Optimizing Provisioning of LCG Software Stacks with Kubernetes

Authors: J Heinz[1], R Bachmann[1], G Ganis[1], P Mato[1], D Konstantinov[2], I Razumov[2]
[1]CERN [2]Institute for High Energy Physics of NRC Kurchatov Institute

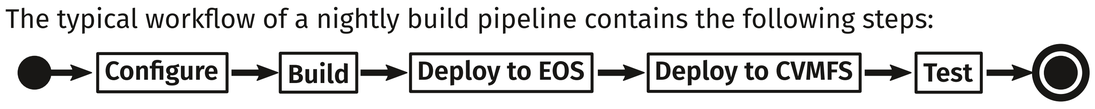
project-lcg-spi-internal@cern.ch

Introduction to the LCG use case

The LCG software stack[1] contains almost 450 packages available for several compilers, operating systems, Python versions and hardware architectures. Among these packages are Monte Carlo generators, machine learning tools, Python modules and HEP specific software. Some of its users are:

Along with several releases per year, 31 development builds are provided each night to allow for quick updates and testing of new versions of ROOT, Geant4, etc. It also provides the possibility to test new compilers and configurations.

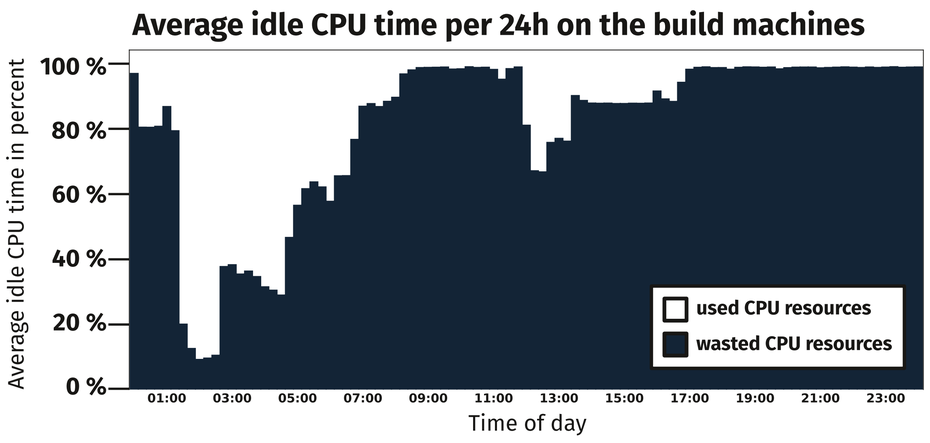
The typical workflow of a nightly build pipeline contains the following steps:



This process is currently automated using a Jenkins continous integration server and 20 virtual machines that function as build slaves for the Jenkins master. These OpenStack machines are running CERN CentOS 7 and are configured using ***Puppet***. They provide a Docker daemon for the build process as well as ***Kerberos*** and ***CVMFS*** integration outside the Docker environment that are bind-mounted for build and deployment.

Goal: Saving resources through orchestration

The goal of moving from VMs to Kubernetes is to ***reduce wasted resources*** and share resources across the CERN datacenter with other groups and experiments. This can be realized by the cluster auto-scaling feature of Kubernetes that can automatically spawn and delete worker nodes as needed. The current CPU usage is displayed in the plot below:

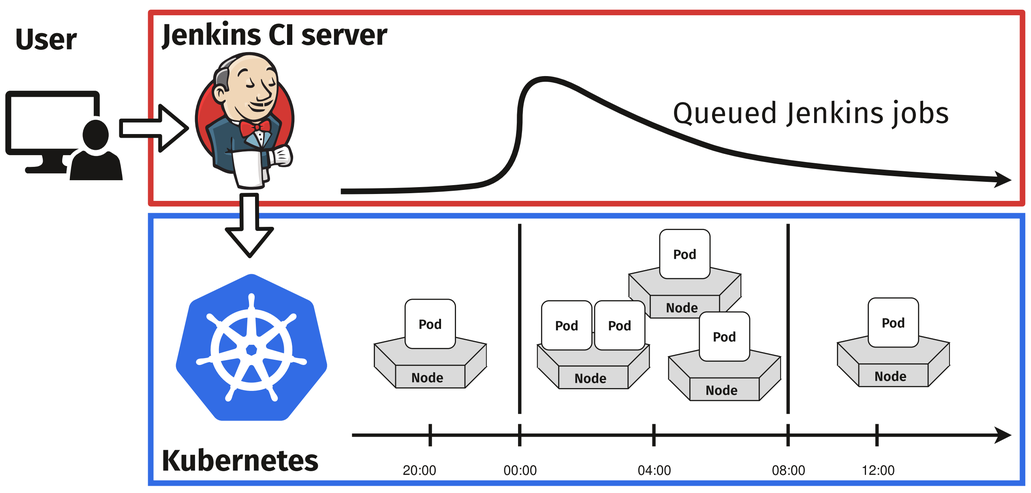


Kubernetes also serves as an additional ***abstraction layer*** that takes care of the underlying operating system, scheduling and cluster management. A user of Kubernetes only needs to provide a suitable container image which is launched inside a ***pod***, the smallest computing unit in the Kubernetes ecosystem.

Vision for a new build infrastructure

To achieve this goal, the following setup is envisioned:

- Keep a Jenkins CI server as a single point of entry to trigger build jobs, manage configurations via variables and store all necessary secrets such as passwords.
- Use Kubernetes like a batch system similar to HTCondor to provide the necessary resources on demand and scale down after the builds are done



- Move environment definitions from the Docker host system inside the container images
- Follow Red Hat's ***single concern principle***[2] to use specialized container images for building, testing and deployment instead of general purpose images

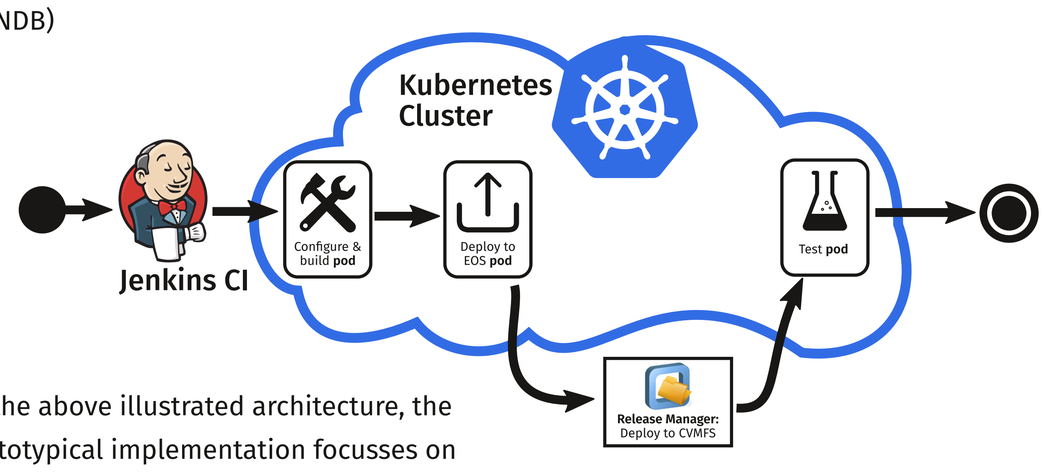
State of the prototype

Currently, there is only a test cluster that is not yet used in production for nightly builds

CERN Infrastructure

The CERN IT deparment provides documentation[3] about the usage of Kubernetes clusters on top of the ***CERN OpenStack*** service. ***CVMFS read access*** was tested successfully thanks to the CernVM-FS CSI driver[4].

Auto-scaling of the cluster works within the quota of the project. If no free node is available, a new one is created in 4 to 12 minutes, depending on the DNS configuration (LANDB)



Of the above illustrated architecture, the prototypical implementation focusses on the first pod which is responsible for the configurration and build of the software stack.

Jenkins integration

The integration into the Jenkins CI landscape is the defining challenge of this prototype because of the need to adapt to complex pre-existing workflows. This includes the configuration of jobs, getting their log files and sending Kerberos tickets.

There are two different approaches to solve this problem which are compared below:

Kubernetes plugin for Jenkins	Kubernetes API (HTTP or kubectl)
Integrates with existing Jenkins instance	Lacks mechanism for returning status and logs to Jenkins synchronously
Requires definition of Groovy pipeline	Can be used from any shell environment
Dependence on external project	Greater internal development workload

Shortcomings and limitations

- The cluster is limited to x86-64 architecture which excludes i386, ARM and PowerPC architectures. There's also no support for Windows or macOS.
- The Kubernetes cluster is not managed by IT directly. Therefore the responsibility for maintaining and monitoring the cluster falls to the project as an additional workload.
- Deployment to CVMFS cannot be done easily within Kubernetes because it still relies on an external pre-configured release manager VM.
- Depending on the flavor (CPU, memory) of the Kubernetes worker nodes, the local SSD might be too small. This can be mitigated by adding an external volume which can slow down the I/O performance during build time.

Future outlook

The Kubernetes service at CERN will provide some features in the near future that help to mitigate some of the problems mentioned above:

- Decrease delay when creating new nodes by avoiding the DNS and working with IPs
- Managed upgrades for clusters will reduce the maintenance workload and simplify the management of a Kubernetes cluster

References

[1] Patricia Mendez Lorenzo, CHEP 2018: *Bulding, testing and distributing common software for the LHC experiments*
[2] Bilgin Ibryam (Red Hat), 2017: *Principles of container-based application design*
[3] *CERN CloudDocs:* https://cern.ch/clouddocs
[4] *CernVM-FS CSI driver:* https://github.com/cernops/cvmfs-csi

Scan to download this poster as PDF