CKAD Crash Course

Sander van Vugt
mail@sandervanvugt.nl

Pearson

Agenda

- Creating the Lab Environment
- Understanding Containers
- Understanding Kubernetes
- Using Management Options
- Managing Pods
- Managing Deployments
- Managing Services
- Managing Ingress
- Managing Storage
- Managing Deployments and Secrets
- Troubleshooting

About Me

- Sander van Vugt
- Living in the Netherlands
- Author and presenter of many titles on this platform – Linux, Kubernetes and Ansible
- Founder of the Living Open Source Foundation
  - The mission of the Living Open Source Foundation is to stimulate the growth of local economies by enabling people to develop themselves as experts in the area of Open Source
  - Current focus is on education in Africa
  - See livingopensource.net for more information

Course resources

- YAML files used in this course are available at https://github.com/sandervanvugt/ckad

Poll Question 1

How would you rate your current Kubernetes Knowledge?

- None
- Beginner
- Intermediate
- Advanced
- Guru

Poll Question 2

How would you rate your current experience with containers?

- None
- Beginner
- Intermediate
- Advanced
- Guru

Poll Question 3

Which of the following statements best applies to you

- I'm new to Kubernetes, teach me everything you know!
- Give me an overview of Kubernetes, I want to understand what it is doing
- Give me hands-on! I want to learn how it works
- Focus on the exam questions! I already know my basics

This course vs my other Kubernetes courses

- Managing Containers on Linux: provides knowledge about the fundamentals of working with containers
- Kubernetes in 4 Hours: introduction level course to Kubernetes, very hands on
- CKAD: this course, focus on running applications in Kubernetes
- CKA: Your next course, focus on Kubernetes cluster administration

CKAD versus CKA

- CKAD can be considered an entry-level Kubernetes exam
- Candidates will need to create and manage applications in containers using standard ingredients
- CKA is more advanced and measures the candidates knowledge about manage Kubernetes clusters

CKAD Crash Course

1. Creating the Lab Environment

To do right NOW

- You can run Kubernetes in many ways. To make it easy to start fast, in this course I’m presenting minikube
- You can use Minikube in many ways. To make it easy to start fast, in the course I’m showing installation of Minikube in a Fedora WS virtual machine
- Start installing the Fedora WS VM now
  - 40GB disk
  - 8 GB RAM (4 GB minimal)
  - Enable CPU-level virtualization
    - In virtual box: VM > Settings > System > processor > Enable Nested VT-x/AMD-V
    - In Vmware: VM > Settings > Processor and memory > Advanced Options > Enable Hypervisor applications
- If you still need to download Fedora, that will have impact on the quality of the media stream in this session

1.1 Understanding Kubernetes Deployment Options

Understanding Deployment Options

- Kubernetes can run in many environments
  - In a public cloud
  - On-premise in a datacenter
  - Using minikube for testing and developing
- In this course we'll use minikube
- Learn how to set up a Kubernetes cluster using **kubeadm** in my CKA course
- Tip: for a quick online try without installation, use tryk8s.com or one of the many other trial platforms that is available

CKAD Crash Course

1.2 Using Minikube

Configuration Requirements

- Minikube offers a complete test environment that runs on Linux, OS-X or Windows
- In this course we'll focus on Minikube as it is easy to setup and has no further dependencies
- You'll also need to have the **kubectl** client on your management platform

Installing a Lab Environment

- Easy installation on top of Fedora 30 is provided through a lab setup script
- Clone my git repo: **git clone https://github.com/sandervanvugt/ckad**
- Run the **kube-setup.sh** script to setup Minikube as well as the **kubectl** client
- Or follow the manual procedure that is described on the next slides

Testing Minikube

- **minikube ssh**: logs in to the Minikube host
- **docker ps**: shows all Docker processes on the MK host
- **ps aux | grep localkube**: shows the localkube process on MK host

CKAD Crash Course

1.3 Running Your First Application

CKAD Crash Course

2. Understanding Containers

CKAD Crash Course

2.1 What is a Container

What is a Container

- A container is a self-contained ready to run application
- This is what makes it different from a virtual machine!
- Containers have all on board that is required to start the application
- To start a container, a container runtime is required
- The container runtime is running on a host platform and established communication between the local host kernel and the container
- So all containers no matter what they do run on top of the same local host kernel

Containers are Linux!

- Containers are based on features offered by the Linux operating system
- Linux Kernel Namespaces provide strict isolation between system components at different levels
  - network
  - file
  - users
  - processes
  - IPCs
- Linux CGroups offer resource allocation and limitation

Understanding Container Runtimes

- The container runtime allows for starting and running the container on top of the host OS
- The container runtime is responsible for all parts of running the container which are not already a part of the running container program itself
- Different container runtime solutions exist
    - lxc
    - runc
    - cri-o
    - rkt
    - containerd
- These runtimes are included into the different container solutions

Understanding the OCI

- OCI is the Open Containers Initiative (https://opencontainers.org)
- It standardizes the use of containers
  - The image-spec defines how to package a container in a "filesystem bundle"
  - The runtime-spec defines how to run that filesystem in a container
- OCI standardization ensures compatibility between containers, no matter which environment they originally come from

Understanding Docker

- Docker is important in the container landscape, but Docker is NOT the only way to run containers
- When it started in 2013, Docker offered the following:
  - Container image format
  - Dockerfile, which is a method for building container images
  - A way to manage container images
  - A way to run containers
  - A way to manage container instances
  - A solution to share container images
- As Docker is so common, we'll use it to demonstrate containers in this lesson, but Docker is not the only way!

Alternatives to Docker

- With the launch of RHEL 8, Red Hat started offering **podman** as an alternative to Docker
  - **podman** runs containers without the need of having a daemon, directly on top of the cri-o container runtime
  - **buildah** is the related service that is used for managing container images
- Other solutions for running containers also exist
  - **LXC** is a Linux-native container runtime
  - **systemd-nspawn** offers containers integrated in Systemd
  - And other solutions are available also

Understanding Container Components

- *Images* are read-only environments that contain the runtime environment that includes the application and all libraries it requires
- *Registries* are used to storage images. Docker Hub is a common registry, but private registries can be created also
- *Containers* are the isolated runtime environments where the application is running. By using *namespaces* the containers can be offered as a strictly isolated environment

CKAD Crash Course

3. Understanding Kubernetes

What is Kubernetes?

- Kubernetes is "an open-source system for automating deployment, scaling and managing of containerized applications" (https://kubernetes.io)
- The word Kubernetes comes from Greek, and means "pilot of the ship"

A New Way to do IT

- Kubernetes and Containers offer a new way of working with application workload
- In old IT, applications were installed on a server, and if more capacity was needed, the capacity of the server was scaled up
- Kubernetes offers a platform where the application is offered as containers, connecting to the application is organized through proxies, and scalability is easily organized using Deployment scalability features
- This is the *microservice* approach, where many instances of the server (i.e., the container) are available to respond to a request
- Users are decoupled from the actual container that runs the service they're connecting to. If that container goes down, nobody really cares as services are decoupled

CKAD Crash Course

3.2 Understanding Kubernetes Origins

Kubernetes Origins

- Kubernetes is based on Google Borg, the internal system that Google has been using for many years to offer access to its applications
  - https://ai.google/research/pubs/pub43438
- Borg was also the origin of important Linux kernel features, used in containers nowadays, such as namespaces
- Google donated the Borg specification to the Cloud Native Computing Foundation (CNCF) which was the start of Kubernetes

CKAD Crash Course

3.3 Understanding Kubernetes API Objects

What is an API?

- An Application Programmers Interface (API) is the interface between a client and a server
- An API is used to standardize how to access items provided by the server
- The Kubernetes API defines objects in a Kubernetes environment
- Any command that is used in a Kubernetes environment, is doing somewhat in some way with the API
- Also, all that can be done in Kubernetes, can be explained by exploring API features
- The **kubectl explain** command is an excellent way to explore API features

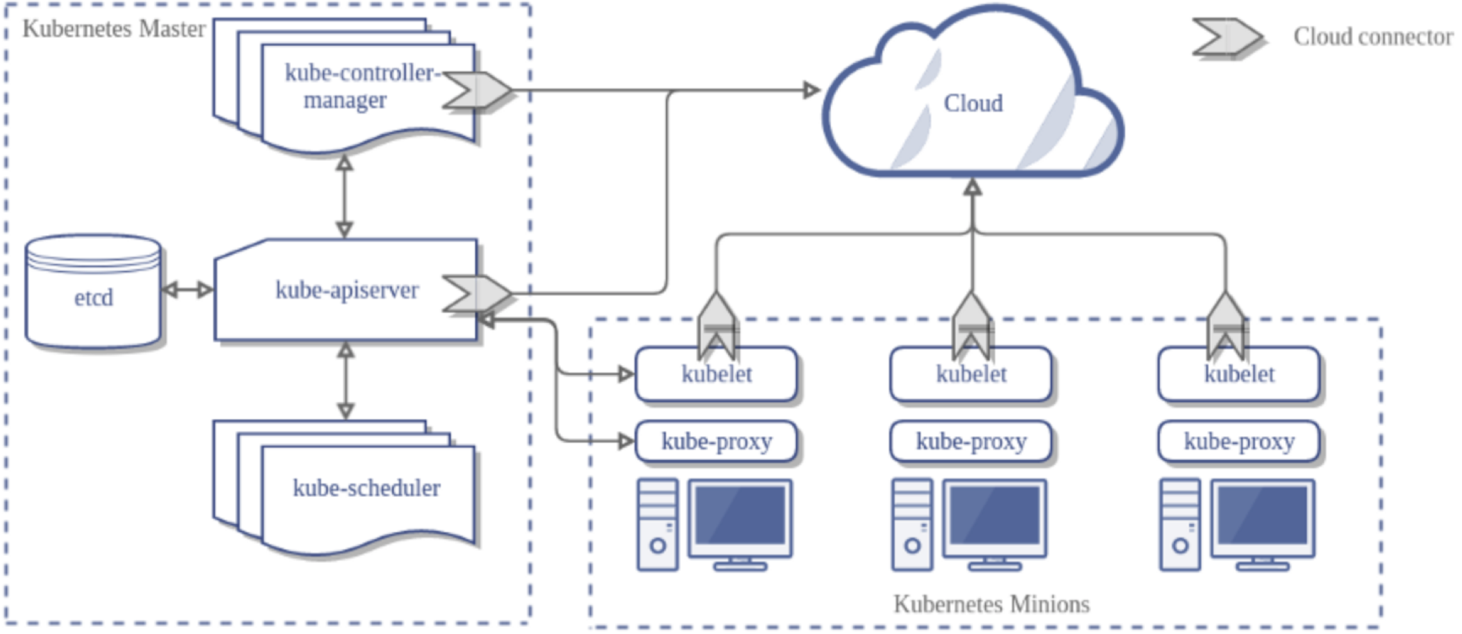
Working with Kubernetes

- The **kubectl** command line utility provides convenient administrator access, allowing you to run many tasks against the cluster
- Direct API access allows developers to address the cluster using API calls from custom scripts
- The Kubernetes Console

CKAD Crash Course

3.4 Kubernetes Architecture

Architecture Overview



Exam Lab 1

Generate a list of objects provided by the API. Explore the options that are available for creating a service object.

CKAD Crash Course

4. Understanding Managing Options

CKAD Crash Course

4.1 Understanding Main Kubernetes Components

Kubernetes Main Objects

- Pods: the basic unit in Kubernetes, represents a set of containers that share common resources such as an IP address and persistent storage volumes
- Deployments: standard entity that is rolled out with Kubernetes
- Services: make deployments accessible from the outside by providing a single IP/port combination. Services by default provide access to pods in round-robin fashion using a load balancer
- Persistent Volumes: persistent (networked) storage that can be mounted within a container by using a Persistent Volume Claim

4.2 Understanding the Kubernetes API

Understanding the API

- The API defines which objects exist, and how to use these object
- The main API documentation is here: https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/
- To make extension of the Kubernetes API easier, different groups are used
- Each API group can have its own version number
- See here for more information: https://kubernetes.io/docs/reference/using-api/api-overview/
- Use **kubectl api-resources** for information about resource types
- Or **kubectl api-versions** for resource and version information

Understanding API access

- Kubernetes functionality is exposed by the **kube-apiserver** in multiple API groups
- **kubectl** makes secured API requests,
- To make your own API requests using **curl**, the appropriate certificates are required, which is taken care of automatically by **kube-proxy**
- Note that this requires access to CA and certificates, without that you won't see much due to security settings
- Use **kubectl --v=10 get pods** (or any other command) to see the underlying API request

Connecting to the API

- The API server exposes its functionality through REST
- Assuming that minikube is running, connect to it by running a local proxy first (on your workstation for instance), after which you can connect using curl
  - **kubectl proxy --port=8001&**
  - **curl http://localhost:8001**
- This shows all of the available API paths and groups, providing access to all exposed functions
- Using a proxy allows you to use **curl** and send API calls directly, where the proxy takes care of the appropriate certificates
- Read the documentation for more info about the API groups
- Every API is built according to strict specifications and always contains specific elements

API Access Example

- On the host that runs kubectl: **kubectl proxy --port=8001 &**
- **curl http://localhost:8001/version**
- **curl http://localhost:8001/api/v1/namespaces/default/pods**-> shows the pods
- **curl http://localhost:8001/api/v1/namespaces/default/pods/httpd/** shows direct API access to a pod
- **curl -XDELETE http://localhost:8001/api/v1/namespaces/default/pods/httpd** will delete the hpptd pod

CKAD Crash Course

4.3 Using Dashboard

Using Kubernetes Dashboard

- Kubernetes Dashboard is *not* the recommended way of creating Kubernetes objects
- For an automated way of managing a Kubernetes environment, it is much better to use YAML manifest with the **kubectl** command
- YAML manifests are easy to reproduce, Graphical dashboard interfaces are not

4.4 Using **kubectl**

Understanding **kubectl**

- **kubectl** under the hood uses **curl** to send API requests to the Kubernetes API
- **kubectl** has many subcommands, making it possible to manage all aspects of Kubernetes
- Use **kubectl command --help** for documentation, including examples
- Tip! Also don't forget https://kubernetes.io/docs, which is available at the exam also!

Using **kubectl config**

- The context that **kubectl** uses is stored in ~/.kube/config
- This context defines which cluster to connect to
- Use **kubectl config view** to view different parts of the current configuration, including current context and namespace

Exam lab 2

Run a curl command to connect to the Kubernetes API and find out which Pods are created in the Default namespace.

CKAD Crash Course

5. Managing Pods

CKAD Crash Course

5.1 Understanding Pods

What is a Pod?

- A Pod is an abstraction of a server
  - It runs one or multiple containers within a single name space, exposed by a single IP address
- The Pod is the minimal entity that can be managed by Kubernetes
- Typically, Pods are only started through a Deployment or ReplicaSet, because "naked" Pods are not rescheduled in case of a node failure

Managing Pods with **kubectl**

- Use **kubectl run** to run a *deployment* based on a default image
  - **kubectl run ghost --image=ghost:0.9**
- Use **kubectl** combined with instructions in a YAML file to do anything you'd like
- **kubectl create -f <name>.yaml**
- **kubectl get pods [-o yaml]**
- **kubectl describe pods** shows all details about a pod, including information about containers running within
  - For instance, **kubectl describe pods newhttpd**
- **kubectl edit pod mypod** allows editing of live pods

CKAD Crash Course

5.1 Creating a YAML Manifest to Configure Pods

Pearson

Using **kubectl** in a Declarative Way

- The recommended way to work with kubectl, is by writing your manifest files and using **kubectl {create|apply} -f manifest.yaml** to the current objects in your cluster
- This declarative methodology is giving you much more control than the imperative methodology where you create all from the CLI
  - Get current state of an object: **kubectl get deployments nginx --export -o yaml**
  - Push settings from a new manifest: **kubectl replace -f nginx.yaml**
  - Apply settings from a manifest: **kubectl apply -f nginx.yaml**

Basic YAML Manifest Ingredients

- **apiVersion:** specifies which version of the API to use for this object
- **kind**: indicates the type of object (Deployment, Pod, etc.)
- **metadata:** contains administrative information about the object
- **spec**: contains the specifics for the object
- Use **kubectl explain** to get more information about the basic properties

Container components

In the containers spec, different parts are needed

- name: the name of the container
- image: the image that should be used
- command: the command the container should run
- args: argument that are used by the command
- env: environment variables that should be used by the container

Example YAML File

```
apiVersion: v1
kind: Pod
metadata:
  name: mypod
  namespace: default
spec:
  containers:
  - name: busybox
    image: busybox
    command:
      - sleep
      - “3600"
  - name: nginx
    image: nginx
```

5.3 Understanding Multi-Container Pods

Understanding Multi-Container Pods

- Typically, just one container is offered through a pod
- Single container Pods are easier to build and maintain
- There are some cases where you might want to run multiple containers in a Pod
  - Sidecar container: a container that enhances the primary application, for instance for logging
  - Ambassador container: a container that represents the primary container to the outside world, such as a proxy
  - Adapter container: used to adopt the traffic or data pattern to match the traffic or data pattern in other applications in the cluster
- When using multi-container pods, the containers typically share date through shared storage

Understanding the Sidecar Scenario

- A sidecar container is providing additional functionality to the main container, where it makes no sense running this functionality in a separate Pod
- Think of logging, monitoring and syncing
- The essence is that the main container and the sidecar container have access to shared resources to exchange information
- Often, shared volumes (discussed in-depth later) are used for this purpose

Demo: Sidecar Example

- **kubectl create –f sidecar.yaml**
- **kubectl exec –it sidecar-pod –c sidecar /bin/bash**
- **yum install –y curl**
- **curl http://localhost/date.txt**

CKAD Crash Course

5.4 Monitoring Pods

Monitoring Pods

- Use **kubectl get pods** to get an overview of all pods and their current state
- More advanced Pod monitoring should happen from the point of view of application monitoring: don't just monitor the individual pod, but the deployment of which it is a part
- Advisor is a more advanced monitoring agent that integrated with the Kubelet to get performance data about Kubernetes components
- Prometheus is a more advanced monitoring solution that allows you to monitor pods as well as nodes

Exam Lab 3

Start a pod that runs an nginx webserver. Use the name nginx-lab. Find the fastest way possible to do this.

5.5 Exploring Pod Configuration with **kubectl describe**

Exploring Pod Configuration

- When deploying a Pod, many parameters are set to a default value
- Use **kubectl describe pod podname-xxx** to see all these parameters and their current setting
- Use documentation at https://kubernetes.io/docs for more information about these settings
- Tip! This documentation is available on the exam as well: use it!

Connecting to a Pod for Further Inspection

- Apart from exploring a Pod externally, you can also connect to it and run commands on the primary container in a pod:
  - **kubectl exec –it nginx-xxx -- sh**
  - From here, run any command to investigate
  - Use Ctrl-p, Ctrl-q to disconnect

Exam lab 4

Explore the logs of the pod you've created in the previous lab. Which volumes does it have mounted?

CKAD Crash Course

5.6 Exploring Pod Logs

Using Pod Logs

- Applications running in Pods are writing to the container logs
- Use **kubectl logs podname-xxx** to read these logs

CKAD Crash Course

5.7 Using Port-Forwarding to Access Pods

Using Port Forwarding to Access Pods

- Pods can be accessed in multiple ways
- A very simple way is by using port forwarding to expose a Pod port on the kubelet host
    - **kubectl apply –f nginx.yaml**
    - **kubectl port-forward pod/nginx 8080:80**
    - **curl http://localhost:8080**
- More advanced ways to access Pod processes are by using services and Ingress

CKAD Crash Course

5.8 Understanding Security Context

Understanding SecurityContext

- A security context defines privilege and access control settings for a pod or container, and includes the following:
  - Discretionary Access Control which is about permissions to access an object
  - Security Enhanced Linux, where security labels can be applied
  - Running as privileged or unprivileged user
  - Using Linux capabilities
  - AppArmor, which is an alternative to SELinux
  - AllowPrivilegeEscalation which controls if a process can gain more privileges than its parent process

Demo

- **kubectl apply –f securitycontextdemo.yaml**
- **kubectl get pods shows pending**
- **kubectl get pods nginxsecure –o yaml** (wait a few minutes)
- **kubectl explain pods.spec.securityContext**

5.9 Managing Jobs and Cron Jobs

Managing Jobs

- Pods normally are created to run forever
- To create a Pod that runs for a limited duration, use Jobs instead
- Jobs are useful for tasks, like backup, calculation, batch processing and more
- A Pod that is started by a Job must have its restartPolicy set to OnFailure or Never
  - OnFailure will re-run the container on the same Pod
  - Never will re-run the failing container in a new Pod

Understanding Job Types

3 different job types can be started, which is specified by the **completions** and **parallelism** parameters:

- Non-parallel Jobs: one Pod is started, unless the Pod fails
  - **completions=1**
  - **parallelism=1**
- Parallel Jobs with a fixed completion count: the job is complete after successfully running as many times as specified in jobs.spec.completions
  - **completions=n**
  - **parallelism=m**
- Parallel Jobs with a work queue: multiple jobs are started, when one completes successfully, the job is complete
  - **completions=1**
  - **parallelism=n**

Managing Cron Jobs

- Jobs are used to run a task a specific number of times
- CronJobs are used for tasks that need to run on a regular basis
- When running a Cronjob, a Job will be scheduled
- This job on its turn will start a pod

CKAD Crash Course

5.10 Managing Resource Limitations

Understanding Resource Limitations

- By default, a Pod will use as much CPU and memory as necessary to do its work
- This can be managed by using Memory/CPU Requests and Limits in pod.spec.containers.resources
- Memory as well as CPU limits can be used
- CPU Limits are expressed in millicore or millicpu, 1/1000 of a CPU core
  - So 500 millicore is 0.5 CPU
- Memory limits can be set also, and are converted to the --memory option that can be used by the docker run command (or anything similar)
- When being scheduled, the kube-scheduler ensures that the node running the pods has all requested resources available

Demo

- **kubectl create –f frontend-resources.yaml**

Exam Lab 6

Create a cronjob. It should send the message "hello world" to syslog every 5 minutes.

Exam Lab 7

Create a Pod that runs an Apache web server. It should not get access to more than 512 MiB of RAM

CKAD Crash Course

5.11 Understanding NameSpaces

Understanding Namespaces

- At a Linux level namespace implements kernel level resource isolation
- This limitation can be shared in Kubernetes as Kubernetes Namespaces
- Different namespaces can be used to strictly separate between customer resources
- Use **kubectl … -n namespace** to work in a specific namespace

Demo

- kubectl get all --all-namespaces
- kubectl create ns secret
- kubectl create –f busybox-ns.yaml

CKAD Crash Course

6. Managing Deployments

CKAD Crash Course

6.1 Understanding Deployment Features

Understanding Deployments

- When running Pods, additional features are needed:
  - Scalability
  - Updates and Update Strategy
  - and more
- That's why you don't run pods, you'll run deployments instead
- When creating an application from Dashboard, you'll create a deployment by default
- The deprecated **kubectl run** command was also used to create deployments

Understanding the Deployment Spec

In the deployment spec, specific properties are managed:

- replicas: explains how many copies of each pod should be running
- strategy: explains how Pods should be updated
- selector: uses matchLabels to identify how labels are matched against the Pod
- template: contains the pod specification and is used in a deployment to create Pods

6.2 Managing Deployment Scalability

Managing Scalability

- Use **kubectl scale deployment my-deployment --replicas=4** to scale the number of currently running replicas
- Alternatively, use **kubectl edit deployment my-deployment** to edit the number of replicas manually

Understanding ReplicaSets

- ReplicaSets can be used to manage scalability from outside a deployment
- From inside a deployment, the spec.replicas parameter is used to indicate the number of desired replicas
- In current Kubernetes, you'll get a ReplicaSet when creating the deployment
- Don't manage scalability through ReplicaSets, the replicas parameter while creating a deployment

Exam Lab 8

Create a deployment. Use the name nginx-lab8, and ensure this deployment runs 3 replicas of an nginx Pod.

CKAD Crash Course

6.3 Managing Deployment History

Understanding Deployment History

- Major changes cause the deployment to create a new replica set that uses the new properties
- The old replica set is still kept, but the number of Pods will be set to 0
- This makes it easy to undo a change
- **kubectl rollout history** will show the rollout history of a specific deployment, which can easily be reverted as well

6.4 Understanding Rolling Updates and Rollback

Understanding Rolling Updates

- It is the task of the deployment to ensure that a sufficient number of Pods is running at all times
- So when making a change, this change is applied as a rolling update: the changed version is deployed and after that has confirmed to be successful, the old version is taken offline
- You can use **kubectl rollout history** to get details about recent transactions
- Use **kubectl rollout undo** to undo a previous change

Understanding Update Strategy

When a deployment changes, the Pods are immediately updated according to the update strategy:

- Recreate: all Pods are killed and new Pods are created. This will lead to temporary unavailability. Useful if you cannot simultaneously run different versions of an application
- RollingUpdate: updates Pods one at a time to guarantee availability of the application. This is the preferred approach, and you can further tune its behavior

Using RollingUpdate Options

- maxUnavailable: determines the maximum number of Pods that is upgraded at the same time
- maxSurge: the number of Pods that is allowed to run beyond the desired number of pods as specified in a replica to guarantee minimal availability

6.5 Understanding Labels, Selectors and Annotations

Understanding Labels

- A label can be used as a key/value pair for further identification of Kubernetes resources
- This can be useful for locating resources at a later stage
- Labels are automatically set when running an app, and the name of the label is based on how the app was started
- Labels are also used by Kubernetes itself for pod selection by deployments and services
  - Deployments are monitoring a sufficient amount of Pods through the run label
  - When creating services using **kubectl expose**, a label is automatically added
- Use the selector option to search for items that have a specific label set
  - **kubectl get pods --selector='run=httpd'**

Understanding Annotations

- Annotations are used to provide detailed metadata in an object
- Annotations can not be used in queries, they are just to provide additional information
- Think of information about licenses, maintainer and more

Demo

MANUALLY SETTING

- **kubectl label deployment ghost state=demo**
- **kubectl get deployments --show-labels**
- **kubectl get deployments --selector state=demo**

AUTOMATED

- **kubectl run nginx --image=nginx**
- **kubectl describe deployment nginx** –> look for label
- **kubectl describe pod nginx-xxx**
- **kubectl label pod nginx-xxx run-** → will remove the auto-assigned run label and start a new pod to meet the requirements
- **kubectl get all --selector run=nginx**

Exam Lab 9

Find all pods that have been started with the **kubectl run** command

CKAD Crash Course

Lab: Managing Deployments

Exam Lab 10

Create a Cron Job that will run the sleep command for 30 seconds. If after 15 seconds, it still runs, kill it and start a new Cron job

CKAD Crash Course

7. Managing Networking

CKAD Crash Course

7.1 Understanding Pod Access Options

Understanding Pod Access Options

- port forwarding: exposes a port on the current node that runs the workload
- services: run a kube-proxy as a load balancer on all nodes involved to forward traffic to the appropriate node and uses 4 types:
  - ClusterIP: internal access through a local cluster IP only
  - NodePort: exposes a high port on a fixed cluster IP
  - LoadBalancer: cloud based entrypoint that is externally available to load-balance incoming traffic to the pods
  - externalName: redirects incoming traffic to DNS to further take care of the incoming requests
- ingress: adds scalability: one ingress controller can be used as access point to multiple services

7.2 Understanding Network Policies

Understanding Network Policies

- By default, all Pods can reach one another
- Network isolation can be configured to block traffic to Pods by running Pods in dedicated namespaces
- NetworkPolicy can be used to block egress as well as ingress traffic and works like a firewall
- Use of NetworkPolicy depends on support from the network provider: not all are offering support and it that case your network policy won't have any effect!
- Labels are used to define what the policy applies to
- See **kubectl explain NetworkPolicy.spec** for more details

Network Policy Usage

- Many examples are on https://github.com/ahmetb/kubernetes-network-policy-recipes
- Policies work with two directions: Ingress and Egress
- If a direction (Ingress or Egress) is listed in the manifest, but not further specified, there is no limitation
- If a direction is listed and contains a specification, that specification will be used
- See pods-with-nw-policy.yaml for an example

Exam Lab 11

Create a pod that runs two containers: nginx and httpd. Configure a network policy that makes that the nginx container cannot initiate any traffic to the httpd container.

CKAD Crash Course

7.3 Understanding Services

Understanding Services

- A service is an abstraction which defines a logical set of Pods and a policy by which to access them
- This together is also referred to as a micro-service
- The set of Pods that is targeted by a service is often determined by a selector (which is a label)
- The controller will continuously scan for Pods that match the selector and include these in the service
- This can be useful if communication needs to happen across namespaces

Understanding Services Decoupling

- Services exist independently from Deployments
- The only thing they do, is watch for a deployment that has a specific label set, based on the Selector that is specified in the service
- That means that one service can provide access to multiple deployments, and while doing so, Kubernetes will automatically load balance between these deployments

Services and Kube-proxy

- The kube-proxy agent on the nodes watches the Kubernetes API for new services and endpoints
- After creation, it opens random ports and listens for traffic to the clusterIP port, and next redirects traffic to the randomly generated service endpoints

Understanding Service Types

According to the needs in different environments, different service types are available:

- ClusterIP: the default type. Provides internal access only
- NodePort: allocates a specific node port which needs to be opened on the firewall
- LoadBalancer: currently only implemented in public cloud
- ExternalName: a relatively new object that works on DNS names. Redirection is happening at a DNS level
- Service without selector: use for direct connections based on IP/port, without an endpoint. Useful for connections to database, or between namespaces

CKAD Crash Course

7.4 Creating Services

Exposing Deployments

- Use **kubectl expose** to expose a current deployment
- **kubectl expose deployment nginx --port=80 --type=NodePort**
  - Notice this command allocates a random port on all backend nodes, optionally use **targetPort** to define the port that should be used
- **kubectl get svc** will show current services
- **kubectl get svc nginx –o yaml** will show service specifics in YAML
- According to the type that is exposed, you may need to pass more parameters while using **kubectl expose**, see --help for more details

Demo

- **kubectl expose nginx**
- **kubectl get svc nginx –o yaml**

CKAD Crash Course

7.5 Managing Service Manifest Files

Demo

- see service.yml
- Note that the essence is in the selector app: MyApp which will make this configuration applies to all objects that meet this selector
- Notice this is a clusterIP type, this clusterIP is used by the service proxies

CKAD Crash Course

7.6 Understanding Services and DNS

Understanding Services and DNS

- With services exposing themselves on dynamic ports, resolving service names can be challenging
- As a solution, a DNS service is included by default in Kubernetes and this DNS service is updated every time a new service is added
- So DNS name lookup from within one pod to any exposed service happens automatically

Demo

Demo:

1. Create busybox.yaml

```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 http**
5. You'll see the IP address being resolved, thus providing proof that DNS is working correctly

Exam lab 11

- Configure a Service for the Nginx deployment you've created earlier. Ensure that this service makes Nginx accessible through port 80, using the ClusterIP type. Verify that works
- After making the service accessible this way, change the type to NodePort and expose the service on port 32000
- Verify the service is accessible on this node port

CKAD Crash Course

8. Managing Ingress

CKAD Crash Course

8.1 Understanding Ingress

Understanding Ingress

- Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster
- Traffic routing is controlled by rules defined on the Ingress resource
- Ingress can be configured to do the following
  - Give Services externally-reachable URLs
  - Load balance traffic
  - Terminate SSL/TLS
  - Offer name based virtual hosting
- An Ingress controller is used to realize the Ingress

Understanding Ingress Controllers

- Creating Ingress resources without Ingress controller has no effect
- Many Ingress controllers exist:
  - nginx: https://kubernetes.github.io/ingress-nginx/
  - haproxy: https://www.haproxy.com/blog/dissecting-the-haproxy-kubernetes-ingress-controller/
  - traefik: https://docs.traefik.io
  - kong: https://konghq.com/solutions/kubernetes-ingress/
  - contour: https://octetz.com/posts/contour-adv-ing-and-delegation
- To enable the minikube addon, use **minikube addon enable ingress**

CKAD Crash Course

8.2 Configuring Ingress

Sample Ingress Manifest

```yaml
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: test-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - http:
    paths:
    - path: /testpath
      backend:
        serviceName: test
        servicePort: 80
```

Demo - 1

- see https://kubernetes.io/docs/tasks/access-application-cluster/ingress-minikube/
- **minikube addons enable ingress**
- **kubectl get pods –n kube-system** look for ingress, will take few minutes
- **kubectl run web --image=gcr.io/google-samples/hello-app:1.0 --port=8080**
- **kubectl expose deployment web --target-port=8080 --type=NodePort**
- **kubectl get service web**
- **minikube service web --url**
- **curl http://that-ip:that-port**
- **vim kubernetes/example-ingress.yaml** (see git repo)

Demo - 2

- **kubectl apply –f kubernetes/example-ingress.yaml**
- **kubectl get ingress** (make take a few minutes – look for the IP address). Note that this is the *internal* ingress IP address, which by default is not routable in minikube. When using minikube locally, use the **minikube ip** address in the following step
- In /etc/hosts: **172.17.0.15 hello-world.info**
- **curl hello-world.info**

CKAD Crash Course

8.3 Configuring Ingress Rules

Understanding Ingress Rules

- Each Ingress Rule contains the following:
- An optional host. If no host is specified, the rule applies to all inbound HTTP traffic
- A list of paths (like /testpath). Each path has its own backend
- The backend, which consists of a serviceName and servicePort. These match K8s API service objects. It is common to configure a default backend in an ingress controller for incoming traffic that doesn't match a specific path

Understanding Backend Types

- In the previous examples, a single Ingress Backend was used. Other options are available:
- Simple fanout: traffic is routed to multiple backends, which allows you to minimize the number of load balancers
- Name based virtual hosting: traffic incoming on a specific name is routed to a specific service
- TLS Ingress: Uses a TLS secret to ensure TLS termination to be happening at the load balancer

Ingress Fanout Example

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - path: /foo
        backend:
          serviceName: service1
          servicePort: 4200
…
```

Ingress Virtual Hosting Example

```
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
spec:
  rules:
  - host: first.bar.com
    http:
      paths:
      - backend:
          serviceName: service1
          servicePort: 80
  - host: second.bar.com
    …
```

CKAD Crash Course

9. Managing Storage

CKAD Crash Course

9.1 Understanding Storage Options

Understanding Storage Options

- Files stored in a container will only live as long as the container itself
- Pod Volumes can be used to allocated storage that outlives a container and stays available during pod livetime
- A persistent volume allows pods to connect to external storage and can exist outside of Pods as well
- To use persistent volumes, persistent volume claims are used to request access to specific storage
- ConfigMaps are specific objects that connect to configuration files
- Secrets do the same, but by encoding the data they contain

CKAD Crash Course

9.2 Configuring Volume Storage

Configuring Volumes

- First, decide if you want to use a regular volume or a persistent volume
- To create a regular volume, the Pod needs to define the volume in spec.volumes.
- Next, the container mounts that volume in spec.containers.volumemounts
- For use of persistent volume, additional external objects are needed

Demo

1. **kubectl create -f morevolumes.yaml**
2. **kubectl get pods morevol**
3. **kubectl describe pods morevol | less** ## verify there are two containers in the pod
4. **kubectl exec -ti morevol -c centos -- touch /centos/test**
5. **kubectl exec -ti morevol -c centos2 -- ls -l /centos2**

Understanding Volume Types

- Many volume types are supported
  - emptyDir
  - azureDisk
  - cephfs
  - awsElasticBlockStore
  - fc: fibrechannel
  - gcePersistentDisk: Google cloud
  - iscsi
  - nfs
  - rbd
  - gitrepo
  - hostPath: connects to host in minikube environment
- And more: choose what fits your specific needs

CKAD Crash Course

9.3 Configuring PV Storage

Understanding Persistent Volumes

- A Persistent Volume is a storage abstraction that is used to store persistent data
  - Use **persistentVolume** to define it
  - Different types (NFS, iSCSI, CephFS and many more) are available
- Using claims with Persistent Volumes creates portable Kubernetes storage that allow you to use volumes, regardless of the specific storage provider
  - Use **persistentVolumeClaims**
  - The persistent volume claim talks to the available backend storage provider and dynamically uses volumes that are available on that storage type
- Kubernetes will use Persistent Volumes according to the availability of the requested volume accessModes and capacity

CKAD Crash Course

9.4 Configuring PVCs

Understanding Persistent Volume Claims

- The Persistent Volume Claims requests access to storage provided by Persistent Volumes according to specific properties
  - accessModes
  - availability of resources
- The persistent volume sets a name to the PVC, which can be used in the Pod which is accessing the storage

CKAD Crash Course

9.5 Configuring Pod Storage

CKAD Crash Course

Lab: Setting up Storage

Lab: Setting up Storage

- Configure a PV that is using HostPath storage
- Set up the storage accessMode such that only multiple pods may access the storage at the same time
- Configure a Pod that runs an httpd web server to mount the default web server documentroot /var/www/html on the HostPath storage

CKAD Crash Course

10. Managing ConfigMaps and Secrets

10.1 Understanding ConfigMaps

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

CKAD Crash Course

10.2 Creating ConfigMaps

Procedure Overview

- Start by defining the ConfigMap and create it
  - Consider the different sources that can be used for ConfigMaps
  - **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>**
- Note that it doesn't really matter from which the ConfigMap is created, usage from within the Pod will be the same:

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

- Next, define the Pod that is using it

Using ConfigMaps for Config Files

- ConfigMaps can also be used to provide complete configuration files
- In this, the Pod is using the ConfigMap as a mounted volume which will just contain the configuration file
- And the ConfigMap itself contains the configuration file

CKAD Crash Course

10.3 Understanding Secrets

Understanding Secrets

- Secrets allow for storage of sensitive data such as passwords, Auth tokens and SSH keys
- Using secrets makes that the data doesn't have to be put in a Pod, and reduces the risk of accidental exposure
- Some secrets are automatically created by the system, users can also use secrets
- Secrets are used by Pods as files in a volume, or used by kubelet when pulling images for the pod

Understanding Secret Types

- Three types of secret are offered
  - docker-registry: used for connecting to a Docker registry
  - TLS: creates a TLS secret
  - generic: creates a secret from a local file, directory or literal value

Understanding Built-in Secrets

- Kubernetes automatically creates secret that contain credentials for accessing the API, and automatically modifies the pods to use this type of secret
- Secrets are not encrypted, they are encoded
- Use **kubectl describe pods <podname>** and look for the mount section to see it

CKAD Crash Course

10.4 Creating Secrets

Creating Your Own Secrets

- Creating secrets is very similar to creating ConfigMaps
- Use **kubectl create secret … --from-file=…** to create it from a file
- Use **kubectl create secret … --from-literal=…** to create it from a string provided on the command line

Demo

- **kubectl create secret generic my-secret --from-file=ssh-privatekey=/home/student/.ssh/id_rsa --from-literal=passphrase=password**
- **kubectl describe secret my-secret**

Creating Secrets from YAML Files

- When creating Secrets from YAML files, they must first be encoded using the **base64** command
- Note that *encoded* is not the same as *encrypted*, users can easily decode using **echo <string> | base64 -d**

Demo

- **echo –n 'lisa' | base64 > secret-yaml.yaml**
- **echo –n 'password' | base64 >> secret-yaml.yaml**
- **cat secret-yaml.yaml**
- **kubectl create –f secret-yaml.yaml**
- **kubectl explain secret** # notice the differences between data: and stringData:

CKAD Crash Course

10.5 Configuring Pods to Use Secrets

Using Secrets

- Secrets can be used by Pods in two ways:
  - As environment variables
  - Mounted as volumes

Demo: mounting secrets as volumes

- **kubectl create secret generic secretstuff --from-literal=password=password --from-literal=user=linda**
- **kubectl create –f pod-secret.yaml**
- **kubectl describe secretbox2**
- **kubectl exec –ti secretbox2 /bin/sh**
- **cat /secretstuff/password**

Demo: secrets as variables

- **kubectl create secret generic mysql --from-literal=password=root**
- **kubectl get secret mysql –o yaml**
- continue with pod-secret-as-var.yaml
- show how to get there, using **kubectl explain pod.spec.containers.env.valueFrom**
- **kubectl create –f pod-secret-as-var.yaml**
- **kubectl exec –ti mymysql /bin/bash**
- **env**

CKAD Crash Course

11. Troubleshooting

CKAD Crash Course

11.1 Generic Tips

Troubleshooting Procedure

- Verify syntax before adding anything
- Use standard Linux tools for troubleshooting
  - Consider starting busybox if tools are absent
- Use **kubectl describe** to see what happens
- Use **kubectl logs** to see what a container in a Pod is doing
- Consider limitations: logging activity across all nodes is not part of Kubernetes and provided by external projects like Fluentd or Prometheus
- In this lesson we'll focus on **kubectl** commands that can be used

Verifying Syntax

- Without Internet Access: use **kubectl explain**
  - **kubectl explain** explains API objects
  - To get information about a specific part of the object, use dotted notation: **kubectl explain pod.spec**
- With Internet Access: use https://kubernetes.io/docs
  - Use the Search feature for examples
  - Most of the useful examples are in the Tasks section of the Docs
  - You are allowed to access kubernetes.io/docs during the exam

Demo

- This demo is mainly about explain and log
- Clean up previously used cm objects and the test1 pod that was created from cm-test-pod.
- edit **cm-test-pod.yaml** and remove restartPolicy: Never
- Use **kubectl create cm variables --from-file=variables**
- Use **kubectl create –f cm-test-pod1.yaml**
- Use **kubectl get pods**: you'll see a Crashloopback failure
- Use **kubectl describe pods test1** and scroll through the output
- Use **kubectl logs test1** to see that the VAR1 and VAR2 are in the list of the logs – which basically means it was successful
- Use **kubectl explain pod.spec**, scroll through options and look for restartPolicy
- Apply restartPolicy to fix

CKAD Crash Course

11.2 Using Probes

Understanding Probes

- Probes can be used to test access to Pods
- A readinessProbe is used to make sure a Pod is not published as available until the readinessProbe has been able to access it
- The livenessProbe is used to continue checking the availability of a Pod
- A probe is a simple test that verifies that the application is reacting

Understanding probe Types

- command: a command is executed and returns a zero exit value
- HTTP request: an HTTP request returns a response code between 200 and 399
- TCP socket: connectivity to a TCP socket (available port) is successful

Demo

- **kubectl create –f busybox-ready.yaml**
- **kubectl get pods** note the READY state, which is set to 0/1, which means that the pod has successfully started, but is not considered ready.
- **kubectl edit pods busybox-ready** and change /tmp/nothing to /etc/hosts. Notice this is not allowed.
- **kubectl exec –it busybox-ready -- /bin/sh**
    - **touch /tmp/nothing; exit**
- **kubectl get pods** at this point we have a pod that is started

Demo 2

- kubectl create –f nginx-probes.yaml
- kubectl get pods

CKAD Crash Course

11.3 Understanding Application Monitoring

Understanding Application Monitoring

- The goal of monitoring is to collect metrics about the infrastructure
- Kubernetes monitoring happens through external CNCF projects
- Heapster is a part of Kubernetes and can be used for monitoring multiple pods
- Prometheus is available as a Kubernetes plugin and provides cross cluster usage metrics
- Logging is provided by the Fluentd project, which can be used to aggregate logs