybox and try again - should be allowed
19. Create a service that references an externalname - <https://api.github.com/users/prabhatsharma>
    1. Test that this works from another pod
20. Create a pod that runs all processes as user 1000.
21. Create a namespace
    1. Run a pod in the new namespace
    2. Put memory limits on the namespace
    3. Limit pods to 2 persistent volumes in this namespace
22. Write an ingress rule that redirects calls to /foo to one service and to /bar to another
23. Write a service that exposes nginx on a nodeport
    1. Change it to use a cluster port
    2. Scale the service
    3. Change it to use an external IP
    4. Change it to use a load balancer
24. Deploy nginx with 3 replicas and then expose a port
25. Use port forwarding to talk to a specific port

2nd practice test for CKA exam preparation. First is at [CKA exam practice test 1](https://prabhatsharma.in/blog/cka-practice-test-1)

1. Display all the pods sorted by start time
2. Create a pod that uses secrets
   1. Create a secret
   2. Pull secrets from environment variables
   3. Pull secrets from a volume
   4. Dump the secrets out via kubectl to show it worked
3. Create a job that runs every 3 minutes and prints out the current time.
4. Create a job that runs 20 times, 5 containers at a time, and prints “Hello parallel world”
5. Create a horizontal autoscaling group that starts with 2 pods and scales when CPU usage is over 50%.
6. Create a custom resource definition - CRD
   1. Display it in the API with curl
7. Create a networking policy such that only pods with the label access=granted can talk to it.
   1. Create a nginx pod and attach this policy to it.
   2. Create a busybox pod and attempt to talk to nginx - should be blocked
   3. Attach the label to busybox and try again - should be allowed
8. Create a service that references an externalname.
   1. Test that this works from another pod
9. Create a pod that runs all processes as user 1000.
10. Create a namespace
    1. Run a pod in the new namespace
    2. Put memory limits on the namespace
    3. Limit pods to 2 persistent volumes in this namespace
11. Write an ingress rule that redirects calls to /foo to one service and to /bar to another
12. Write a service that exposes nginx on a nodeport
    1. Change it to use a cluster port
    2. Scale the service
    3. Change it to use an external IP
    4. Change it to use a load balancer
13. Deploy nginx with 3 replicas and then expose a port
    1. Use port forwarding to talk to a specific port
14. Make an API call using CURL and proper certs
15. Join a new node to cluster
16. Rotate certificates
17. Taint a node and un-taint it
18. Restart kubelet
19. Configure the cluster to use 8.8.8.8 and 8.8.4.4 as upstream DNS servers.
20. Create a pod with nginx and place a file using an init container that creates a simple index.html file with content - “created by init container”
21. You have a Container with a volume mount. Add an init container that creates an empty file in the volume. (the only trick is to mount the volume to 1. init-container as well)
22. Backup an etcd cluster
23. List the members of an etcd cluster
24. Find the health of etcd

Feel free to add any questions in the comment section that you have created or have found in the wild.

<https://www.dumpsbase.com/freedumps/certified-kubernetes-administrator-certification-cka-exam-dumps.html>

**Check CKA Certified Kubernetes Administrator Free Dumps**

Top of Form

**1. Create a namespace called 'development' and a pod with image nginx called nginx on this namespace.**

Solution: kubectl create namespace development kubectl run nginx --image=nginx --restart=Never -n development

**2. Create a nginx pod with label env=test in engineering namespace See the solution below.**

Solution: kubectl run nginx --image=nginx --restart=Never --labels=env=test --namespace=engineering --dry- run -o yaml > nginx-pod.yaml kubectl run nginx --image=nginx --restart=Never --labels=env=test --namespace=engineering --dry- run -o yaml | kubectl create -n engineering -f - YAML File: apiVersion: v1 kind: Pod metadata: name: nginx namespace: engineering labels: env: test spec: containers: - name: nginx image: nginx imagePullPolicy: IfNotPresent restartPolicy: Never kubectl create -f nginx-pod.yaml

**3. Get list of all pods in all namespaces and write it to file "/opt/pods-list.yaml"**

Solution: kubectl get po -all-namespaces > /opt/pods-list.yaml

**4. Create a pod with image nginx called nginx and allow traffic on port 80**

Solution: kubectl run nginx --image=nginx --restart=Never --port=80

**5. Create a busybox pod that runs the command "env" and save the output to "envpod" file**

Solution: kubectl run busybox --image=busybox --restart=Never --rm -it -- env > envpod.yaml

**6. List pod logs named "frontend" and search for the pattern "started" and write it to a file "/opt/error- logs"**

Solution: Kubectl logs frontend | grep -i "started" > /opt/error-logs

**7. Create a pod that echo "hello world" and then exists. Have the pod deleted automatically when it's completed**

Solution: kubectl run busybox --image=busybox -it --rm --restart=Never -- /bin/sh -c 'echo hello world' kubectl get po # You shouldn't see pod with the name "busybox"

**8. Create a pod with environment variables as var1=value1.Check the environment variable in pod**

Solution: kubectl run nginx --image=nginx --restart=Never --env=var1=value1 # then kubectl exec -it nginx -- env # or kubectl exec -it nginx -- sh -c 'echo $var1' # or kubectl describe po nginx | grep value1

**9. Get list of all the pods showing name and namespace with a jsonpath expression.**

Solution: kubectl get pods -o=jsonpath="{.items[\*]['metadata.name' , 'metadata.namespace']}"

**10. Check the image version in pod without the describe command**

Solution: kubectl get po nginx -o jsonpath='{.spec.containers[].image}{"  
"}'

**11. List the nginx pod with custom columns POD\_NAME and POD\_STATUS**

Solution: kubectl get po -o=custom-columns="POD\_NAME:.metadata.name, POD\_STATUS:.status.containerStatuses[].state"

**12. List all the pods sorted by name**

Solution: kubect1 get pods --sort-by=.metadata.name

**13. Create a pod that having 3 containers in it? (Multi-Container)**

Solution: image=nginx, image=redis, image=consul Name nginx container as "nginx-container" Name redis container as "redis-container" Name consul container as "consul-container" Create a pod manifest file for a container and append container section for rest of the images kubectl run multi-container --generator=run-pod/v1 --image=nginx -- dry-run -o yaml > multi- container.yaml # then vim multi-container.yaml apiVersion: v1 kind: Pod metadata: labels: run: multi-container name: multi-container spec: containers: - image: nginx name: nginx-container - image: redis name: redis-container - image: consul name: consul-container restartPolicy: Always

**14. Create 2 nginx image pods in which one of them is labelled with env=prod and another one labelled with env=dev and verify the same.**

Solution: kubectl run --generator=run-pod/v1 --image=nginx -- labels=env=prod nginx-prod --dry-run -o yaml > nginx-prodpod.yaml Now, edit nginx-prod-pod.yaml file and remove entries like "creationTimestamp: null" "dnsPolicy: ClusterFirst" vim nginx-prod-pod.yaml apiVersion: v1 kind: Pod metadata: labels: env: prod name: nginx-prod spec: containers: - image: nginx name: nginx-prod restartPolicy: Always # kubectl create -f nginx-prod-pod.yaml kubectl run --generator=run-pod/v1 --image=nginx -- labels=env=dev nginx-dev --dry-run -o yaml > nginx-dev-pod.yaml apiVersion: v1 kind: Pod metadata: labels: env: dev name: nginx-dev spec: containers: - image: nginx name: nginx-dev restartPolicy: Always # kubectl create -f nginx-prod-dev.yaml Verify : kubectl get po --show-labels kubectl get po -l env=prod kubectl get po -l env=dev

**15. Get IP address of the pod - "nginx-dev"**

Solution: Kubect1 get po -o wide Using JsonPath kubect1 get pods -o=jsonpath='{range .items[\*]}{.metadata.name}{"t"}{.status.podIP}{"  
"}{end}'

**16. Print pod name and start time to "/opt/pod-status" file**

Solution: kubect1 get pods -o=jsonpath='{range .items[\*]}{.metadata.name}{"t"}{.status.podIP}{"  
"}{end}'

**17. Check the Image version of nginx-dev pod using jsonpath**

Solution: kubect1 get po nginx-dev -o jsonpath='{.spec.containers[].image}{"  
"}'

**18. Create a busybox pod and add "sleep 3600" command**

Solution: kubectl run busybox --image=busybox --restart=Never -- /bin/sh -c "sleep 3600"

**19. Create an nginx pod and list the pod with different levels of verbosity**

Solution: // create a pod kubectl run nginx --image=nginx --restart=Never --port=80 // List the pod with different verbosity kubectl get po nginx --v=7 kubectl get po nginx --v=8 kubectl get po nginx --v=9

**20. List the nginx pod with custom columns POD\_NAME and POD\_STATUS**

Solution: kubectl get po -o=custom-columns="POD\_NAME:.metadata.name, POD\_STATUS:.status.containerStatuses[].state"

**21. List all the pods sorted by name**

Solution: kubectl get pods --sort-by=.metadata.name

**22. List all the pods sorted by created timestamp**

Solution: kubect1 get pods--sort-by=.metadata.creationTimestamp

**23. List all the pods showing name and namespace with a json path expression**

Solution: kubectl get pods -o=jsonpath="{.items[\*]['metadata.name', 'metadata.namespace']}"

**24. List "nginx-dev" and "nginx-prod" pod and delete those pods**

Solution: kubect1 get pods -o wide kubectl delete po "nginx-dev" kubectl delete po "nginx-prod"

**25. Delete the pod without any delay (force delete)**

Solution: Kubect1 delete po "POD-NAME" --grace-period=0 -force

**26. Create a redis pod and expose it on port 6379**

Solution: kubectl run redis --image=redis --restart=Never --port=6379 YAML File : apiVersion: v1 kind: Pod metadata: labels: run: redis name: redis spec: containers: - image: redis name: redis ports: - containerPort: 6379 Rt restartPolicy: Always

**27. Create the nginx pod with version 1.17.4 and expose it on port 80**

Solution: kubectl run nginx --image=nginx:1.17.4 --restart=Never -- port=80

**28. Change the Image version to 1.15-alpine for the pod you just created and verify the image version is updated.**

Solution: Kubect1 set image pod/nginx nginx=nginx:1.15-alpine kubect1 describe po nginx // another way it will open vi editor and change the version kubeclt edit po nginx kubect1 describe po nginx

**29. Change the Image version back to 1.17.1 for the pod you just updated and observe the changes**

Solution: kubectl set image pod/nginx nginx=nginx:1.17.1 kubectl describe po nginx kubectl get po nginx -w # watch it

**30. Create a redis pod, and have it use a non-persistent storage Note: In exam, you will have access to kubernetes.io site, Refer : https://kubernetes.io/docs/tasks/configure-pod-container/configurevolume-storage/**

Solution: apiVersion: v1 kind: Pod metadata: name: redis spec: containers: - name: redis image: redis volumeMounts: - name: redis-storage mountPath: /data/redis ports: - containerPort: 6379 volumes: - name: redis-storage emptyDir: {}

**31. Create a Pod with three busy box containers with commands "ls; sleep 3600;", "echo Hello World; sleep 3600;" and "echo this is the third container; sleep 3600" respectively and check the status**

Solution: // first create single container pod with dry run flag kubectl run busybox --image=busybox --restart=Always --dry-run -o yaml -- bin/sh -c "sleep 3600; ls" > multi-container.yaml // edit the pod to following yaml and create it apiVersion: v1 kind: Pod metadata: labels: run: busybox name: busybox spec: containers: - args: - bin/sh - -c - ls; sleep 3600 image: busybox name: busybox-container-1 - args: - bin/sh - -c - echo Hello world; sleep 3600 image: busybox name: busybox-container-2 - args: - bin/sh - -c - echo this is third container; sleep 3600 image: busybox name: busybox-container-3 restartPolicy: Always // Verify Kubectl get pods

**32. Check logs of each container that "busyboxpod-{1,2,3}"**

Solution: kubectl logs busybox -c busybox-container-1 kubectl logs busybox -c busybox-container-2 kubectl logs busybox -c busybox-container-3

**33. Create a Pod with main container busybox and which executes this "while true; do echo `Hi I am from Main container' >> /var/log/index.html; sleep 5; done" and with sidecar container with nginx image which exposes on port 80. Use emptyDir Volume and mount this volume on path /var/log for busybox and on path /usr/share/nginx/html for nginx container. Verify both containers are running.**

Solution: // create an initial yaml file with this kubectl run multi-cont-pod --image=busbox --restart=Never -- dry-run -o yaml > multi-container.yaml // edit the yml as below and create it kubectl create -f multi-container.yaml vim multi-container.yaml apiVersion: v1 kind: Pod metadata: labels: run: multi-cont-pod name: multi-cont-pod spec: volumes: - name: var-logs emptyDir: {} containers: - image: busybox command: ["/bin/sh"] args: ["-c", "while true; do echo 'Hi I am from Main container' >> /var/log/index.html; sleep 5;done"] name: main-container volumeMounts: - name: var-logs mountPath: /var/log - image: nginx name: sidecar-container ports: - containerPort: 80 volumeMounts: - name: var-logs mountPath: /usr/share/nginx/html restartPolicy: Never // Create Pod kubectl apply -f multi-container.yaml //Verify kubectl get pods

**34. Exec into both containers and verify that main.txt exist and**

Solution: NEED TO WRITE ANSWER FOR THIS

**35. Create an nginx pod and set an env value as 'var1=val1'. Check the env value existence within the pod**

Solution: kubectl run nginx --image=nginx --restart=Never --env=var1=val1 # then kubectl exec -it nginx -- env # or kubectl exec -it nginx -- sh -c 'echo $var1' # or kubectl describe po nginx | grep val1 # or kubectl run nginx --restart=Never --image=nginx --env=var1=val1 -it --rm - env

Bottom of Form

|  |
| --- |
| root@controlplane:~# k get po --show-labels --selector env=dev  NAME READY STATUS RESTARTS AGE LABELS  app-1-kltk2 1/1 Running 0 6m bu=finance,env=dev,tier=frontend  app-1-pxpsh 1/1 Running 0 6m bu=finance,env=dev,tier=frontend  app-1-tzz8w 1/1 Running 0 6m bu=finance,env=dev,tier=frontend  db-1-8mlpk 1/1 Running 0 5m59s env=dev,tier=db  db-1-p27cd 1/1 Running 0 5m59s env=dev,tier=db  db-1-sszjv 1/1 Running 0 5m59s env=dev,tier=db  db-1-w94qx 1/1 Running 0 5m59s env=dev,tier=db  selector is like real selector, you can specify just key or key=value to tighten the filter. NOTE – we can use –l as selector..  root@controlplane:~# k get po --show-labels -L env  NAME READY STATUS RESTARTS AGE ENV LABELS  app-1-kltk2 1/1 Running 0 6m6s dev bu=finance,env=dev,tier=frontend  app-1-pxpsh 1/1 Running 0 6m6s dev bu=finance,env=dev,tier=frontend  app-1-tzz8w 1/1 Running 0 6m6s dev bu=finance,env=dev,tier=frontend  app-1-zzxdf 1/1 Running 0 6m5s prod bu=finance,env=prod,tier=frontend  app-2-lbznn 1/1 Running 0 6m5s prod env=prod,tier=frontend  auth 1/1 Running 0 6m5s prod bu=finance,env=prod  db-1-8mlpk 1/1 Running 0 6m5s dev env=dev,tier=db  db-1-p27cd 1/1 Running 0 6m5s dev env=dev,tier=db  db-1-sszjv 1/1 Running 0 6m5s dev env=dev,tier=db  db-1-w94qx 1/1 Running 0 6m5s dev env=dev,tier=db  db-2-l862s 1/1 Running 0 6m5s prod bu=finance,env=prod,tier=db  There is nothing like env=dev selection when it comes to –L, it will just key and there will be column created like EMV with value populated…  root@controlplane:~# k get po --show-labels --selector bu=finance,tier=frontend,env=prod  NAME READY STATUS RESTARTS AGE LABELS  app-1-zzxdf 1/1 Running 0 9m15s bu=finance,env=prod,tier=frontend  root@controlplane:~# |

|  |
| --- |
| Deleting a pod which is running on any other node,  root@controlplane:/etc/kubernetes/manifests# ssh node01  root@node01:~# ps -auxww | grep **/bin/kubelet**  root 11051 0.0 0.1 3780360 99848 ? Ssl 10:54 0:33 /usr/bin/kubelet --bootstrap-kubeconfig=/etc/kubernetes/bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf **--config=/var/lib/kubelet/config.yaml** --network-plugin=cni --pod-infra-container-image=k8s.gcr.io/pause:3.2  root 17422 0.0 0.0 11468 1048 pts/0 R+ 11:06 0:00 grep --color=auto /bin/kubelet  root@node01:~# cat /var/lib/kubelet/config.yaml | grep -i static  **staticPodPath: /etc/just-to-mess-with-you**  root@node01:~# cd /etc/just-to-mess-with-you  root@node01:/etc/just-to-mess-with-you# ls -ltr  total 4  -rw-r--r-- 1 root root 301 May 13 10:54 greenbox.yaml  root@node01:/etc/just-to-mess-with-you# \rm greenbox.yaml  root@node01:/etc/just-to-mess-with-you# ls -ltr  total 0  root@node01:/etc/just-to-mess-with-you# exit  logout  Connection to node01 closed.  root@controlplane:/etc/kubernetes/manifests# k get po --all-namespaces | grep green  root@controlplane:/etc/kubernetes/manifests# |



CRI – Container Runtime Interface

CNI – Container Networking Interface

CSI - Container Storage Interface