

New Features in CMSWEB Kubernetes Cluster at CERN

Project Report
CERN, Geneva

Submitted by
Aroosha Pervaiz
CERN Summer Student, 2021

Supervised by
Muhammad Imran

Co-Supervised by
Valentin Kuznetsov
Panos Paparrigopoulos
Spyridon Trigazis
Andreas Pfeiffer

Abstract

The Compact Muon Solenoid (CMS) experiment heavily relies on the CMSWEB cluster to host critical services for its operational needs. Recently, there has been migration of the CMSWEB cluster from the VM cluster to the Kubernetes (k8s) cluster. The new cluster of CMSWEB in Kubernetes enhances sustainability and reduces the operational cost. In this project, we added new features to the CMSWEB k8s cluster. The new features include the deployment of services using Helm's chart templates and the incorporation of canary releases using Nginx ingress weighted routing that is used to route traffic to multiple versions of the services simultaneously. 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 rolledback. Helm enables us to deploy multiple versions of the services to run 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 rolledback immediately in case of issues. Using Helm, we can also deploy different application configurations at runtime.

Table of contents

I	Introduction	4
II	HELM Charts	6
II.1	Repository	6
II.2	Contact Command and Control servers	6
II.3	Helm Working Scenario	7
II.4	Updating and Maintaining the Helm Charts	8
II.5	Helm Versioning System	8
II.6	Usage of Helm charts in Different Environments	8
III	Canary Releases Using Nginx Ingress	10
IV	Documentation and Support	11
V	Conclusion	12
	Acknowledgements	13
	Bibliography	14

I. Introduction

CMSWEB Kubernetes cluster was instantiated when the migration of CMS cluster from VM clusters to Kubernetes cluster was performed. This CMSWEB cluster manages dozens of services in a consistent fashion. The framework for the deployment is written in bash scripts which enables the deployment and management of these services. To put it into perspective, the CMSWEB enables the independent deployment, management and evolution of services, the simplification of integration and regression testing when these services are updated, and also the development and association of external services that allow for the information to be integrated in a clean, organized way.

Each service in CMSWEB has its own development group that upgrades the relevant service in the monthly release cycle. The CMSWEB development life cycle includes all the hosted services that are actively developed. The CMSWEB team is not responsible for maintaining individual services but only provides the web infrastructure and manages service deployment.

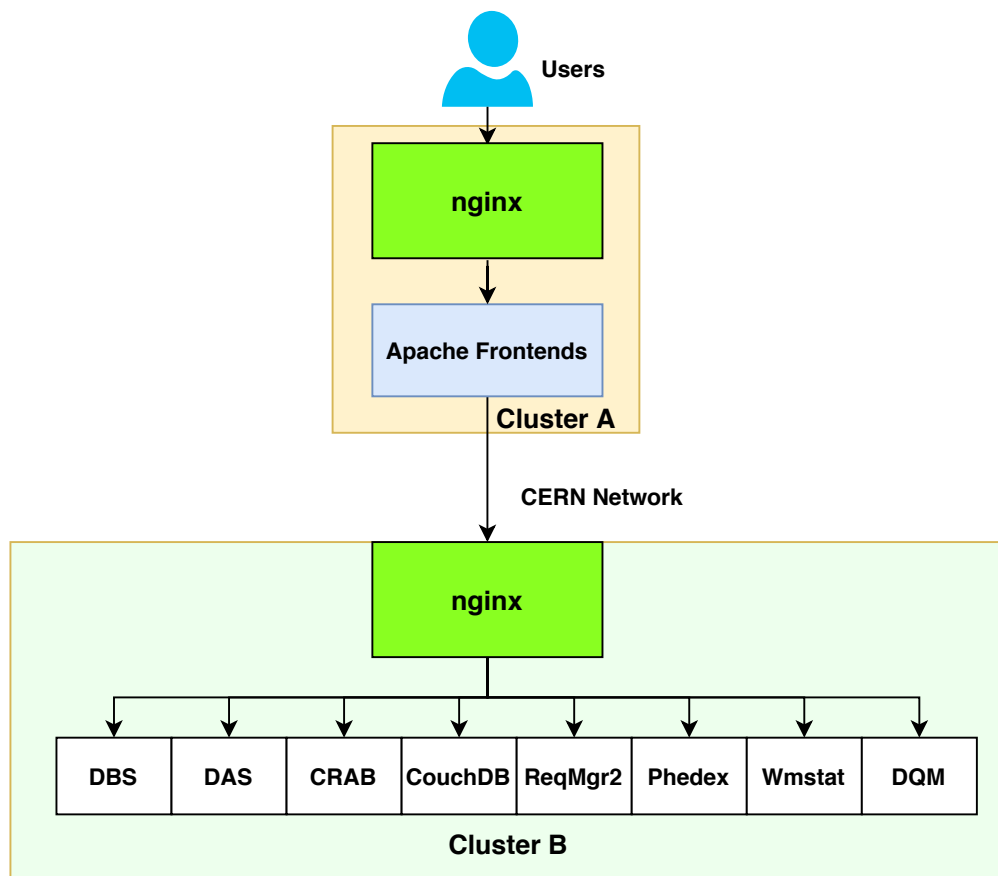


Figure I.1: The CMSWEB Kubernetes cluster architecture[1]

When it comes to the architecture of the CMSWEB VM cluster, it has two

layers of services, i.e., frontend and backend services as depicted in Figure I.1. The frontend service relies on Apache which first does the task of authenticating the request using X509 certificates and then the task of redirecting the request to the backend services. It is actually independent and individual backend services that carry out the requested and relevant tasks. There are dozens of namespaces of this Kubernetes VM, mainly, crab, dbs, reqmgrms, reqmgr2, reqmon, das. The services in these namespaces include crabserver, crabcache, das-server, das-mongo, das-mongo-exporter, dbs-global-r, dbs-global-w, dbs-migrate, dbsmigration, dbs-phys03-r, dbs-phys03-w, ms-monitor, ms-output, ms-output-mongo, ms-rulecleaner, ms-transferor, reqmgr2, reqmgr2-tasks, reqmon, reqmon-tasks, workqueue, t0reqmon, t0reqmon-tasks, t0wmadatasvc, httpgo, exitcodes, and warchive. More detail on these services can be found at [1].

Each of the CMSWEB services are currently deployed through manifest files in Yaml using Kubernetes's `kubectl` tool. However, it becomes increasingly tedious to manage all those manifest files because of all the different sorts of objects that Kubernetes has to offer — such as configMaps, services, pods, persistent volumes — in addition to the number of releases you need to manage. Furthermore, it is not possible to run the multiple versions of the services without changing the service manifest files and Nginx ingress rules. It is not easy to rollback, upgrade, and configure services using `kubectl` tool. We also need to share multiple files for deployment configurations with service developers and the developers also need to have knowledge about the deployments' configurations and Kubernetes. This is where Helm charts come to help us. 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 rolled back. Helm enables us to deploy multiple versions of the services to run 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. Using Helm, we can also deploy different application configurations at run-time.

In this report, Section II gives an overview of the Helm charts and describes how they are used. Section III then discusses the canary releases using Nginx ingress that is additionally included in those Helm charts. Section IV provides brief detail about documentation and support. Finally, we conclude in Section V.

II. HELM Charts

Helm [2] is a combined effort of Deis (which is now a part of Microsoft) and Google. It became a part of the Kubernetes 1.4 release in 2016 and has been easing the life of developers ever since. In April 2020, Helm was classified as a CNCF project in April 2020 which further added to its popularity. It is one of the open-source projects that wish to address the complexities of Kubernetes. Helm, a packet manager for Kubernetes, makes it easier to deploy, upgrade and roll back the Kubernetes deployments by simplifying the processes to a single CLI command.

The package format of Helm is called chart, which is a collection of files describing Kubernetes resources. Helm charts enable the definition and installation of even the most complex Kubernetes applications. These Helm charts can be customized at runtime that also automates the upgrading and rolling back of releases. Helm also maintains the history of previous versions of the releases, so if things go wrong, we can roll back. Helm also allows us to package all the related Kubernetes resources. Hence, it becomes effortless to package, distribute, and download and install these Helm charts.

Helm also allows for the same chart to be deployed multiple times across multiple environments with different values, unlike Kubernetes. Pertaining to all the benefits that Helm offers, we converted our Kubernetes manifest files of CMSWEB services into Helm charts and stored them in a separate Helm repository. We believe that it will reduce the complexity of modification and upgrading of services. We will also have the entire history of our previous versions of the releases which will make it easier to track back. By creating a repository for the Helm charts of the services, the users will be easily able to install all the services locally. They can also easily update the Helm repository and leverage it to install their desired services.

II.1 Repository

Helm charts can be hosted at an online repository. In order to host our Helm charts, and make them available to the CMSWEB team, they are being served at two locations: [3] and [4]. One can fetch these repositories, push Helm charts to these repositories, and update their locally cached repositories to match the changes in the aforementioned repositories.

II.2 Contact Command and Control servers

Helm uses a packaging format known as chart. A chart is a collection of interrelated files inside a directory. The name of the directory reflects the name of the chart. A typical chart contains a *Chart.yaml* file and a *templates* directory. This *template* directory then contains the interrelated deployments for the same chart. The *Chart.yaml* and *templates* directory is a required prerequisite for a directory to be deemed as Helm chart. It is shown in Figure II.1.

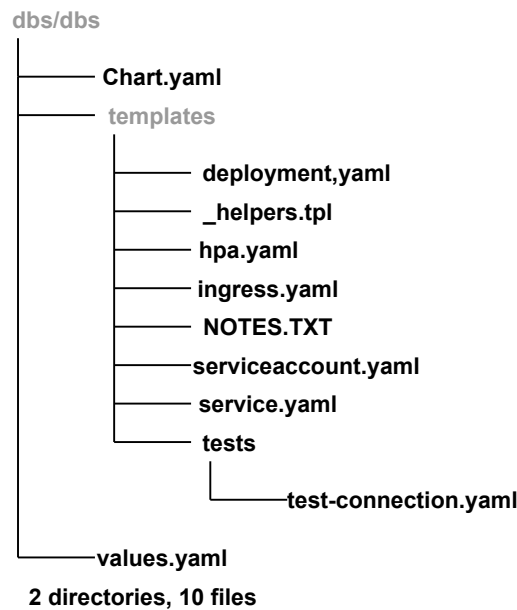


Figure II.1: A typical Helm chart directory.

The brief description of each file is shown below:

- 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 combined with values.yaml.
- 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
- We can add more manifest files in the templates directory corresponding to our needs for more Kubernetes objects.

II.3 Helm Working Scenario

Helm is an executable that is implemented in two components:

- 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 client connects and installs the helm chart to the Kubernetes Cluster by interacting with Kubernetes API Server as shown in Figure [II.2](#).

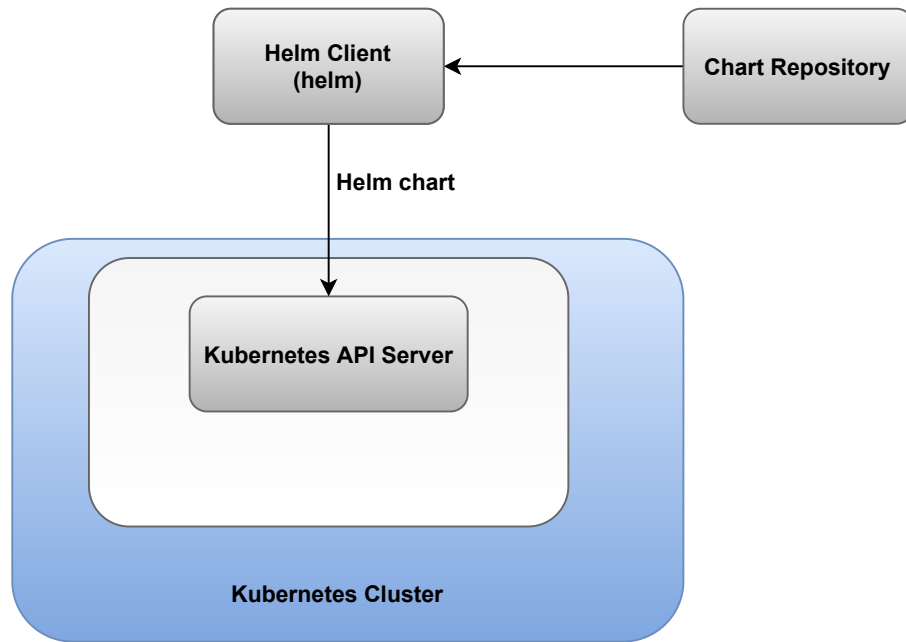


Figure II.2: Helm client connects and installs the helm chart to the Kubernetes Cluster by interacting with Kubernetes API Server.

II.4 Updating and Maintaining the Helm Charts

The Helm charts are updated, maintained and updated through CI/CD pipeline. If a Helm chart has been created or updated the chart and needs to be uploaded to our GitHub repository, you just need to upload the updated code to the *helm* directory in the repository. It will automatically add the release at [3].

II.5 Helm Versioning System

Helm uses semantic versioning to name the charts through which these charts are identified. As depicted in Figure II.3, the semantic versioning has three components, MAJOR version, MINOR version, PATCH version. It is in the format MAJOR.MINOR.PATCH.

Given this format, we increment:

1. MAJOR version is changed when we make changes to API that are incompatible to the previous versions.
2. MINOR version is changed when such functionality is added that is backwards compatibility.
3. PATCH version is changed when we fix backwards compatible bug fixes.

II.6 Usage of Helm charts in Different Environments

Currently, there are three environments available for CMSWEB services, namely, production, pre-production and development environments. [1] The Kubernetes

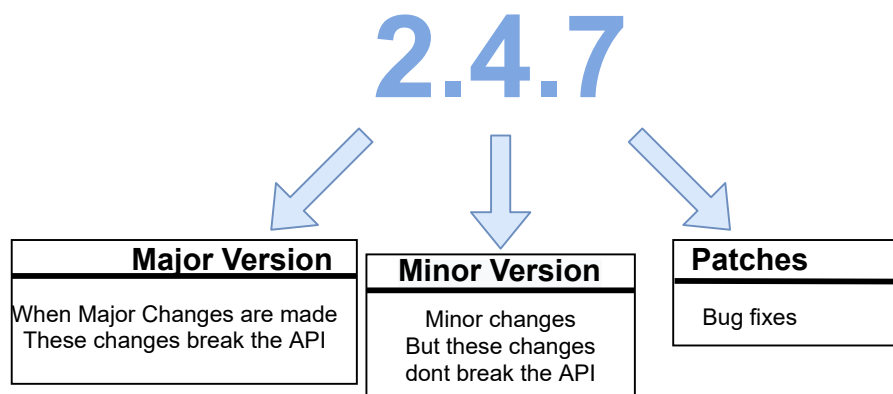


Figure II.3: Helm Versioning System uses Semantic Versioning to name its charts. It comprises of three parts: major, minor, and patch version.

manifest files needed to be commented at some parts, and uncommented on others, to be deemed suitable to work in these different environments. So, 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. More detail on this implementation can be found at [\[5\]](#) and [\[6\]](#).

III. Canary Releases Using Nginx Ingress

Our production cluster hosts all the major services required by the CMS for operational needs. Therefore, we cannot risk the unavailability of our Kubernetes cluster, or any particular service. 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, upgraded releases without having to modify or delete the existing release. This mechanism is shown in Figure III.1

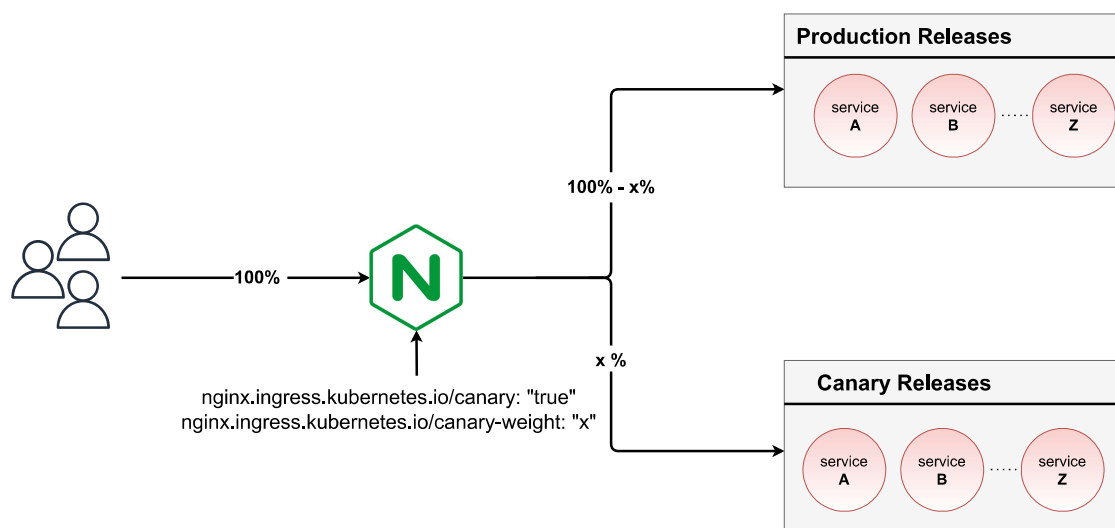


Figure III.1: A depiction of how ingress divides the traffic between canary and production releases. As it can be seen, we assign a value to X variable; X% of the traffic goes to the canary releases, while the rest of the traffic stays assigned to the production releases.

Canary releases are a way to release updates to a shifting percentage of the total user base. If there is a need to deploy a new code, or update the image, this newer release can be installed as a canary release. In case of any bug, it'll only affect a small percentage of the users before we can detect and rollback.

IV. Documentation and Support

Documentation plays an important part in familiarizing others of the work you have done and increases the usability of one's code by fourfold. For this reason, I have documented the entire procedure along with details of how my work can be used. This can be found at [\[5\]](#). If there is a need to familiarize oneself further with how Helm works, the official Helm documentation can be found at [\[7\]](#).

V. Conclusion

In this project, we added new features to the CMSWEB kubernetes cluster. The new features include the deployment of services using Helm's chart templates and the incorporation of canary releases using Nginx ingress weighted routing that is used to route traffic to multiple versions of the services simultaneously.

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 rolledback. Helm enables us to deploy multiple versions of the services to run 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 rolledback immediately in case of issues. Using Helm, we can also deploy different application configurations at runtime.

The code for all these Helm charts and the Nginx functionality can be found at the repository [8] and its related documentation can be obtained through our official documentation at [5]

Acknowledgments

I am very thankful to CERN for providing me with this invaluable opportunity to not only gain technical knowledge, but also gain insight of how things work at a professional level. I am thankful to all my supervisors, namely, Muhammad Imran, Valentin, Panos, Spyros and Andreas. However, I am especially indebted to Muhammad Imran for always guiding me through everything and teaching me invaluable skills; it wouldn't have been possible without his help and support.

Bibliography

- [1] [Migration of CMSWEB cluster at CERN to Kubernetes: a comprehensive study.](#) 24, 3085–3099 (2021).
- [2] *Helm*. <https://helm.sh/>.
- [3] *dmwm/CMSKubernetes*. <https://github.com/dmwm/CMSKubernetes/releases>.
- [4] *CERN Registry*. <https://registry.cern.ch/chartrepo/CMSWEB/index.yaml>.
- [5] *CMSWeb Documentation*. <https://cms-http-group.docs.cern.ch/Helm/deploy-services/>.
- [6] *Training Session (CMSWEB New Features)*. <https://indico.cern.ch/event/1075972/>.
- [7] *Docs Home. Helm.sh*. <https://Helm.sh/docs/>.
- [8] *GitHub - dmwm/CMSKubernetes: Set of instructions and examples to deploy CMS data-services to Kubernetes cluster. GitHub.* (2021). <https://github.com/dmwm/CMSKubernetes>.