CKA Crash Course

Intro

Your Instructor

- Sander van Vugt
- Author of many Linux, Ansible and Kubernetes related courses on this platform
- mail@sandervanvugt.nl
- Founder of the Living Open Source Foundation: livingopensource.net

This course

- This is not an introduction course to Kubernetes
- To successfully attend this course and prepare for the CKA exam, you should have good basic knowledge of Kubernetes already
- The perfect learning path is
  - Managing Containers on Linux
  - Kubernetes in 4 Hours
  - Certified Kubernetes Application Developer (CKAD) Crash Course

Poll Question 1

Rate your own Kubernetes knowledge/experience

- Just beginning
- Unstructured working knowledge
- Knowledge at CKAD level
- Knowledge at CKA level

Poll Question 2

Where are you from?

- Africa
- India
- Asia (not India)
- North and Central America
- South America
- Netherlands
- Europe
- Australia/pacific

Required Lab Environment

- 3 virtual machines running CentOS 7
- 1 control node
  - 2 vCPUs
  - 2 GB RAM
  - 20 GB disk
  - No swap
- 2 worker nodes
  - 1 vCPU
  - 1 GB RAM
  - 20 GB disk
  - No swap
- Disable firewalld

Optional HA Lab Requirements

- A new cluster with 5 virtual machines running CentOS 7
- 3 control node
  - 2 vCPUs
  - 2 GB RAM
  - 20 GB disk
  - No swap
- 2 worker nodes
  - 1 vCPU
  - 1 GB RAM
  - 20 GB disk
  - No swap
- Disable firewalld

Course resources

- https://github.com/sandervanvugt/cka
- https://kubernetes.io/docs
- Cheat Sheet
- Stay away from exam crams (you're going for the knowledge right?)

Course Agenda

This course does not have a strict agenda, as it is an interactive scenario based course. The following topics are included

- Installing a Kubernetes cluster
- Backup of the etcd database
- Configuring pods for autostart
- Configuring HA
- Managing RBAC
- Kubernetes storage, ConfigMaps and Secrets
- Managing Scheduler settings
- Resource monitoring
- Node maintenance
- Managing DNS
- Managing Network Policy

CKA Crash Course

Understanding this Course Philosophy

How NOT to learn for CKA

- You won't learn by memorizing bullet points on slides
- You won't learn by copy-pasting lines of code in a demo environment
- You won't learn by getting all the exam questions and memorize the answers

How to Learn for CKA

- Understand what you're doing and what the different objects are all about
- Practice to become fast in it
- Use available resources to find information in case you don't know it

How this is applied in this course

- This course is scenario based
- We work with sample questions, which could look like the questions that you'll see on the exam
- I'll use my virtual whiteboard to explain concepts behind the objects you have to work with
- You'll use available resources to figure out how to do it – just the way you would do it on the exam
- You will have a few minutes for each of the scenarios, then we'll discuss
- And by the end of this course you've build understanding and skills to get you successfully through the exam
- When needed, slides with a procedure description is added

CKA Crash Course

Generic Exam Tips

Before you Sign up

- See https://training.linuxfoundation.org/certification/certified-kubernetes-administrator-cka/ for the current objectives
- Notice that the exam follows the latest Kubernetes release, so a new update is published every 3 months!
- With the exam voucher, one free retake is included

Documentation is available!

While doing the exam

- One browser tab with access to docs.kubernetes.io is allowed
- Use **kubectl explain** for any additional details
- Notice that some documentation is outdated / inaccurate

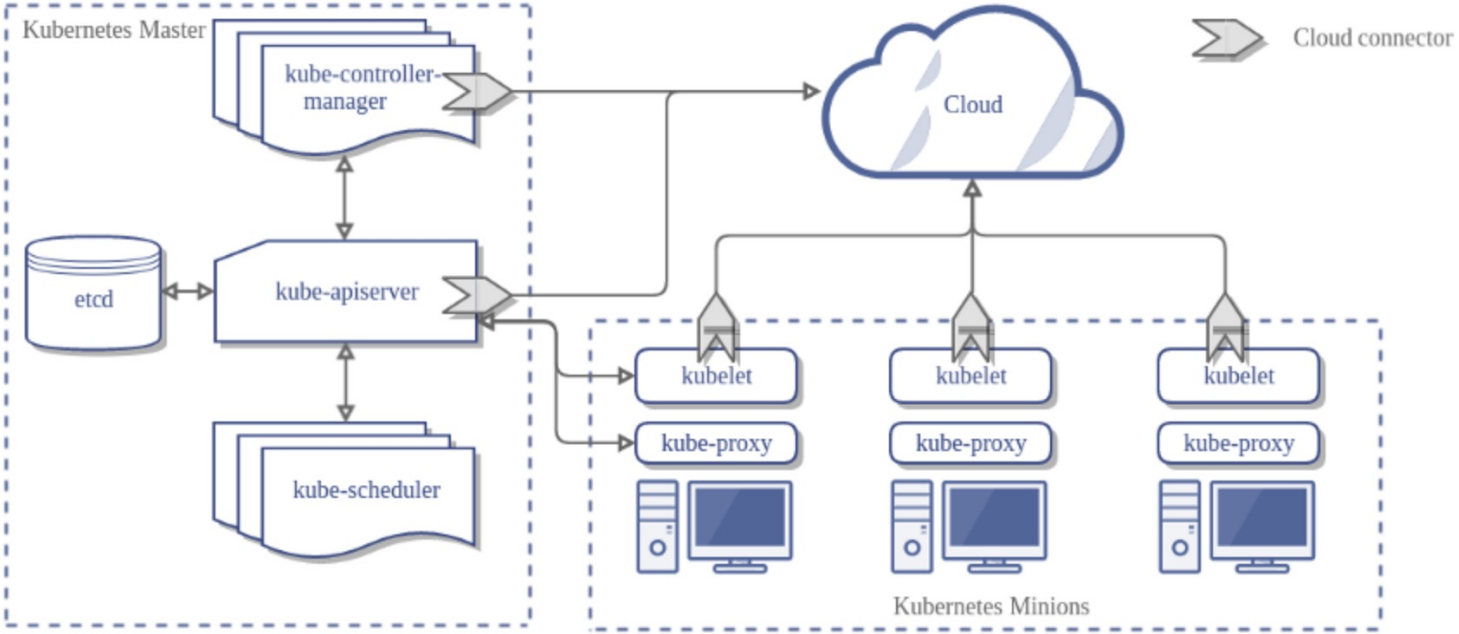
CKA Crash Course

1. Creating a Cluster

1. Creating a Cluster

1.1 Understanding Kubernetes Architecture

Architecture Overview



Understanding the Control Node

- kube-apiserver: front-end of the cluster that services REST operations and connects to the etcd database
- kube-scheduler: schedules pods on specific nodes based on labels, taints and tolerations set for the pods
- etcd: a B+tree key-value store that keeps the current cluster state. Organized with master and following instances of the database
- kube-controller-manager: manages current state of the cluster
- cloud-controller-manager: interacts with outside cloud managers
- Different optional add-ons
  - DNS
  - Dashboard
  - Cluster level resource monitoring
  - Cluster level logging

Understanding the Worker Nodes

- kubelet: passes requests to the container engine to ensure that Pods are available
- kube-proxy: runs on every node and uses iptables to provide an interface to connect to Kubernetes components
- container runtime: takes care of actually running the containers
- supervisord: monitors and guarantees the availability of the kubelet and docker processes
- network agent: implements a software defined networking solution, such as weave
- logging: the CNCF project Fluentd is used for unified logging in the cluster. A fluentd agent must be installed on the K8s nodes

1. Creating a Cluster

1.2 Understanding node Networking Requirements

Cluster Node Requirements

- To set up a Kubernetes on-premise cluster, **kubeadm** is used
- You'll need a minimum of one node, for the setup used in this course you'll need 3 nodes, 5 nodes tomorrow
- Use Centos 7.x or Ubuntu for best support
- The control node needs 2 CPUs
- All nodes need 1 GiB for a test environment, a minimum of 2 GiB or (much) more is required for production environments
- Install without using swap space
- Shut off the firewall

Software Installation

- Before starting installation, you need a container runtime
- Different runtimes are supported, in this course we'll use **docker**
- To make installation easy, use **git clone** https://github.com/sandervanvugt/cka which provides 2 scripts
  - **setup-docker.sh** installs the Docker container runtime
  - **setup-kubetools.sh** installs the Kubernetes tools
- As root, run these scripts on all nodes

Understanding the Procedure

- Prepare nodes
- Run **kubeadm init** on the control node as root
- Create .kube/config on control node (or client) as non-root
- Add network plugin using kubectl as non-root
- Join worker nodes using kubeadm as root
  - **kubeadm token create --print-join-command**
- As non-root on control: use **kubectl get pods -A**

Installing a Pod Network Add-on

- A network add-on must be installed for pods to communicate
- CNI is the Container Network Interface. It works with add-ons to implement networking
- Different project exist for offering Kubernetes network support, which requires support for the following types of networking:
  - container-to-container
  - pod-to-pod
  - pod-to-service
  - external-to-service
- Look for an add-on that supports *network-policy* as well as *RBAC* (both covered later in this course)
- In this course, we'll use the Weave add-on
- Notice the documentation has moved and is now obtained from weave.works/docs/net/latest/kubernetes/kube-addon/#install

Common Pod Networking Plugins

- Flannel: a layer 3 IPv4 network between cluster nodes. Can use several backend mechanisms such as VXLAN
- Weave: a common add-on for a CNI-enabled Kubernetes cluster
- Calico: a layer 3 network solution that uses IP encapsulation and is used in Kubernetes, OpenStack, OpenShift, Docker and others
- AWS VPC

Exercise 1

- Use **kubeadm** to create a cluster. control.example.com is set up as cluster controller node, worker{1..3} are set up as worker nodes
- The task is completed if **kubectl get nodes** shows all nodes in a ready state

CKA Crash Course

2. Creating a Pod

Exercise 2

- Create a Pod that runs the latest version of the alpine image. This pod should be configured to run the sleep 3600 command repeatedly and it should be created in the mynamespace namespace

CKA Crash Course

3. Creating a Multi-container Pod

Exercise 3

- Configure a Pod that runs containers based on the 3 following images
  - redis
  - nginx
  - busybox

CKA Crash Course

4. Creating a Pod with an init Container

Exercise 4

- Configure a Pod that runs 2 containers. The first container should create the file /data/runfile.txt. The second container should only start once this file has been created. The second container should run the "sleep 10000" command as its task

CKA Crash Course

5. Creating a Deployment

Exercise 5

- Create a Deployment with the name nginx-ex5. The deployment should start 5 replicas and use the nginx image

CKA Crash Course

6. Creating a Manifest File

Exercise 6

- Create a Deployment using a Manifest File. The deployment should run a Busybox container that executes the "sleep 10000" command, and it should start 3 replicas. Run the manifest file to create the desired configuration. After running it, make sure that all API objects created by the deployment are removed again.

CKA Crash Course

7. Applying Restrictions

Exercise 7

- Create a namespace with the name restricted. In this namespace, no more than 3 pods can be operational, and the name space is limited to a max. usage of 1 GB
- Start an nginx deployment that runs 3 pod instances
- Try to start a busybox pod, using the command sleep3600. Verify that it fails

CKA Crash Course

8. Configure a ConfigMap

Understanding ConfigMap

- ConfigMaps can be used to separate dynamic data from static data in a Pod
- They are not encoded or encrypted
- They can be used in three different ways:
  - Make variables available within a Pod
  - Provide command line arguments
  - Mount them on the location where the application expects to find a configuration file
- Secrets are encoded ConfigMaps which can be used to store sensitive data
- ConfigMaps must be created before the pods that are using them

Understanding ConfigMap Sources

- ConfigMaps can be created from different sources
  - Directories: uses multiple files in a directory
  - Files: puts the contents of a file in the ConfigMap
  - Literal Values: useful to provide variables and command arguments that are to be used by a Pod
- Since Kubernetes 1.14 the kustomization.yaml generator can be used
  - This used the configMapGenerator API object to generate the ConfigMap based on input data in the kustomization.yaml file

Procedure Overview

- Start by defining the ConfigMap and create it
  - Consider the different sources that can be used for ConfigMaps
  - **kubectl create cm myconf --from-file=my.conf**
  - **kubectl create cm variables --from-env-file=variables**
  - **kubectl create cm special --from-literal=VAR3=planets --from-literal=VAR4=moon**
  - Verify creation, using **kubectl describe cm <cmname>**
- Use **--from-file** to put the contents of a config file in the configmap
- Use **--from-env-file** to define variables
- Use **--from-literal** to define variables or command line arguments

Procedure Overview - 2

- To include *variables* from a ConfigMap:

```
envFrom:
  - configMapRef:
      name: ConfigMapName
```

- To include *config files* from a ConfigMap:

```
volumes:
  - configMap:
      name: ConfigMapName
      items:
      - key: my-custom.conf
        path: default.conf
```

Demo: Creating a ConfigMap from a File

- Show contents of variables (in github)
- Create the ConfigMap: **kubectl create cm variables --from-env-file=variables**
- Verify creation: **kubectl describe cm variables**
- Create a Pod: **kubectl create –f cm-test-pod1.yml**
- Check that the variables are available: **kubectl logs po/test** (or whatever the name is)

Demo: Configuring a ConfigMap from a Literal

- **kubectl create cm morevars --from-literal=VAR3=planets --from-literal=VAR4=moon**
- **kubectl get cm/morevars**

Demo: Using ConfigMaps for ConfigFiles

- Create the ConfigMap: **kubectl create cm nginx-cm --from-file nginx-custom-config.conf**
- Check the contents of the ConfigMap: **kubectl get configmap/nginx-cm –o yaml**
- Next, create the Pod: **kubectl create -f nginx-cm.yml**
- Check the config file: **kubectl exec -it nginx-cm /bin/bash**
- **cat /etc/nginx/conf.d/default.conf**

Exercise 8

- Create a ConfigMap that defines the variable myuser=mypassword. Create a Pod that runs Alpine, and uses this variable from the configMap

CKA Crash Course

9. Creating a Persistent Volume

Exercise 9

- Create a Persistent Volume that uses local host storage. This PV should be accessible from all name spaces. Run a Pod with the name pv-pod that uses this persistent volume from the "myvol" namespace

10. Running a Pod in a Namespace

exercise 10

- In the run-once namespace, run a Pod with the name xxazz-pod, using the alpine image and the command sleep 3600. Create the namespace if needed. Ensure that the task in the Pod runs once, and after running it once, the Pod stops

CKA Crash Course

11. Create and Upgrade a Deployment

Exercise 11

- Create a Deployment that runs Nginx, based on the 1.14 version. After creating it, enable recording, and perform a rolling upgrade to upgrade to the latest version of Nginx. After successfully performing the upgrade, undo the upgrade again.

CKA Crash Course

12. Find Pods by Label

Exercise 12

- Find all Pods that have the label app set to working. Use kubectl features to sort this list on host name, and write the resulting output to the file /var/exam/sortlist.txt

CKA Crash Course

13. Expose a Pod

Exercise 13

- Expose the Nginx Pod such that it can be reached by external users. To expose it, use a cluster IP address

CKA Crash Course

14. Creating and Using a Secret

Exercise 14

- Create a secret that contains the variable definition userpw=s3cr3tPW
- Configure one Pod to use that secret by mounting it in the /etc/secret directory. Configure another Pod that uses that secret in a way that shows userpw is scrambled when the Pod YAML code is dumped

15. Configure a Daemonset

Exercise 15

- Run a Daemonset, that ensures your Nginx application runs on every host. The name of the application should be nginx-ds, and it should run the latest version of the nginx image

CKA Crash Course

16. Using NetworkPolicy

Understanding Network Policies

- By default, there are no restrictions to network traffic in K8s
- Pods can always communicate, even if they're in other namespaces
- To limit this, Network Policies can be used
- Network Policies need to be supported by the network plugin though
- If in a policy there is no match, traffic will be denied
- If no Network Policy is used, all traffic is allowed

Network Policy Identifiers

- In Network Policies, three different identifiers can be used
  - Pods: (podSelector) note that a Pod cannot block access to itself
  - Namespaces: (namespaceSelector) to grant access to specific namespaces
  - IP blocks: (ipBlock) notice that traffic to and from the node where a pod is running is always allowed
- When defining a pod- or namespace-based NetworkPolicy, a selector label is used to specify what traffic is allowed to and from the Pods that match the selector
- Network Policies do not conflict, they are additive

Demo

- **kubectl apply -f nwpolicy-complete-example.yaml**
- **kubectl expose pod nginx --port=80**
- **kubectl exec -it busybox -- wget --spider --timeout=1 nginx** will fail
- **kubectl label pod busybox access=true**
- **kubectl exec -it busybox -- wget --spider --timeout=1 nginx** will work

Lab 16: Managing Network Policy

- Run a Pod with the name nwp-busybox from the new namespace nwp-namespace
- Run a Pod with the name nwp-nginx in the default namespace
- Expose the nwp-nginx Pod on port 80
- Verify there is nothing preventing the busybox pod from accessing the nginx Pod in the default namespace
- Write a network policy that blocks all access from the nwp-namespace to the default namespace
- Verify that the busybox Pod can no longer access the nginx Pod

17. Enable a Node to Run Pods Again

Exercise 17

- Analyze the nodes in the local cluster. One of the nodes is not available to run Pods. Enable it to run Pods again

18. Mark a Node as Unavailable

Exercise 18

- In your cluster, mark the node worker2 as unavailable. Ensure that all Pods are moved away from the local node and are started again somewhere else

19. Allow Pods to run on the Control Node

Exercise 19

- Allow user Pods to also run on the control node

20. Finding the Pod with the Highest CPU Load

Exercise 20

- Find the Pod with the highest CPU load and write its name to the file /var/exam/cpu-pods.txt

Configuring Monitoring and **kubectl top**

- **git clone https://github.com/kubernetes-sigs/metrics-server.git**
- Read the documentation on the Github Repo!!!
- **kubectl create –f metrics-server/manifests/base/** (see command on git doc)
- **kubectl –n kube-system get pods** # look for metrics-server
- **kubectl –n kube-system edit deployment metrics-server**
  - In spec.template.spec.containers.args, add
    - --kubelet-insecure-tls
    - --kubelet-preferred-address-types=InternalIP,ExternalIP,Hostname
  - Under dnsPolicy, add a line that reads **hostNetwork: true** (might not be needed anymore in 1.19!)
- **kubectl –n kube-system logs metrics-server<TAB>** should show "Generating self-signed cert" and "Serving securely on [::]443

Running **kubectl top**

- **kubectl top pods --all-namespaces** will show most active Pods
- Give it time to collect the metrics – should work in about 60 seconds

CKA Crash Course

21. Backing up the etcd Database

Exercise 21

- Create a backup of the etcd database. API version 3 is used for the current database. Write the backup to /var/exam/etcd-backup

CKA Crash Course

22. Pod and Service DNS Connectivity

Testing DNS - 1

1. Create busybox.yaml

```
apiVersion: v1
kind: Pod
metadata:
 name: busybox2
 namespace: default
spec:
 containers:
 - image: busybox
   name: busy
   command:
     - sleep
     - "3600"
```

2. Create the pod, using **kubectl create -f busybox.yaml**
3. Use **kubectl get svc** to validate the name of any exposed service
4. Use **kubectl exec -ti busybox2 -- nslookup kubernetes**
5. You'll see the IP address being resolved, thus providing proof that DNS is working correctly
6. Note that services can be resolved through DNS, Pods cannot by default

Testing DNS - 2

- **kubectl create –f pod-and-service-dns.yaml**
- **kubectl exec –it busybox2 --nslookup default-subdomain**
- **kubectl exec –it busybox2 --nslookup busybox-1**

Understanding DNS

- CoreDNS is the default DNS service in Kubernetes
- Its IP address is exposed by a service kube-dns that lives in the kube-system namespace
- This service IP address should match the contents of /etc/resolv.conf on the Pod nodes
- When starting a container, the Kubelet passes DNS to it, using **--cluster-dns=<dns-service-ip>**
- Also, the kubelet is configured with its local DNS domain, using **–cluster-domain=<default-local-domain>**

Analyzing DNS - 1

- Check Pod DNS resolver: **kubectl exec podname -- cat /etc/resolv.conf**
  - The nameserver should match the IP address of the core-DNS service
  - You may have a search path, containing **search default.svc.cluster.local svc.cluster.local cluster.local …** (you should see this path being queried also)
- Use **kubectl get pods --namespace=kube-system –l k8s-app=kube-dns** to verify the CoreDNS Pod is up
- If pods fail: **kubectl –n kube-system describe pods core-dns-nnn**
- Check logs in CoreDNS Pods: **for p in $(kubectl get pods --namespace=kube-system –l k8s-app=kube-dns –o name); do kubectl logs --namespace=kube-system $p; done**

Analyzing DNS – 2

- Verify the DNS service is up: **kubectl get svc –n kube-system**
- Verify the endpoints are exposed: **kubectl get ep kube-dns –n kube-system**

Troubleshooting DNS

- Disable all firewalling on all nodes**: systemctl disable --now firewalld** such that the network plugin can create iptables rules to set up traffic in the kube network
- Restart Dockerd**: systemctl restart docker**
- Remove core-dns Pods: **kubectl delete pod –n kube-system –l k8s-app=kube-dns**, they are automatically recreated
- Remove your network plugin pod and re-install
- Replace network plugin: Calico is doing better than Weave

Resolving Pod DNS Names

- You typically don't need to resolve Pod DNS names (they aren't accessible directly anyway); you'll address the service and not the Pod
- < 1.18: If you need to do it anyway, use podipaddress.namespace.pod
  - Use **kubectl get pods -o wide** to see IP addresses
  - Resolve as 10-17-0-23.mynamespace.pod
- >= 1.19: **kubectl exec -it busybox22 -- nslookup podname**

Exercise 22

- Start a Pod that runs busybox image. Use the name busy22 for this Pod. Expose this Pod on a cluster IP address. Configure the Pod and Service such that DNS name resolution is possible, and use the **nslookup** command to look up the names. Write the output of the DNS lookup command to the file /var/exam/dnsnames.txt

23. Configuring a Node to Autostart a Pod

Exercise 23

- Configure your node worker3 to automatically start a Pod that runs an nginx web server, using the name auto-web. Put the manifest file in /etc/kubernetes/static

24. Troubleshooting Cluster Connectivity

Troubleshooting the Cluster

- Restoring .kube/config
  - **sudo cp –I /etc/kubernetes/admin.conf $HOME/.kube/config**
  - **sudo chown $(id –u):$(id –g) $HOME/.kube/config**
- Check the kubelet service: **systemctl status kubelet**

Exercise 24

- You currently are incapable of accessing the cluster, getting a service unavailable error message. Troubleshoot this problem

CKA Crash Course

25. Node Scheduling

Using nodeSelector and nodeName

- The nodeSelector field in the pod.spec specifies a key-value pair that must match a label which is set on nodes that are eligible to run the Pod
- Use **kubectl label nodes worker1.example.com disktype=ssd** to set the label on a node
- Use **nodeSelector: disktype: ssd** in the pod.spec to match the Pod to the specific node
- **nodeName** is part of the pod.spec and can be used to always run a Pod on a node with a specific name
  - Not recommended: if that node is not currently available, the Pod will never run

demo

- **kubectl label nodes worker2.example.com disktype=ssd**
- **kubectl apply –f selector-pod.yaml**

Understanding Affinity

- nodeSelector provides a simple way to constrain nodes, the (anti) Affinity feature enhances the options
  - it uses a more expressive language
  - it offers an option to use soft rules
  - it can work with labels that are set on other pods, to make sure that specific pods cannot be co-located
- Also, two types of Affinity are offered
  - Node Affinity sets Affinity rules on nodes
  - inter-pod Affinity specifies rules between Pods

Understanding Taints

- *Taints* are applied to a node to mark that the node should not accept any Pod that doesn't tolerate the taint
- *Tolerations* are applied to Pods and allow (but do not require) Pods to schedule on nodes with matching Taints – so they are an exception to taints that are applied
- Where Affinities are used on Pods to attract them to specific nodes; Taints allow a node to repel a set of Pods
- Taints and Tolerations are used to ensure Pods are not scheduled on inappropriate nodes, and thus make sure that dedicated nodes can be configured for dedicated tasks

Understanding Taint Types

- Three types of Taint can be applied:
- NoSchedule: does not schedule new Pods
- PreferNoSchedule: does not schedule new Pods, unless there is no other option
- NoExecute: migrates all Pods away from this node
- If the Pod has a toleration however, it will ignore the taint

demo

- **kubectl taint nodes worker1.example.com example-key=value1:NoSchedule**
- **kubectl describe nodes worker1.example.com**
- **kubectl create deployment nginx-taint --image=nginx**
- **kubectl scale deployment nginx-taint –replicas=3**
- **kubectl get pods –o wide** # will show that pods are all on worker2
- **kubectl create –f taint-toleration.yaml** # Note the example-key which much match the key that was set in the node

Exercise 25

- Create an nginx resource that will never run on the node worker2

CKA Crash Course

26. RBAC

Understanding Users

- Kubernetes user accounts are NOT oridinary user accounts that exist in a centralized database
- In fact there are different ways to create a Kubernetes user account
- An easy way that doesn't require any external authentication, is the X509 based user
- The procedure roughly consists of two parts:
  - Use **openssl x509** to create the user public/private key pair
  - Use the Kubernetes CA to validate the key pair
  - Put the keys in the appropriate directory
  - Use **kubectl config** to create the user as well as a context
  - Based on /etc/kubernetes/admin.conf on the master node, create a user configuration file

Demo: Creating a Kubernetes User - 1

- Perform the following steps with root privileges to create user anna, as well as a key pair for this user
- **sudo useradd anna && su -; cd /home/anna**
- **sudo openssl genrsa -out anna.key 2048**
- **sudo openssl req -new -key anna.key -out anna.csr -subj "/CN=anna"**
- **sudo openssl x509 -req -in anna.csr -CA /etc/kubernetes/pki/ca.crt -CAkey /etc/kubernetes/pki/ca.key -CAcreateserial -out anna.crt -days 600**
- **sudo mkdir .certs && mv anna.crt anna.key .certs/**

Demo: Creating a Kubernetes User - 2

- Perform the following as **kube-admin**, but with access to /home/anna (use as current directory)
- **kubectl config set-credentials anna --client-certificate=/home/anna/.certs/anna.crt --client-key=/home/anna/.certs/anna.key**
- **kubectl config set-context anna-context --cluster=kubernetes --user=anna**
- **sudo mkdir .kube**
- **sudo cp /etc/kubernetes/admin.conf /home/anna/.kube/config**
- Modify according to instructions on the next slide (see GitHub config.anna)
- After modifying, don't forget to make anna owner of all files in her $HOME: **chown -R anna /home/anna/**

Demo: Creating a Kubernetes User - 3

```
contexts:
- context:
    cluster: kubernetes
    user: anna
  name: anna-context
current-context: anna-context
kind: Config
preferences: {}
users:
- name: anna
  user:
    client-certificate: /home/anna/.certs/anna.crt
    client-key: /home/anna/.certs/anna.key
```

Understanding RBAC Objects

- Role: Used to provide access to resources within a namespace
- ClusterRole: Used to give access to:
  - cluster-scoped resources
  - non-resource endpoints
  - namespaced resource across all namespaces
- RoleBinding: Used to give permissions to a user or a set users. Used within a namespace
- ClusterRoleBinding: Like RoleBinding, but for a cluster. Can be used for namespaces, or limited to one namespace if this is specified

Demo: Configuring RBAC

- In this demo we'll create a simple RBAC configuration based on user anna created in previous demo. Use your regular K8s-admin account
- **kubectl create -f podaccessrole.yaml**
- **kubectl create -f rolebinding.yaml**
- **kubectl run rolepod --image=nginx**
- **kubectl get pods**
- Now become anna to test the limitations of the anna role:
- **su - anna**
- **kubectl get pods**
- **kubectl get all**

Using **kubectl can-i**

- Ad kube-admin: **kubectl auth can-i get pods --as anna**
- As anna: **kubectl auth can-i get pods**

CKA Crash Course

27. Configuring High Availability

Understanding HA Options

- A load balancer is used by the worker nodes to distribute workload across cluster nodes
- *Stacked control plane nodes* requires less infrastructure as the etcd members and control plane nodes are co-located
  - Control planes and etc members are running together on the same node
  - For optimal protection, requires a minimum of 3 stacked control plane nodes
- *External etcd cluster* requires more infrastructure as the control plane nodes and etcd members are separated
  - Etcd service is running on external nodes, so this requires twice the number of nodes

Node Requirements

- 3 VMs to be used as controllers in the cluster. Install K8s software but don't init the cluster yet
- 2 VMs to be used as worker nodes. Install K8s software
- Ensure /etc/hosts is set up for name resolution of all nodes and copy to all nodes
- Disable **selinux** on all nodes if applicable
- Disable firewall if applicable

Create a Load Balancer

- In the loadbalancer setup, HAProxy can be used to balance load between the different master nodes
- To do so, keepalived can be used to provide a virtual IP address
- Keepalived ensures that the VIP is running on one of the master nodes, on a specific port (which is NOT 6443) we'll use 8443
- **kubectl** clients connect to this VIP:8443, from there HAProxy will load balance the work to one of the nodes
- Use the **setup-lb.sh** script provided in the GitHub repository for easy setup
- Additional instructions are in the script

Initialize the HA Setup

- **sudo kubeadm init --control-plane-endpoint "192.168.4.100:8443" --upload-certs**
  - Safe the output of the command which shows next steps
- Configure networking
  - **kubectl apply -f https://docs.projectcalico.org/v3.14/manifests/calico.yaml**
- Copy the **kubectl join** command that was printed after successfully initializing the first master
  - Make sure to use the command that has **--control-plane** in it!
- Complete setup on other masters as instructed
- Use **kubectl get nodes** to verify setup
- Continue and join worker nodes as instructed

Initialize the client

- On the machine you want to use as operator workstation, create a .kube directory and copy /etc/kubernetes/admin.conf from any master to the client machine
- Install the **kubectl** utility
- Ensure that host name resolution goes to the new master VIP

Verifying the HA Setup

- **kubectl get pods -n kube-system** will show multiple control plane services running on the different master nodes

28. Cluster upgrade overview

Cluster upgrade overview - 1

On the control node

- **yum makecache fast**
- Upgrade **kubeadm** using distribution tools
  - **yum list --showduplicates kubeadm --disableexcludes=kubernetes** will show available versions
  - On the control node, use **yum install -y kubeadm-<version-number> --disableexcludes=kubernetes**
- Use **kubectl drain control --ignore-daemonsets**
- Use **sudo kubeadm upgrade plan** to prepare upgrade and read instructions in the output
- Use **sudo kubeadm upgrade apply v1.19.x** (or whatever version)
- Uncordon the control plane using **kubectl uncordon control**

Cluster upgrade overview - 2

- Upgrade kubelet and kubectl: **sudo yum install -y kubelet-<latestversion> kubectl-<latestversion>**
- Restart kubelet: **sudo systemctl daemon-reload; sudo systemctl restart kubelet**

Cluster upgrade overview - 3

For the worker nodes, not all at the same time

- control $ **kubectl drain <nodename> --ignore-daemonsets --force --delete-local-data**
  - If **metrics-server** is complaining, delete it using **kubectl delete deployment metrics-server -n kube-system**
- worker $ **sudo kubeadm upgrade node**
- worker $ **sudo yum install -y kubelet-<latest> kubectl-<latest> --disableexcludes=kubernetes**
- worker $ **sudo systemctl daemon-reload; sudo systemcl restart kubelet**
- control $ **kubectl uncordon <nodename>**
- control $ **kubectl get nodes**

CKA Crash Course

29. Troubleshooting

Applying Troubleshooting

- Use the following commands to get you some trouble
  - **kubectl create ns trouble**
  - **kubectl run nginx --image=nginx --namespace=trouble**
  - **kubectl delete ns trouble**
  - Use **Ctrl-C** when you have waited long enough
- Check what is going on with **kubectl get ns**
- Fix it!

CKA Crash Course

30. Scheduling

Scheduling

- Mark the second worker node (worker2) in such a way that no new pods can be scheduled on it, and currenlty running pods are evicted. Create a deployment with the name "idontcare" that runs an nginx image and two replicas and will be able to run on worker2 anyway.

CKA Crash Course

Where to go from here

Where to go from here

- Consult training.linuxfoundation.org/certification/certified-Kubernetes-administrator-cka/
- Hopefully you're ready for the exam now! Time to order your voucher:
  - https://linuxfoundation.org