Overview of CMSWEB Cluster in Kubernetes

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

Agenda

- Introduction
- Motivation
- Current VM Cluster Architecture
- Proposed Architecture of CMSWEB in Kubernetes
  - Services Deployment Procedures
  - Services Deployment Cycle
  - HPA
  - Monitoring for CMSWEB Kubernetes
  - Migration Status
  - Issues Faced and Fixed During Migration
- Lessons Learned
- Future Work
- 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 is based upon virtual machines (VMs) on the CERN OpenStack cloud infrastructure.
- Each service is managed by its own development team.
  - Due to the complexity of the heterogeneous environment, different schedules of development teams, etc, only monthly release cycles can be afforded.

Introduction

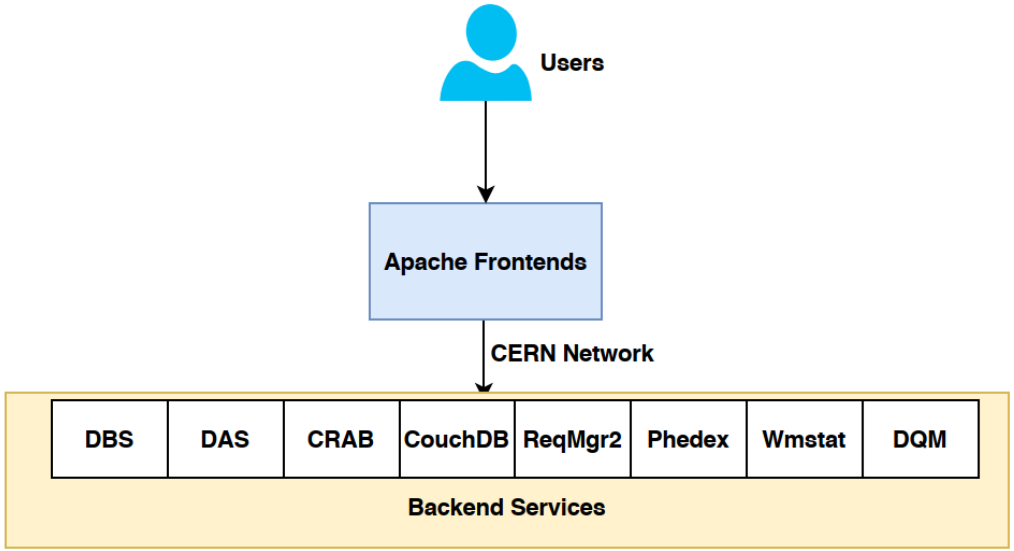
- Each upgrade cycle includes:
  - for each service the build of RPMs from source code
  - the cross-validation of all software components
  - and the validation of the correct interactions of all services
- This requires:
  - a lot of interactions between development teams and operator
  - Manual interventions by operator to manage clusters.
- By policy, production releases are only deployed once a month,
  - so developers may have to wait up to **4 weeks** before the new version of their services is deployed into production.
  - Because production upgrade incurs downtime of some services (3-4 hours)

Motivation

- To enhance the sustainability of CMSWEB, CMS decided to migrate to a containerized solution:
  - based on Docker
  - and orchestrated with Kubernetes (“k8s”)
- With the containerized approach, developers will not have to ask the operators to deploy their services,
  - they can deploy new versions of their services in a few seconds.
- This significantly reduces:
  - the release upgrade cycle
  - the efforts on end-to-end deployment procedures
  - operational cost

VM Cluster Architecture

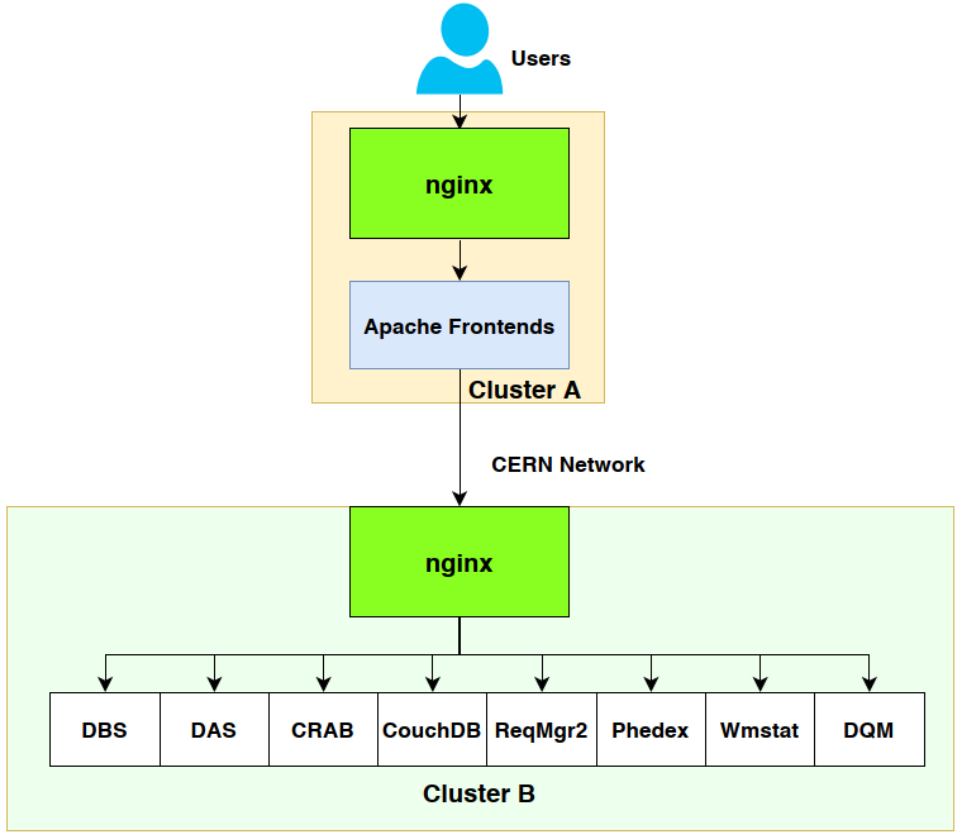
- It has two layers of services i.e., frontend and backend services.
- The frontend service is based on Apache which performs authentication using certificates and redirects requests to the requested backend service



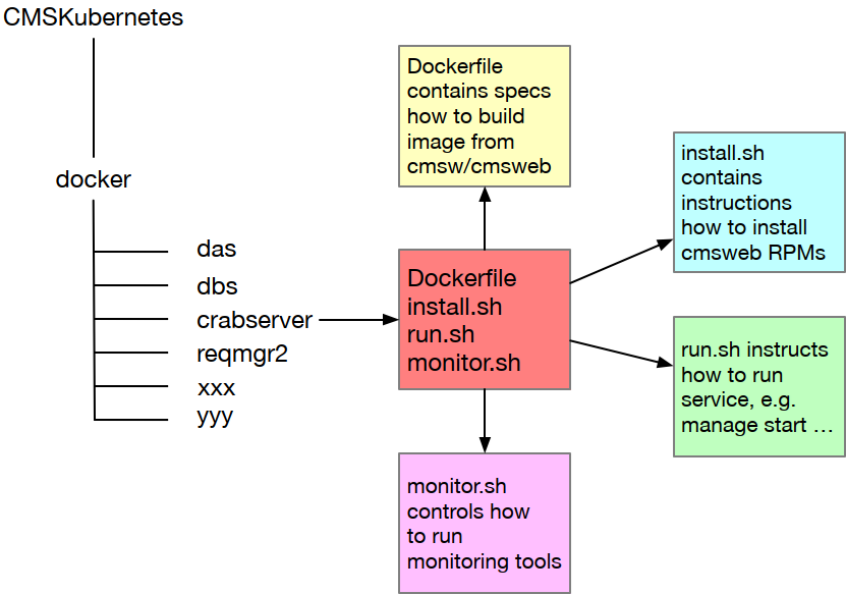
- The backend services perform their relevant tasks.
- The frontend service has redirect rules to forward the requests to the relevant VM node running the backend service.
- Backend services only allow requests coming from the frontend service.

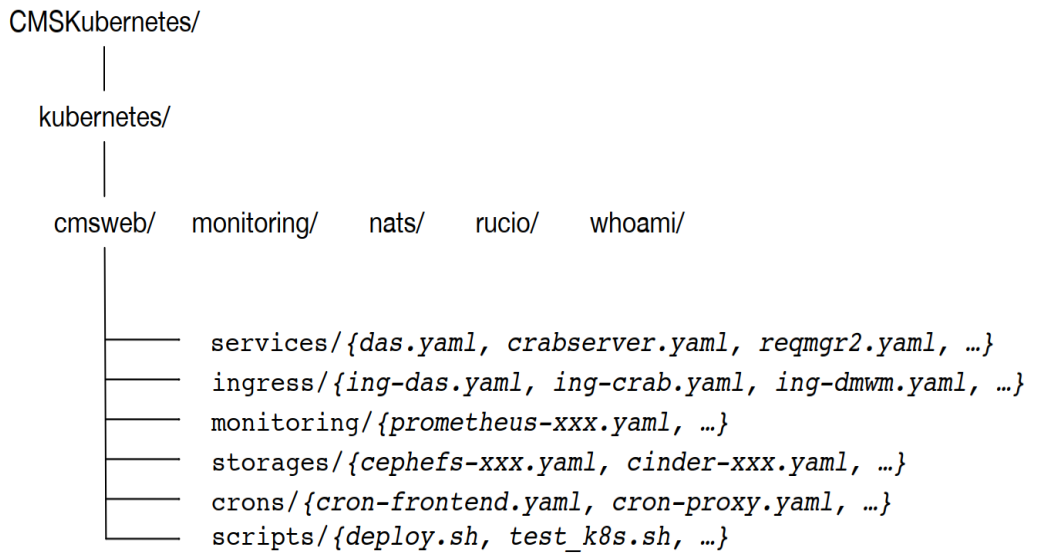
Architecture of CMSWEB in Kubernetes

- It has the two components:
  - **frontend cluster**
  - **backend cluster**
- The frontend cluster ingress controller provides TLS passthrough capabilities to pass client's requests (with certificates) to the Apache frontend.
- The Apache frontend performs CMSWEB 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.



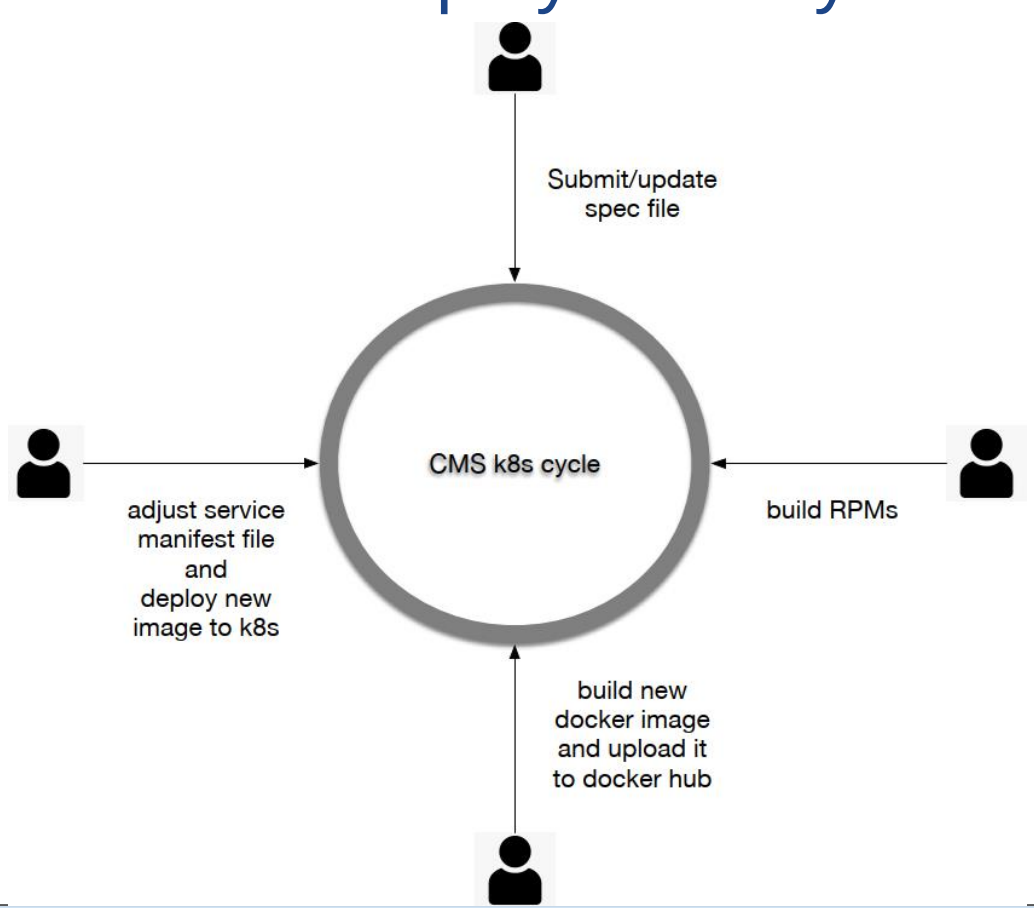
Services Deployment Procedures





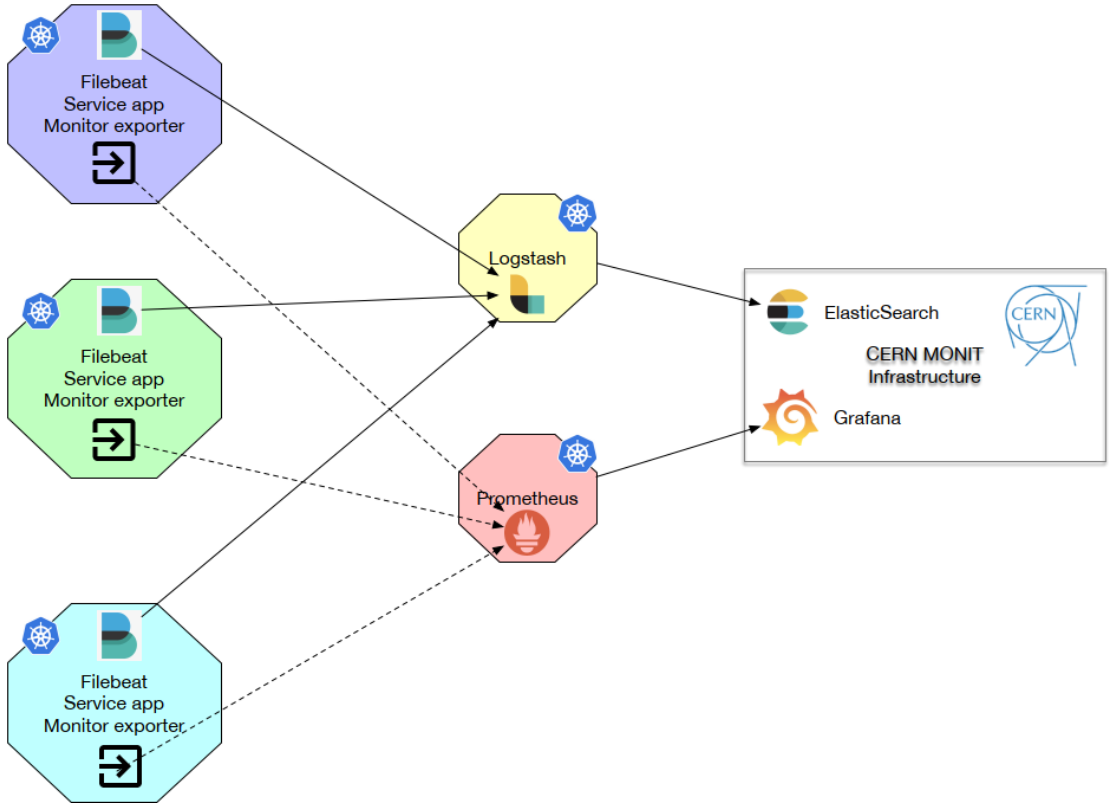
- A central repository with docker and cmsweb sub-directories.
- docker area is for creation of images
- cmsweb area is for deployment of images in Kubernetes

Services Deployment Cycle



CMSWEB Kubernetes Monitoring Architecture

- Cluster monitoring is performed via Prometheus and Logstash.
  - Prometheus provides monitoring
  - Logstash is used to collect logs via Filebeat daemon.



Logs

- We use Cephfs to mount service logs from containers on a dedicated VM.
- Users can access logs in real time and can investigate using tools such as grep, awk etc.
- Currently exploring following scenarios:
  - S3
  - EOS
  - Copying logs directly to dedicated VMs

Migration Status

- Services Migrated
  - 20
- Services to be migrated soon
  - DBS, DMWM microservices
- Services will be running in VM
  - Couchdb, DQM, Confdb, Phedex
- More detail
  - https://gitlab.cern.ch/cms-http-group/doc/-/issues/236

Horizontal Pod Autoscaling (HPA)

- We applied HPA to various services based on hybrid metrics such as CPU and Memory.
- Recently, we applied HPA based on custom metrics as well on DBS.
  - CPU, Memory and open file descriptors
  - More detail is available on separate talk by Tommaso

Issues Faced and Fixed During Migration

- Various issues were found, which are divided into two categories:
  - Infrastructure Issues
  - Service Issues
- Infrastructure Issues
  - Network Degradation
  - Ceph Issues
  - Permission Mount Issues
  - Nginx-ingress controller issues
  - Load balancing
  - DNS caching
- Service Issues
  - Generic database like CouchDB
  - Data placement system (PhEDEx)

- All these issues have been fixed
- Thanks to support from CERN IT

Lessons Learned

- The existing VM cluster requires:
  - a lot interactions between the operator and developers
  - manual interventions is needed from the operator to deploy and maintain the clusters.
- The new Kubernetes infrastructure:
  - greatly reduces the efforts and workload on the CMSWEB operator.
  - automates the procedure of service deployment
  - enables developers to deploy their services directly in the K8s cluster without needing input from the operator.
- Developers will not have to wait for a month before their services are put into the production.
- No intervention for services

Lessons Learned

- The auto-scaling feature of Kubernetes scales up cluster resources as soon as they are required, and scales them back down once they are not any more needed.
  - The current VM cluster lacks this feature.
  - Every service is deployed on particular VM nodes and the resources are assigned on the VM level instead of the service level.
  - During high load:
    - services might become less responsive
    - the CMSWEB operator manually interferes and resolves issues with individual services.
- Manifest files in Kubernetes requires verification and proper indentation

Future Work

- Service-mesh deployment:
  - the service-mesh provides plenty of benefits to Kubernetes
    - including traffic encryption within the cluster,
    - traffic routing between different releases,
    - canary deployment and rolling release cycles.
  - Using Istio or the Gloo middlewares.
- Deployment using Helm
- Image size of Python based apps is very large ~3GB.
  - Motivating developers to migrate their services to Go based apps.

Conclusion

- The new cluster of CMSWEB in Kubernetes enhances sustainability and reduces the operational cost of CMSWEB.
- With the containerized approach, developers will not have to wait for the operators to deploy their services, they can deploy new versions of their services in a few seconds without intervention.
- This allows CMS to significantly reduces:
  - the release upgrade cycle,
  - the efforts on end-to-end deployment procedures,
  - and reduce operational cost.