Kubernetes in 4 Hours

Sander van Vugt

Pearson

Agenda

- Understanding Kubernetes
- Kubernetes Installation and Configuration
- Running Applications in Pods and Containers
- Exposing Applications using Services
- Using Volumes to Provide Storage
- Using ConfigMaps to decouple site specific information from code

Expectations

- This class is for people new to Kubernetes
- I'll teach you how to get started and deploy applications on Kubernetes
- Don't expect much information about advanced topics
- For more in-depth information consider one of the following classes
  - CKAD is a 3 days class that prepares for the CKAD exam
  - Kubernetes in 3 weeks covers all important concepts in a 3 weeks program
  - Managing Microservices with Kubernetes and Istio is a 5 hours class in which you'll learn how to use Kubernetes to build Microservices
  - GitOps and Kubernetes Automation in 3 Weeks explains how to use GitOps to work with Kubernetes more efficiently

Poll question 2

- How would you rate your knowledge about containers
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5

Poll question 3

- How would you rate your Kubernetes knowledge and experience?
  - 0
  - 1
  - 2
  - 3
  - 4
  - 5

Poll question 4

- Where are you from?
  - India
  - Asia (other countries)
  - Africa
  - North or Central America
  - South America
  - Europe
  - Australia / Pacific
  - Netherlands

Kubernetes in 4 Hours

What is Kubernetes?

What is Kubernetes?

- Kubernetes is a platform for running container-based cloud-native applications
- It offers different resources that allow for storing information in the cloud instead of on a local host
- It offers enterprise features like scalability and availability
- It orchestrates containers in such a way that they are providing the services that are required in the environment where these services are required
- The solution is based on the Borg technology that Google has been using for many years in their datacenters

What are Containers?

- A container image includes all dependencies required to run an application
- Containers are running instances of container images
- To run a container, a container engine is required. Container engines run on top of a host operating system
- Docker and Podman are common solutions for running containers on stand-alone computers
- Kubernetes adds cluster features to containers by managing them in pod resources

Container needs in Datacenter and Cloud

- Storage that is not bound to any specific physical environment
- A cluster of hosts to run the containers
- Monitoring and self-healing of containers
- A solution for updates without downtime
- A flexible network that can self-extend if that is needed

About the Kubernetes Host Platform

Kubernetes can be offered through different host platforms

- As a hosted service in public cloud
- On top of a physical cluster (on premise)
- As an all-in-one solution, running on Minikube

CNCF: Standardization on K8s

- Cloud Native Computing Foundation (CNCF) is a governing body that solves issues faced by any cloud native application (so not just Kubernetes)
- Google donated Kubernetes to the Cloud Native Computing Foundation, which is a foundation in Linux Foundation
- CNCF owns the copyright of Kubernetes

Kubernetes and the Ecosystem

- CNCF hosts many cloud native projects and Kubernetes is just one of them
- In Kubernetes installations, other CNCF projects are included:
  - Network Plugins
  - Storage Provisioners
  - Ingress and more
- Distributions bundle Kubernetes with other CNCF projects to get a completely working environment

Kubernetes Distributions

- Kubernetes is the open-source standard for orchestrating containers, competing products include
  - Amazon ECS
  - Docker Swarm
  - Apache Nomad
  - Amazon Fargate
- Common Kubernetes distributions include
  - Rancher
  - Red Hat OpenShift
  - Google Anthos
  - Public cloud distributions like EKS, AKS and GKS

Kubernetes in 4 Hours

Installing a Kubernetes Test Cluster

Kubernetes Usage Options

- There are many options
  - Minikube
  - Cloud based
  - Docker Desktop
  - O'Reilly Sandbox
- Demo'ing in this course: minikube

Minikube Overview

- Minikube offers a complete test environment that runs on Linux, MacOS or Windows
- Other test environments can also be used
- In all cases, you'll need to have the **kubectl** client on your management platform

Installing Minikube

- A scripted installation is provided for Ubuntu 20.04 and later
- Install either of these with at least 4 GB RAM and 20 GB disk space (8 GB and 40GB recommended)
- Use **git clone https://github.com/sandervanvugt/kubernetes**
- From there, use the **minikube-docker-setup.sh** script and follow instructions

Minikube on WSL2

- Install WSL2
- Install Docker Desktop (https://docs.docker.com/desktop/install/windows-install/)
- Use **wsl -l -v** to confirm your Ubuntu is WSL v2
- If not, from a root powershell, use **wsl --set-version Ubuntu-22.04**
- Run the **setup-docker-minikube.sh** script
- Start it, using **minikube start --vm-driver=docker --cni=calico**

Running Your First Application

- From **minikube dashboard**, click +CREATE in the upper right corner
- Specify **httpd** as the container image as well as the container name
- This will pull the container image and run it in the minikube environment

Kubernetes in 4 Hours

Accessing and Using the Cluster

Managing 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 offers a web based interface

Using kubectl

- The **kubectl** command is the generic command that allows you to manage all aspects of pods and containers
- Use **kubectl create** to create deployment
- Or **kubectl get ...** or one of the many other options to get information about pods
- Start with **kubectl completion -h**

Kubernetes in 4 Hours

Understanding Kubernetes Resource Types

Understanding Main Kubernetes Resource Types

- *Pods*: the basic unit in Kubernetes, represents one or more containers that share common resources
- *Deployments*: the application itself, standard entity that is rolled out with Kubernetes
- *Services*: make deployments accessible from the outside by providing a single IP/port combination.
- *Persistent Volumes*: persistent (networked) storage
- *ConfigMaps*: Allow for storing configuration and other specific parameters in a cloud environment

Understanding the Pod

- Kubernetes manages Pods, not containers
- The Pod is a Kubernetes resource, defined in the Kubernetes API to provide features required for managing containers in a clustered environment
- Containers can be put together in a Pod, together with Pod-specific storage, but a typical pod runs one container only

Understanding the Deployment

- To run applications in Kubernetes, create deployments
- A deployment is adding scalability as well as zero-downtime upgrades to Pods
- Do **NOT** run standalone Pods, run deployments only

Kubernetes in 4 Hours

Managing Applications with **kubectl**

Managing applications with kubectl

- Use **kubectl create deploy ...** to run an application
  - **kubectl create deploy mynginx --image=nginx --replicas=3**
- Use **kubectl get** to get information about running applications
  - **kubectl get all**
  - **kubectl get pods**
  - **kubectl get all --selector app=mynginx**
- Use **kubectl describe** to get information about resource properties
  - **kubectl describe pod mynginx-aaa-bbb**

Using kubectl in a declarative way

- To work with Kubernetes the DevOps way, you should define the desired configuration in a YAML manifest file
- 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 -o yaml**
  - Push settings from a new manifest: **kubectl create -f nginx.yaml**
  - Apply settings from a manifest: **kubectl apply -f nginx.yaml**

Creating YAML Files

- YAML files are used in declarative way
- Don't write them from scratch, generate them
- Use **kubectl create deploy mynginx --image=nginx --dry-run=client -o yaml > mypod.yaml** to easily generate a YAML file
- Use **kubectl explain** for more information about properties to be used in the YAML files
- Consult kubernetes.io/docs for many examples!

Understanding Namespaces

- Namespaces create isolated environments for running applications
- Use namespaces to create virtual datacenters
- Kubernetes core services run in the **kube-system** namespace
- Role Based Access Control (RBAC) can be used to delegate administrator / user privileges to a namespace
- Quota can be used to restrict resources in a namespace

Troubleshooting Kubernetes Applications

- **kubectl describe pod ...** is showing cluster information about Pods and should be the first thing to troubleshoot Kubernetes workloads
- **kubectl logs** is giving access to the Pod application STDOUT, which allows you to see what is going on in an application
- **kubectl get pods podname -o yaml** shows detailed information about what is going on in a Pod
- **kubectl exec -it PODNAME -- /bin/sh** gives access to a shell running within a Pod

Demo: Troubleshooting Applications

- **kubectl create deploy mydb --image=mariadb --replicas=3**
- **kubectl describe pod mydb-aaa-bbb**
- **kubectl logs mydb-aaa-bbb**
- **kubectl set env deploy/mydb MARIADB_ROOT_PASSWORD=secret**

Kubernetes in 4 Hours

Accessing applications from Outside

Understanding Pod Access

- Pods are connected to the pod network. The pod network is behind the firewall, and its addresses cannot be directly accessed
- As typically multiple instances of pods are started, a load balancer is needed to connect incoming user requests to a specific pod
- This API-based load balancing functionality is offered by the *Service* resource
- The service provides one single IP-address that should be addressed to connect to a specific pod

Understanding Ingress

- *Ingress* is an additional resource, that provides external access to HTTP and HTTPS based services
- *Ingress* also defines a virtual service name to provide easy access to services
- Different Ingress solutions are provided by the Kubernetes ecosystem

Understanding Service Types

- **ClusterIP** is accessible from within the cluster only
- **NodePort** exposes an external port on the cluster nodes, thus providing a primitive way for offering access to the services
- *Ingress* is what should be used to provide user-friendly access to services

Demo: Using Services - 1

- **kubectl create deployment nginxsvc --image=nginx**
- **kubectl scale deployment nginxsvc --replicas=3**
- **kubectl expose deployment nginxsvc --port=80**
- **kubectl describe svc nginxsvc** # look for endpoints
- **kubectl get svc**
- **kubectl get endpoints**

Demo: Using Services - 2

- **minikube ssh**
- **curl http://svc-ip-address**
- **exit**
- **kubectl edit svc nginxsvc**

  …

  protocol: TCP

  nodePort: 32000

  type: NodePort
- **kubectl get svc**
- (from host): **curl http://$(minikube ip):32000**

Kubernetes in 4 Hours

Working with Storage

Understanding Container Storage

- Container storage by nature is ephemeral
- To provide persistent storage, the Pod specification can define containers as well as volumes
- The pod volumes can be used to refer to any type of storage
- Use **kubectl explain pod.spec.volumes** to see which volume types are supported

Demo

1. **kubectl create -f morevolumes.yaml**
2. **kubectl get pods morevol2**
3. **kubectl describe pods morevol2 | less** ## verify there are two containers in the pod
4. **kubectl exec -ti morevol2 -c centos1 -- touch /centos1/test**
5. **kubectl exec -ti morevol2 -c centos2 -- ls -l /centos2**

Kubernetes in 4 Hours

Using ConfigMaps

Understanding ConfigMaps

- In a cloud-native environment, a solution must be provided to store site-specific data
- Storing configuration files, variables and startup parameters inside the pod specification would make it less portable
- ConfigMaps allow for storing site-specific information in dedicated API resources
- By storing site-specific information in ConfigMaps, you can keep the Pod and Deployment specifications generic
- ConfigMaps are commonly used for storing variables and Configuration Files
- Secrets are base64 encoded ConfigMaps

Creating ConfigMaps - 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=cow --from-literal=VAR4=goat**
  - 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

Demo: Creating ConfigMaps for Variables

- Create a deployment: **kubectl create deploy mynewdb --image=mysql --replicas=3**
- Use **kubectl get pods --selector app=mynewdb** to see that the pods in the new deployment are failing
- Create a ConfigMap: **kubectl create cm mynewdbvars --from-literal=MYSQL_ROOT_PASSWORD=password**
- Check the contents of the ConfigMap: **kubectl describe cm mynewdbvars**
- Apply it: **kubectl set env --from=configmap/mynewdbvars deploy/mynewdb**
- Use **kubectl get all --selector app=mynewdb** to see what is happening

Kubernetes in 4 Hours

Summary

Next Steps

- To learn more, consider one of the following live courses
  - Kubernetes in 3 weeks
  - GitOps and Kubernetes Automation in 3 weeks
  - CKAD Crash Course
  - CKA Crash Course
  - Managing MicroServices with Kubernetes and Istio
- Or one of the following recorded courses
  - Getting Started with Kubernetes3/ed
  - Hands on Kubernetes
  - Certified Kubernetes Application Developer
  - Certified Kubernetes Administrator