CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Kubernetes

Dr. Philipp Leitner

philipp.leitner@chalmers.se

@xLeitix

LECTURE 4

Covers …

- **Kubernetes**
- **Helm (the Kubernetes Package Manager)**

Note: much of this lecture is based on examples and slides by Lucas Käldström (@kubernetesonarm)

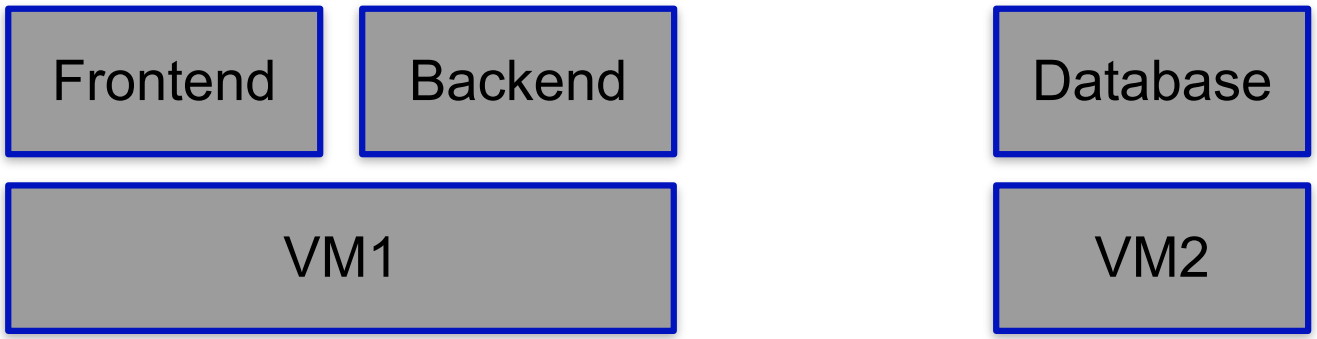
Kubernetes

https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/

Docker-compose allows you to easily start a composition **on a single computer**

… but that's not how we actually want to deploy our application in production

"Kubernetes is a portable, extensible, open-source platform for managing containerized workloads and services, that facilitates both declarative configuration and automation."

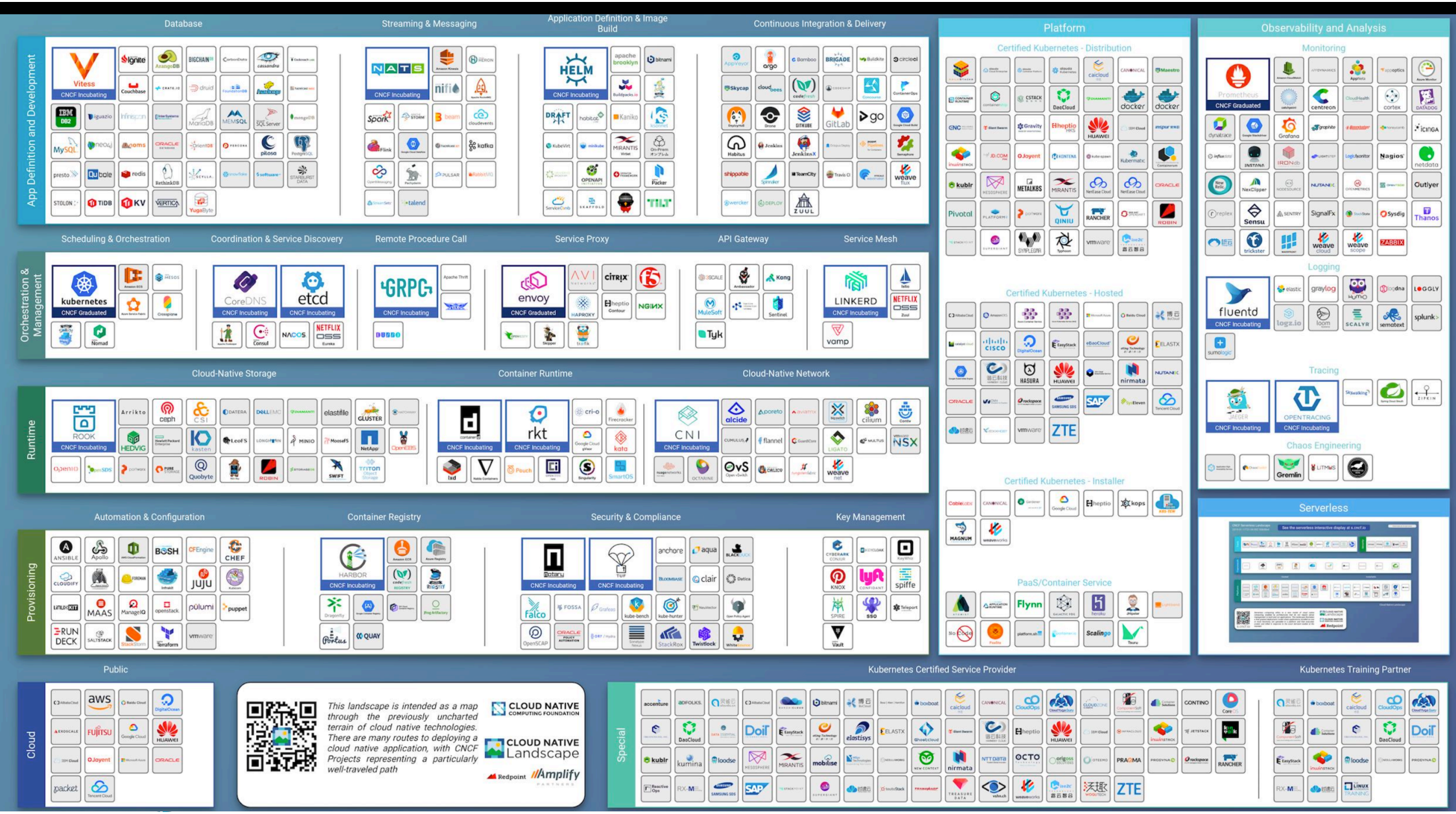


What's Kubernetes?

A Production-Grade Container Orchestration System

Open source project spun out of Google ("Project Borg")
Developed within CNCF, which is itself part of the Linux foundation
Incredibly fancy right now (~ 2 billion containers launched per week)

Extremely large and complex ecosystem


App Definition and Development
Database
Vitess
CNCF Incubating
Streaming & Messaging
NATS
CNCF Incubating
Application Definition & Image Build
HELM
CNCF Incubating
Continuous Integration & Delivery
Orchestration & Management
Scheduling & Orchestration
kubernetes
CNCF Graduated
Coordination & Service Discovery
CoreDNS
CNCF Incubating
etcd
CNCF Incubating
Remote Procedure Call
GRPC
CNCF Incubating
Service Proxy
envoy
CNCF Graduated
API Gateway
Service Mesh
LINKERD
CNCF Incubating
Runtime
Cloud-Native Storage
ROOK
CNCF Incubating
Container Runtime
containerd
CNCF Incubating
rkt
CNCF Incubating
Cloud-Native Network
CNI
CNCF Incubating
Provisioning
Automation & Configuration
Container Registry
HARBOR
CNCF Incubating
Security & Compliance
Notary
CNCF Incubating
TUF
CNCF Incubating
Key Management
Public
Cloud
Special
Platform
Certified Kubernetes - Distribution
Certified Kubernetes - Hosted
Certified Kubernetes - Installer
PaaS/Container Service
Observability and Analysis
Monitoring
Prometheus
CNCF Graduated
Logging
fluentd
CNCF Incubating
Tracing
JAEGER
CNCF Incubating
OPENTRACING
CNCF Incubating
Chaos Engineering
Serverless
This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path
CLOUD NATIVE COMPUTING FOUNDATION
CLOUD NATIVE Landscape
Redpoint
Amplify PARTNERS
Kubernetes Certified Service Provider
Kubernetes Training Partner


1. CONTAINERIZATION
• Commonly done with Docker containers
• Any size application and dependencies (even PDP-11 code running on an emulator) can be containerized
• Over time, you should aspire towards splitting suitable applications and writing future functionality as microservices
2. CI/CD
• Setup Continuous Integration/Continuous Delivery (CI/CD) so that changes to your source code automatically result in a new container being built, tested, and deployed to staging and eventually, perhaps, to production
• Setup automated rollouts, roll backs and testing
3. ORCHESTRATION & APPLICATION DEFINITION
• Kubernetes is the market-leading orchestration solution
• You should select a Certified Kubernetes Distribution, Hosted Platform, or Installer: cncf.io/ck
• Helm Charts help you define, install, and upgrade even the most complex Kubernetes application
kubernetes
CNCF Graduated
HELM
CNCF Incubating
4. OBSERVABILITY & ANALYSIS
• Pick solutions for monitoring, logging and tracing
• Consider CNCF projects Prometheus for monitoring, Fluentd for logging and Jaeger for Tracing
• For tracing, look for an OpenTracing-compatible implementation like Jaeger
Prometheus
fluentd
OPENTRACING
CNCF Incubating
JAEGER
CNCF Incubating
5. SERVICE PROXY, DISCOVERY, & MESH
is useful for service discovery
• Envoy and Linkerd each enable service mesh architectures
• They offer health checking, routing, and load balancing
envoy
CNCF Graduated
CoreDNS
CNCF Incubating
LINKERD
CNCF Incubating
6. NETWORKING
To enable more flexible networking, use a CNI-compliant network project like Calico, Flannel, or Weave Net.
CNI
CNCF Incubating
7. DISTRIBUTED DATABASE & STORAGE
When you need more resiliency and scalability than you can get from a single database, Vitess is a good option for running MySQL at scale through sharding. Rook is a storage orchestrator that integrates a diverse set of storage solutions into Kubernetes.
Vitess
CNCF Incubating
ROOK
CNCF Incubating
8. STREAMING & MESSAGING
When you need higher performance than JSON-REST, consider using gRPC or NATS. gRPC is a universal RPC framework. NATS is a multi-modal messaging system that includes request/reply, pub/sub and load balanced queues.
gRPC
CNCF Incubating
NATS
CNCF Incubating
9. CONTAINER REGISTRY & RUNTIME
Harbor is a registry that stores, signs, and scans content. You can use alternative container runtimes. The most common, all of which are OCI-compliant, are containerd, rkt and CRI-O.
HARBOR
CNCF Incubating
containerd
CNCF Incubating
rkt
CNCF Incubating
10. SOFTWARE DISTRIBUTION
If you need to do secure software distribution, evaluate Notary, an implementation of The Update Framework.
Notary
CNCF Incubating
TUF
CNCF Incubating

What's Kubernetes?

Basic tenet of Kubernetes is that it's declarative and self-healing

Users declare the state of their application in config files

Kubernetes attempts to always keep the application in that state

What's Kubernetes?

Example 1: (availability)

User: "My application consists of 3 MongoDB shards"

k8s: starts three pods with MongoDB running

k8s: observes that one MongoDB shard crashed, starts another

Example 2: (scalability, auto-scaling)

User: "My application runs best with a CPU load [0.4;0.6]"

k8s: starts default number of pods, observes CPU utilization

k8s: observes that CPU utilization >0.6, starts more pods

k8s: observes that CPU utilization <0.4, releases some pods

Core Terminology

https://kubernetes.io/docs/concepts/

Node

A (physical or virtual) resources that Kubernetes should deploy the application *on*
If you use a hosted Kubernetes cluster (like GKE) the resources (nodes) will be VMs provided by the cloud

Cluster

A set of nodes, managed by a Kubernetes installation

Control Plane

The management / scheduling functionality of the cluster, provided by the cluster provider (GCE in our case)

Pod

A group of one or multiple containers that should be deployed together, on the same node

Core Terminology

https://kubernetes.io/docs/concepts/

Service

A group of one or multiple pods that together forms something that can be accessed through the Internet, e.g., a Web application or API

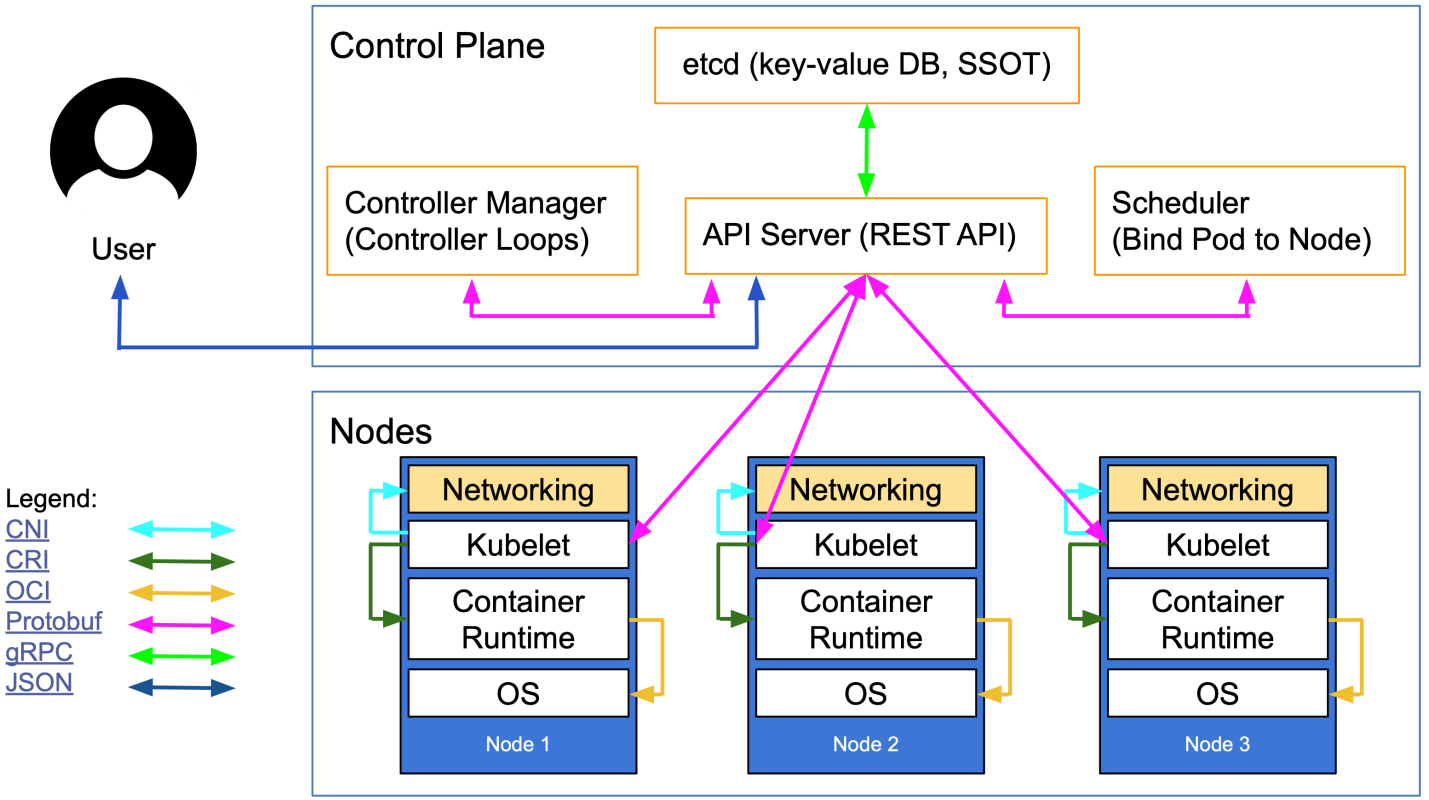
Workload

An application

Namespace

A way to group and isolate different workloads etc. within a cluster

High-Level Architecture



Getting Started with Kubernetes on GCE

Step 1:

Create a cluster

```
gcloud container clusters create \
 --region europe-north1-a --num-nodes 2 dat490
```

Getting Started with Kubernetes on GCE

Step 2:

Run an example job on the cluster

```
kubectl create job hello \
 --image=busybox -- echo "Hello World"
```

Getting Started with Kubernetes on GCE

Step 3: using our cluster

Pods, deployments, services, ingress

Core components of a Kubernetes application
Defined through YAML files (manifest files)

Creating a Pod

Atomic deployable unit

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  containers:
  - image: nginx:1.13.9
    name: nginx
    ports:
    - name: http
      containerPort: 80
```

Creating a Pod

Register manifest with cluster:

```
kubectl apply -f pod.yaml
```

Creating a Deployment

Essentially a collection of pods

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: nginx
  name: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - image: nginx:1.13.9
        name: nginx
        ports:
        - name: http
          containerPort: 80
```

Pod Template

(c) Lucas Käldström

Creating a Deployment

Register manifest with cluster:

```
kubectl apply -f deployment.yaml
```

Creating a Service

Provides a “front-end” for a set of pods
(often a deployment)

Accessible within the cluster through
Internal IP address or internal DNS name:

```
nginx.default.svc.cluster.local
```

(c) Lucas Käldström

```yaml
apiVersion: v1
kind: Service
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
  selector:
    app: nginx
```

Pod Selector

Creating a Deployment

Register manifest with cluster:

```
kubectl apply -f service.yaml
```

Send a test request:

```
kubectl create job testnginx\
 --image=busybox -- wget http://nginx.default.svc.cluster.local
```

(check logs of created pod)

Ingress

Ingress is a separate service that acts as the gateway between the “outer world” and a service.

Separate tool that should either exist externally or be installed in the cluster.

Path of an external request:

Request -> Ingress -> Service -> one pod in the deployment

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: nginx
  namespace: default
  labels:
    app: nginx
spec:
  rules:
  - host: nginx.kubernetesfinland.com
    http:
      paths:
      - path: /
        backend:
          serviceName: nginx
          servicePort: 80
```

Service Reference

Deployment without Explicit Ingress

Configure the type “LoadBalancer” rather than “ClusterIP” in `service.yaml`

```
service:
  type: LoadBalancer
  port: 80
  targetPort: 80
```

Helm

So far we have seen how to configure k8s "manually"
(through a large number of increasingly complex YAML files)

Helm (https://helm.sh) is the standard **package manager** for Kubernetes

Two core use cases:

(1) Easily install existing workloads (applications) in our cluster

(2) Package our own applications using Helm and deploy all in one go (rather than registering manifest by manifest)

Using Helm to Install an Existing Application

Let's install MongoDB as an example workload via Helm

(also part of Assignment 2)

Add package repository:

```
helm repo add bitnami\
 https://charts.bitnami.com/bitnami
```

Install package:

```
helm install mongo bitnami/mongodb
```

values.yaml

Typically, Helm packages (“Helm charts”) can be configured / customised through a values.yaml file

```
helm install -f values.yaml\
 gitlab-runner gitlab/gitlab-runner
```

values.yaml

Configurations can also be applied retroactively

```
helm upgrade -f values.yaml\
 gitlab-runner gitlab/gitlab-runner
```

Packaging our own application as a Helm Chart

Helm charts are written as templates using the Go templating system tpl

Templates for the different k8s manifest files that are needed to deploy the app

Creating a new chart:

```
helm create my-application
```

```
phileitn@dev-machine:~$ ls my-application/
Chart.yaml  charts  templates  values.yaml
```

Structure of a Helm Package

Chart.yaml … basic information about the package

values.yaml … default values, can be overwritten during install

templates (folder) … templates for all types of manifests that a k8s application tends to need (deployment, service, ingress, …)

templates/NOTES.txt … what gets outputted after the install

Packaging and installing

Lint (check) our chart

```
helm lint my-application
```

Create the package

```
helm package my-application
```

Install on cluster

```
helm install my-application my-application-0.1.0.tgz
```