Towards a container-based architecture for CMS data acquisition

**Presenter:** Dainius Šimelevičius on behalf of CMS DAQ group  
**Primary authors:** Luciano Orsini - CERN, Dainius Šimelevičius - CERN/Vilnius University
Outlook

- Introduction
- Motivation
- Pilot project
- Conclusion
Introduction
CMS Data Acquisition (DAQ) for LHC run 3

Data flow, aggregation, control and monitoring software

High Level Trigger software
DAQ as a service-oriented architecture

- DAQ is implemented as a service-oriented architecture where DAQ applications, as well as general applications such as monitoring and error reporting, are run as self-contained services.

- The task of deployment and operation of services is achieved by using several heterogeneous facilities, custom configuration data and scripts in several languages.

- Deployment of all software is carried out by installation of rpms through Puppet management system on physical and virtual machines in computer network.

- Two main approaches are used to operate and control the life cycle of the different services: short-lived services, such as event building and read-out, are managed using a custom-built infrastructure, while auxiliary, long-running services are managed using systemd.
Motivation
Motivation

- The current system works well, nevertheless we identified points of improvement for the future system.
- User experience and lessons learned provided a groundwork for reassessment of the software.
- There are opportunities coming from new technologies.
- Coping with a changing environment.
Points to improve (1)

- Infrastructure **lock-in** caused by dependency on a given operating system and libraries. Inability to use different alternatives without substantial switching costs.

- **Time consuming** transition from development to deployment.

- Significantly **different** development, validation and production environments.
Points to improve (2)

- Non-optimal usage of hardware resources due to application binding to physical or virtual machines
- Cost-intensive building of development and validation environments
- Update and rollback of software is non-trivial
- The need for system administration effort during deployment and operation
- Lack of portability regarding physical host, network or system reconfiguration
- Need for custom (private) scripts for software deployment and operation of distributed applications in development and test environments
System homogenization

- Two approaches to operate and control the life cycle of services
  - Custom-built infrastructure for short-lived application services (e.g., event building and read-out)
  - Systemd for long-running application services (e.g., monitoring)

- Restructuring the existing system into a homogeneous, scalable cloud architecture adopting a uniform paradigm where all applications are orchestrated in an environment with standardized facilities
Container-based architecture

- In this new paradigm DAQ applications are organized as groups of containers and the required software is packaged into container images.

- Automation of all aspects of coordinating and managing containers is provided by the Kubernetes environment, where a set of physical and virtual machines is unified in a cluster of compute resources.

- More reliable validation of software in test or production environments:
  - Consistent installation on different targets is guaranteed by image construction at build time.
  - Tests in production environment do not disrupt installation, roll back of installed software is not needed.
Containerization transforms the network from being **machine-oriented** to being **application-oriented**

**Container image encapsulates** all (or almost all) of an application’s dependencies
Pilot project
Goals

- Achieve extensive use of the cloud approach by homogenizing DAQ system elements into Kubernetes patterns.
- Adopt a full software life-cycle for the new environment, focusing on the development, delivery, deployment and operation.
- Resolve issues concerning building and maintenance of Kubernetes cluster and hardware resources. Determine implications on current system administration environment.
Final output would be to have a working **DAQ column** from detector front-end to High Level Trigger including run control and monitoring
Scope of the project

- The new approach applies to all dimensions of expertise within CMS DAQ system
  - Operation
  - Development
  - Release
  - Deployment
  - System administration
  - Networking
  - Security
  - Integration
Study modules

- Performance and scalability
- Hardware access and binding (RDMA, FEROL, PCI, VME, etc.)
- System fine-tuning (driver interrupts, NUMA settings, etc.)
- User access (GUI, CLI, Dashboards, API, etc.)
- Networking (configuration, fine-tuning TCP/IP, RoCE, etc.)
- Control and monitoring
- Configuration
- Additional technologies and their integration (Elasticsearch, Oracle, PVSS, etc.)
- Scalable Event Builder prototype and startup measurements
What was achieved already?

- User access possibilities through GUI, CLI, Dashboards and Kubernetes API were investigated
- RDMA based device access (RoCE) investigated and prototyped
- Various networking possibilities were investigated and prototyped (flannel, Calico, ipvlan)
- Software configuration techniques inside a pod were tested: based on custom scripting and/or Helm scripting
- Elasticsearch and OpenSearch were installed and tested inside Kubernetes cluster for monitoring purposes
- Several Event Building prototypes working inside Kubernetes cluster were made (see also presentation “Event Building studies for CMS Phase-2 at CERN” by Andrea Petrucci and Rafał Krawczyk)
- Startup/termination time measurements were performed
Startup/termination measurements

- Event builder is designed as a number of processing nodes communicating to each other to aggregate event fragments.

- Measurements of startup time up to the moment when the full pod interconnection is achieved and event builder is ready to take data.

- Measurements of termination time for all pods after ready state is achieved.
Test cases

- helm install
- chart deployed
- started
Test cases

- helm install
- chart deployed
- pods are scheduled to worker nodes

started

pod scheduled
Test cases

- helm install
- chart deployed
- pods are scheduled to worker nodes
- all containers were created and running

- started
- pod scheduled
- containers ready
Test cases

- helm install
- chart deployed
- pods are scheduled to worker nodes
- all containers were created and running
- started
- pod scheduled
- containers ready
- app alive

process is alive
Test cases

started
pod scheduled
containers ready
app alive
pod connectable
Test cases

- helm install
- chart deployed
- pods are scheduled to worker nodes
- all containers were created and running
- all DNS names (inside k8s cluster) resolved to IP addresses
- configuration is loaded, Mellanox RDMA plugin loaded and configured

started
pod scheduled
containers ready
app alive
pod connectable
app ready
Test cases

- **helm install**
- **chart deployed**
- **pods are scheduled to worker nodes**
- **all containers were created and running**
- **all DNS names (inside k8s cluster) resolved to IP addresses**
- **configuration is loaded, Mellanox RDMA plugin loaded and configured**
- **DAQ is ready to take data. IBV QPs are created and connected**

- **started**
- **pod scheduled**
- **containers ready**
- **app alive**
- **pod connectable**
- **app ready**
- **app connected**
Test cases

- helm install
- chart deployed
- pods are scheduled to worker nodes
- all containers were created and running
- all DNS names (inside k8s cluster) resolved to IP addresses
- configuration is loaded, Mellanox RDMA plugin loaded and configured
- DAQ is ready to take data, IBV QPs are created and connected

- process is alive

- started
- pod scheduled
- containers ready
- app alive
- pod connectable
- app ready
- app connected
- app uninstall
- uninstall time
Startup time (one pod per node)
Startup time (several pods per node)
Termination time

![Graph showing termination time vs. number of pods]
Conclusion
Conclusion

- Encouraging results were achieved so far, containerization was shown to be an appealing technology for DAQ applications as a replacement for a bare metal infrastructure.

- Containerization and orchestration solve most of constraints experienced by running with a traditional infrastructure.

- The approach fits well with distributed inter-communicating applications such as event builder.

- No limitations of the approach were discovered during investigation of predefined objectives.
Future work

- Continue working on unresolved study modules
- Finalize a working DAQ column from detector front-end to High Level Trigger
Questions?