Implementation of CMSWEB Services Deployment Procedures using HELM

Aroosha Pervaiz, **Muhammad Imran**[1], Valentin Kuznetsov[2], Panos Paparrigopoulos[1] Spyridon Trigazis[1], Andreas Pfeiffer[1] ,
[1]CERN, Geneva, Switzerland
[2]Cornell University, USA
Email: muhammad.imran@cern.ch

Agenda

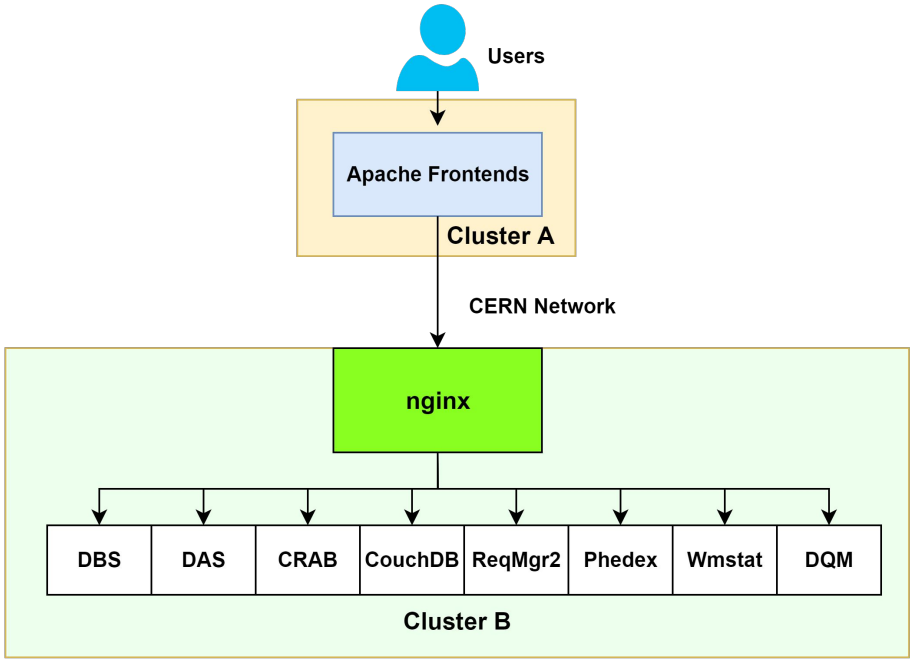
- Introduction
- Architecture of CMSWEB
- Motivation
- Kubernetes Vs Helm
- So what Helm is, at heart?
  - Basic Structure of Helm Chart
  - Helm Working Scenario
  - Helm Versioning System
  - Usage of HELM Charts in Different Environments
- Canary Releases
- Conclusion

Introduction

- The CMS experiment runs hundreds of thousands of jobs daily on its distributed computing system to simulate, reconstruct and analyse the data taken during collision runs.
- A dedicated cluster ("CMSWEB") is used to hosts essential CMS central services which are responsible for the CMS data management, data discovery, and various data bookkeeping tasks.
- The CMSWEB cluster was migrated to Kubernetes infrastructure from virtual machines (VMs).
    - Migration has resulted in high availability of services, reduced operational cost, end-to-end service deployment by developers and efficient utilization of resources.

Architecture of CMSWEB

- It has the two components:
  - **frontend cluster**
  - **backend cluster**
- The Apache frontend performs x509 certificates authentication and redirects the request to the backend cluster.
- On the backend cluster, the ingress controller has basic redirect rules to the appropriate services and only allows requests from the frontend cluster.



Motivation

- Although, we have done migration to Kubernetes but we realized that Kubernetes is complex because of all different sorts of objects — such as ConfigMaps, services, pods, persistent volumes — in addition to the number of releases we need to manage.
- Instead, we now want to be able to:
  - Package all of the related objects into one simple application.
  - Install them easily
  - Focus more on application development
  - Reduce the complexity of maintaining an app catalog in a Kubernetes environment.
  - Easily rollback and upgrade our deployments.
  - Run the multiple versions of the services without changing the service manifest files and Nginx ingress rules.
  - Easily share configuration files
- This is where helm comes to help!

Kubernetes Vs HELM

kubectl vs helm

- Using kubectl:
  - Create, and manage deployment
  - Not easy to rollback, upgrade, and configure
  - We cannot create multiple deployments without changing the metadata manually
- And what about sharing the deployment configurations with colleagues?
  - We need to share multiple files
  - They need to have knowledge about the deployments' configurations

- No expertise of Kubernetes needed
- Helm packages all dependencies
- Desired configuration can be passed at runtime
- Easy to update and rollback
- Same workload can be deployed multiple times
  - Helm uses templates functionality so, we can also deploy different application configurations at runtime.
- Easy to share as only one packaged chart

So what Helm is, at heart?

- It can be thought of as a package manager for Kubernetes, like yum, apt, homebrew.
- It streamlines and simplifies the installation, and then subsequently, the management of Kubernetes applications
- It's package format is called chart.
- A chart is a collection of files that describe Kubernetes resources
- It can be run anywhere i.e., Laptop, CI/CD pipeline, e.t.c.
- We have also implemented CI/CD workflow that directly uploads the packaged chart to registry.cern.ch and github repos upon adding new versions to the github directory.

Basic Structure of Helm Chart

Required files:

Chart.yaml - A YAML file containing information about the chart.

A **templates** directory:

- **templates/** - A directory of templates with Kubernetes manifest files or that will generate valid Kubernetes manifest files when combine with values.yaml.

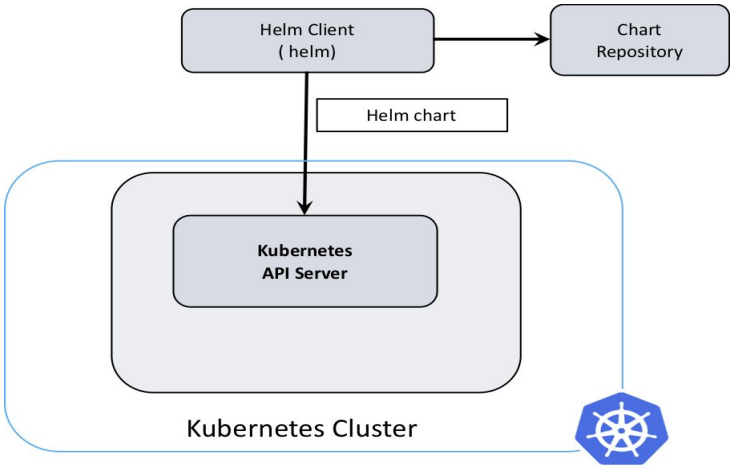
Optional files:

LICENSE - A plain text file containing the license for the chart
README.md - A human-readable README file
values.yaml - The default configuration values for this chart
templates/NOTES.txt - A plain text file containing short usage notes
template/_helpers.tpl – template helpers that you can re-use throughout the chart

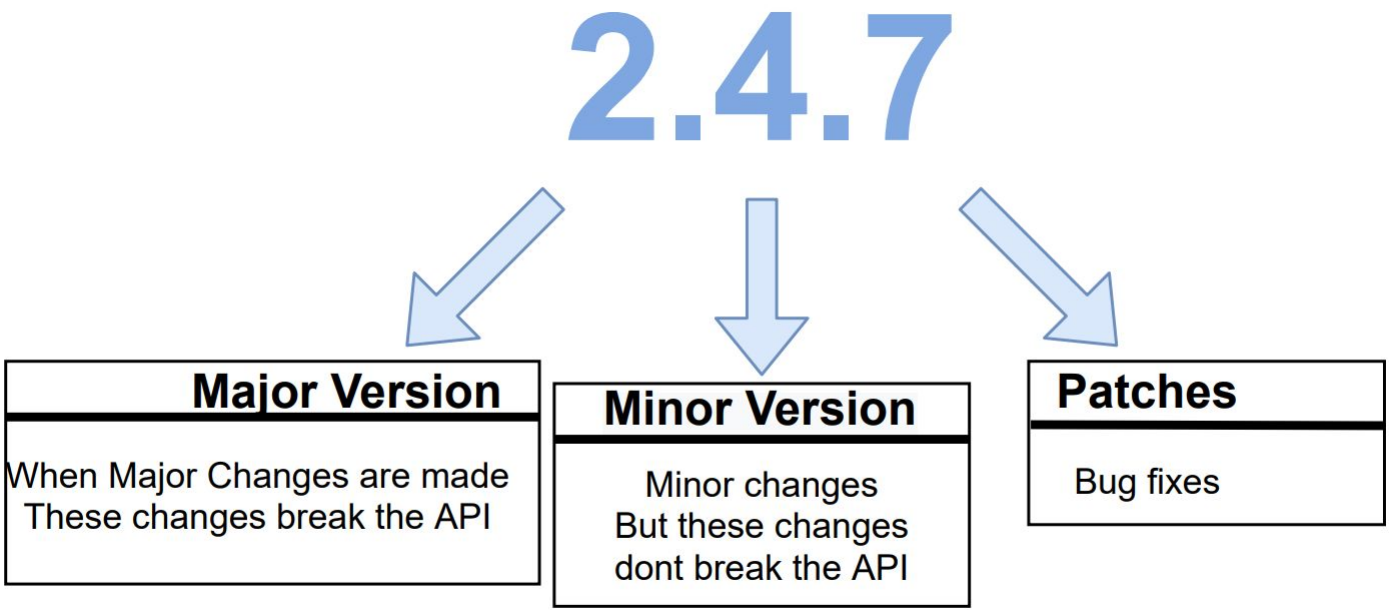
```
[apervaiz@lxplus802 ~]$ tree dbs/dbs
dbs/dbs
├── Chart.yaml
├── templates
│   ├── deployment.yaml
│   ├── _helpers.tpl
│   ├── hpa.yaml
│   ├── ingress.yaml
│   ├── NOTES.txt
│   ├── serviceaccount.yaml
│   ├── service.yaml
│   └── tests
│       └── test-connection.yaml
└── values.yaml

2 directories, 10 files
```

Helm Working Scenario

- Helm Client:
  - It is a command line client for the end users of Helm.
  - It is responsible for development of Local Charts, Managing Charts, Interfacing with the Helm Library, and Managing Releases.
  - It interacts with the Kubernetes API server to install the helm chart to the Kubernetes Cluster.
- Helm Library:
  - It provides logic for executing Helm Operations. It is a standalone library which has all the logic needed for Helm operations to execute



Helm Versioning System



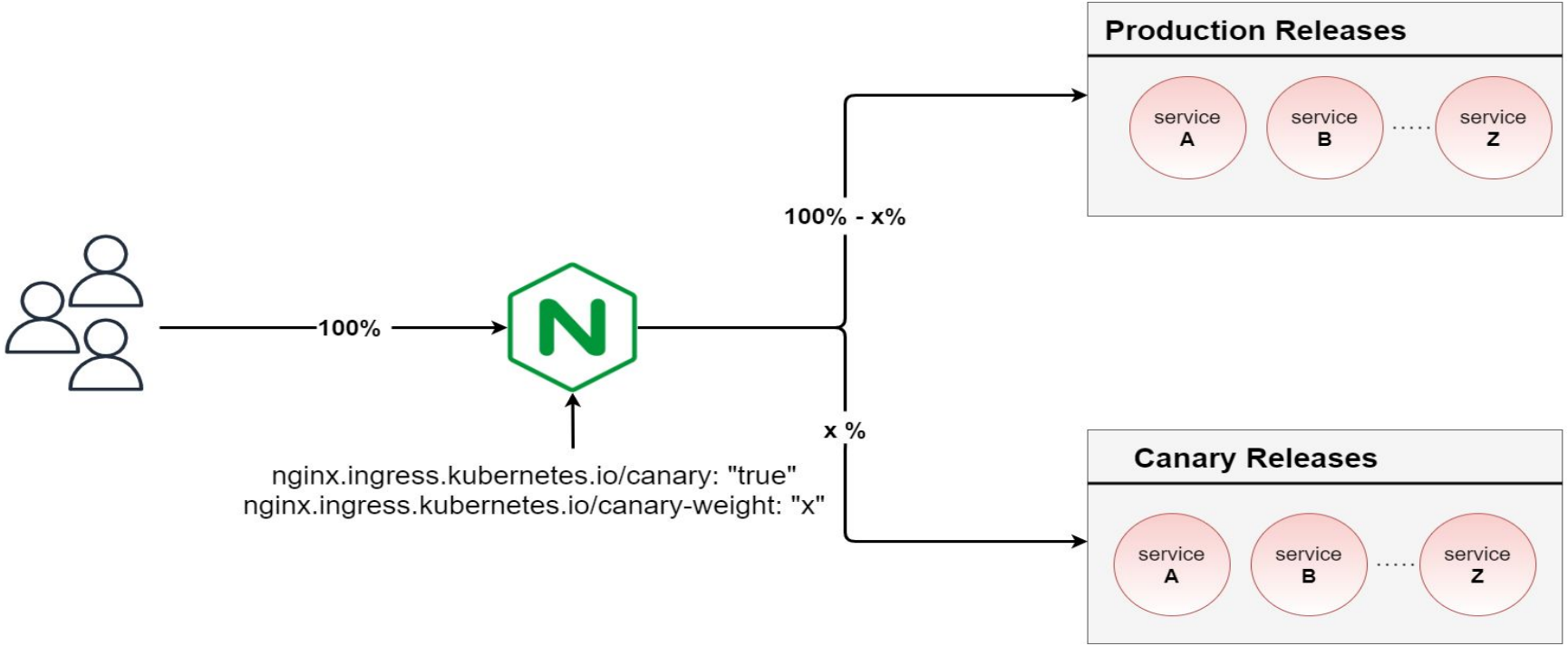
Usage of HELM Charts in Different Environments

- Three environments available for CMSWEB services:
  - production, pre-production and development environments.
- The Kubernetes manifest files needed to be adjusted according to environments
  - We implemented bash scripts to do necessary adjustments according to environments
  - However, need adjustments for all new services and functionalities.
- We devised a way through Helm, to do this all with just one variable.
  - The users need to set the environment during the installation of the Helm chart through CLI, and this environment can be upgraded, rolled back and reset using CLI.
  - No need to change manifest files

Canary Releases

- We cannot risk the unavailability of our Kubernetes cluster, or any particular service in production environment.
- In order to make our system more scalable, we used and implemented the concept of “canary release” provided by Nginx ingress.
- Using this technique, we can redirect some percentage of our traffic to the new deployment.
- Upgrade releases without having to modify or delete the existing release.
- In case of any bug, it’ll only affect a small percentage of the users before being detected and rolled back.

Canary Releases



Conclusion

- We simplified the deployment procedures of CMSWEB services using Helm's charts and incorporated canary releases using Nginx ingress weighted routing that is used to route traffic to multiple versions of the services simultaneously.
  - This feature is very useful for developers to test the new versions of the services by assigning some weight to the new service version and rolled back immediately in case of issues.
- The usage of Helm simplifies the deployment procedure and no expertise of Kubernetes are needed anymore for service deployment.
- Helm packages all dependencies and services are easily deployed, updated and rollbacked.
- Using Helm, we can also deploy different application configurations at runtime