Building Microservices with Containers

Introduction

Agenda

- Understanding Microservices
- Micro Introduction to Containers
- Understanding Kubernetes
- Creating Container Based Microservices

Poll Question 1

Rate your knowledge/experience about containers

- none
- minimal
- just attended a basic course
- some working experience
- good working experience
- lots of working experience

Poll Question 2

Rate your knowledge/experience about kubernetes

- none
- minimal
- just attended a basic course
- some working experience
- good working experience
- lots of working experience

Poll Question 3

Which part of the world are you from?

- Europe
- Netherlands
- Africa
- North/Central America
- South America
- India
- Asia
- Australia/Pacific

Course Setup

- No setup is needed, just listen and learn and play later
- If you want to follow along:
  - Any running environment with Kubernetes will do (use cloud hosted for easy access, or minikube for local access)
  - Course github repository is available at https://github.com/sandervanvugt/microservices

Building Microservices with Containers

1. Understanding Microservices

Understanding the goals of this course

- In this course you'll learn how to build Microservices using containers
- We'll explore the devops cycle, starting with containers and their workings, and ending in the use of Microservices in the Kubernetes orchestration platform
- This course does NOT focus on the programmatic parts of microservices, so knowledge of specific programming languages is not required
- The main goal in this course is to teach students how to build a microservices based infrastructure based on orchestrated containers, where the focus is on elements provided by Kubernetes to implement *decoupling*

What are Microservices?

- Microservices are grains of application code that run minimal functionality that can be isolated from other grains of application code
- The different grains are loosely coupled to eachother
- The different grains are independently developed and maintained

Understanding Containers and Microservices

- A container is an application that runs based on a container image.
- The container image is a lightweight standalone executable package of software that includes everything that is needed to run a application
- Because containers are lightweight and include all dependencies required to run an application, containers have become the standard for developing applications
- As containers are focusing on their specific functionality, they are perfect building blocks for building microservices

How to Break up Monolithic

- The transition to Microservices involves breaking up old monolithic applications
- In this process, several changes are often applied
  - Connection parameters to database and middleware need to be changed from hard-coded to variables that can be managed in a flexible way
  - In web applications, application calls need to be changed to public DNS host names
  - Security needs to be modified, to ensure that one application is allowed to access other applications. This is known as cross-origin resource sharing
- Kubernetes facilitates breaking up monolithic applications

Understanding the role of the API

- In Microservices, different components need to be able to communicate
- To do so, APIs can be used
- An API defines how programs request access to services from either operating systems or other applications
- RESTful API is common
- gRPC is another API that is said to be more efficient

The Role of CI/CD in Microservices

- In a microservices oriented way of working, it is important to implement changes in an easy and non-disruptive way
- CI/CD is used to guarantee this can happen. The code typically originates in a Git repository, from which it can be deployed by using automation solutions like Dockerfile or OpenShift, which on its turn are hosted in a Git repository as well
- By hosting everything in Git repositories, it's easy to manage development and version differences

2. A Micro-introduction to Containers

Containers Defined

- A container is an application with all of its dependencies included
- To run a container, a container runtime is needed. This is the layer between the host operating system and the container itself
- Different solutions exist to run containers
  - Docker: started the container revolution in 2014 and is de facto standard
  - LXC: Linux native containers
  - Podman: standard in Red Hat platforms since RHEL 8

Understanding Container Types

- System containers provide a base operating system that can be used as foundation for building an application container
- Application containers are used to start just one application and are used to replace legacy applications
- Orchestration solutions like Kubernetes focus on managing application containers and may become confused while working with system images as they don't have a default application to start

Understanding a Perfect Match

- Microservices focus on developing minimal pieces of code and joining them
- Containers are focussing on developing minimal running application components
- So both have the same objective in mind
- The only thing that needs to be added, is a layer that connect containers together
- This is going to be done by the container orchestration layer

Containers and Images

- A container is a running instance of an image
- The image contains the application code, language runtime and libraries
- External libraries such as libc are typically provided by the host operating system, but in a container is included in the images
- The container image is a read-only instance of the application that just needs to be started
- While starting a container, it adds a writable layer on the top to store any changes that are made while working with the container

3. The Role of Container Orchestration

Understanding Enterprise Container Requirements

- Containers are based on the Container Runtime, a solution that has nothing to connect multiple containers together in a Microservice architecture
- Containers need to be distributed, scheduled and load balanced
- Apart from that, High Availability is required
- As well as an easy solution to provide updates without downtime
- To run containers as microservices, additional platform components are required as well, such as software defined networking and software defined storage

Understanding Microservices Platform Requirements

- To understand Microservices platform requirements, you need to understand the typical Microservices application
- Microservices applications typically drill down to a database part and an accessibility part, using REST or Web to provide accessibility

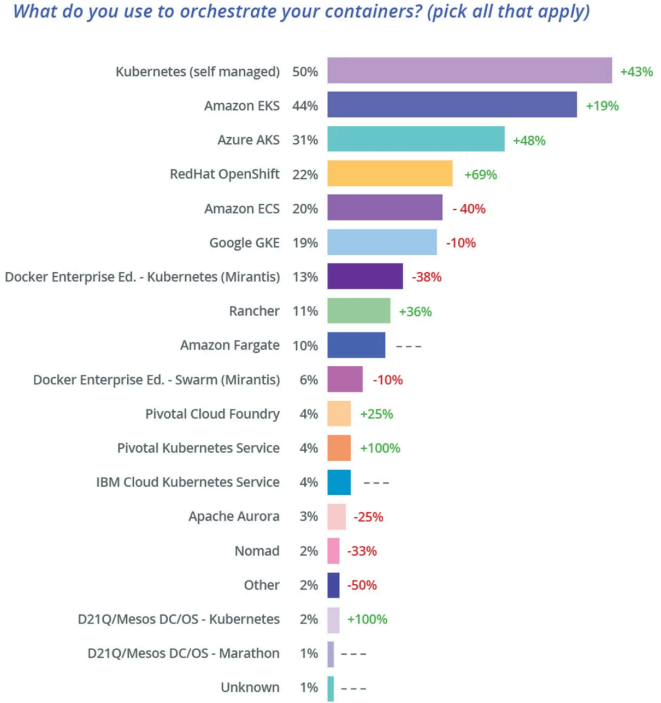
From Microservice to Platform

- Based on the Microservice overview, the following minimal platform requirements can be defined
  - Different databases must be running as connected applications
  - Front-end services need to be added to that, and add an accessibility layer as well
  - At the user side, accessibility must be added to different types of user requests
- Add some scalability to this, as well as some high availability, and you'll have the basic platform requirements
- And optionally, add integration of a CI/CD pipeline as well to make the cycle from source code to application complete

Understanding the Orchestration Landscape

- Kubernetes is the leading solution and the open source upstream for many other platforms
- Docker Swarm was developed by Docker Inc and offered as Docker Enterprise - now a part of Mirantis
- Red Hat OpenShift is based on the OpenShift Kubernetes Distribution (OKD) and offers Kubernetes with strong developed CI/CD

Orchestration platforms overview



https://www.stackrox.com/kubernetes-adoption-security-and-market-share-for-containers/

Understanding the Leading Position of Kubernetes

- Kubernetes is the leading technology that is in nearly all container orchestration platforms
- This is because of its origins, coming from Google Borg
- Google donated the Kubernetes specifications to the open source community after running Borg internally for over a decade
- As a result, a free, stable and open source project was introduced, where all vendors could base their own solution upon
- This rocked the world of container orchestration, and resulted in Kubernetes being the only relevant platform for orchestration

Understanding Kubernetes Delivery Options

- Kubernetes as a managed service in public and private cloud
  - Amazon EKS
  - Google Cloud Platform GKE
  - Azure AKS
  - OpenStack Magnum
- Kubernetes as an on-premise installation using a Kubernetes distribution
- Kubernetes as a test-drive platform, mainly developed to learn Kubernetes

4. Understanding Kubernetes

Getting Started with Minikube

- Minikube is an all-in-one Kubernetes virtual machine
- Using Minikube is recommended as it gives access to all Kubernetes features, without any limitations
- In this course we'll use Minikube on an Ubuntu virtual or physical machine
- Minikube can also be installed directly on top of MacOS or Windows
- For instructions on setting up Minikube on other platforms, got to: https://kubernetes.io/docs/tasks/tools/install-minikube/
- To do the hands on parts in the next lessons, it's fine to use any other Kubernetes platform as well

Installing Minikube

- Install the most recent version of Ubuntu Workstation/Fedora, using 40GB hard disk space, 8GB of RAM (more is recommended), and 2 CPU cores with embedded virtualization enabled
- Notice that Oracle Virtual Box does NOT support embedded virtualization for AMD processors
- Use **git clone** https://github.com/sandervanvugt/microservices to access the GitHub repository for this course
- Run the **kube-setup.sh** script from the GitHub repository and follow the instructions it provides
- currently not supported in a VM on MacOS Big Sur

Key Resource Types in Kubernetes

- Pod: the resource that runs the containers and adds cluster properties to contaienrs
- Deployment: equivalent to the applications, offers services by running scaled Pods
- Replicaset: part of the deployment that takes care of the replication
- Service: load balancer that provides access to scaled pods
- Ingress: delivers a URL to provide external access to the application
- PersistentVolume/StorageClass: provides access to storage
- PersistentVolumeClaim: allows a Pod to connect to storage without need to know about storage specifics

Running Containers in Kubernetes

- Do NOT run naked pods unless if its for testing
- **kubectl run** will run a pod
- **kubectl create deployment** will create a deployment

Understanding Decoupling in Microservices

- In microservices, it's all about independent development cycles, allowing developers to focus on their chunk of code
- To make reusing code easy, separation of static code from dynamic values is important
- This approach of decoupling items should be key in your Microservices strategy
- Kubernetes helps, by adding many resource types that focus on decoupling

Running Containers in Pods and Deployments

- To run containers, the *imperative* approach can be used
  - **kubectl create deployment**
  - **kubectl run**
- Alternatively, the *declarative* approach can be used, where specifications are done in a YAML file
- The declarative approach is preferred in a DevOps environment, as versions of YAML manifests can easily be maintained in Git repositories

Demo: Exposing Applications Using Services

- **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 nginx -o=yaml**
- **kubectl get svc**
- **kubectl get endpoints**

Demo: Accessing Deployments by Services

- **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**

Building Microservices with Containers

5. Creating Container Based Microservices

Decoupling Resource Types in K8s

- ConfigMap: used for storing variables and config files
- Secrets: used for storing variables in a protected way
- PersistentVolumes: used to refer to external storage
- PersistentVolumeClaims: used to point to the storage type you need to use

Understanding ConfigMaps

- The goal of the ConfigMap is to separate configuration from code
- ConfigMaps are clear-text
- They can be used in three different ways:
  - Make variables available within a Pod
  - Provide command line arguments
  - Mount them on a 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

Procedure Overview: Creating ConfigMaps

- Start by defining the ConfigMap and create it
  - Consider the different sources that can be used
  - **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 arguments

Demo: Creating a ConfigMap from a File

Files are provided in the GitHub repository

- Consider the contents of the file **variables**
- Create the ConfigMap: **kubectl create cm variables --from-env-file=variables**
- Verify creation: **kubectl describe cm variables**
- Create a Pod: **kubectl create -f cm-vars.yaml**
- Check that the variables are available: **kubectl logs po/cm-vars**

Configuring Storage

- The Pod specification contains a volumes part
- It's possible to directly point to the storage to be used, but this is deprecated as it doesn't separate code from site specifics
- For that reason, storage is referred to in PersistentVolumes or StorageClass (which is site default persistent storage)
- Containers are using PersistentVolumeClaim to bind to available storage

Understanding Kustomize

- **kustomize** is a Kubernetes feature that makes it easy to apply changes to existing resources, or as a template to create new resources
- Use **kubectl apply -k ./** in the directory with the **kustomization.yaml** and the files it refers to to apply changes
- Use **kubectl delete -k ./** in the same directory to delete all that was created by the Kustomization

Demo: Using Kustomization.yaml

- **cat deployment.yaml**
- **cat service.yaml**
- **cat kustomization.yaml**
- **kubectl apply -k ./**

Building Microservices with Containers

6. Bringing it all together: Microservices lab

Demo: Creating Microservices

- The lab13 directory in the Github repository contains a kustomization.yaml file that creates a Microservice based on Wordpress and MariaDB
- This procedure is based on https://kubernetes.io/docs/tutorials/stateful-application/mysql-wordpress-persistent-volume

Demo: Creating Microservices

- vim lesson13lab/wordpress-deployment.yaml
- vim lesson13lab/mysql-deplyment.yaml
- vim lesson13lab/kustomization.yaml
- kubectl apply -k ./
- kubectl get secrets
- kubectl get deployments
- kubectl get pvc (will take few minutes)
- kubectl get pods (wait until all are running)
- kubectl get services wordpress
- minikube service wordpress --url
- From minikube host: http://$(minikube ip):$(serviceport)

Further Learning: Live Courses

- Containers in 4 Hours: Docker and Podman (Jan. 21st)
- Kubernetes in 4 Hours (Jan 7th)
- Getting Started with OpenShift (Jan. 29th)
- Certified Kubernetes Application Developer (CKAD) Crash Course (Jan. 11-13th)
- Certified Kubernetes Administrator (CKA) Crash Course (Jan. 14-15th)
- Container Based Devops in 4 Weeks (Feb 1, 8, 22, March 1)

Further Learning: Recorded Courses

- Hands-on Kubernetes
- Getting Started with Kubernetes LiveLessons 2nd Edition
- Modern Container-Based DevOps: Managing Microservices using Kubernetes and Docker
- Certified Kubernetes Application Developer (CKAD)
- Certified Kubernetes Administrator (CKA)
- Red Hat OpenShift Fundamentals 3/ed