**CI / CD PIPELINE NOTES**

**INTRODUCTION:**

A CI/CD pipeline is a series of steps that must be performed in order to deliver a new version of software. [Continuous integration/continuous delivery (CI/CD](https://www.redhat.com/en/topics/devops/what-is-ci-cd)) pipelines are a practice focused on improving software delivery using either a [DevOps](https://www.redhat.com/en/about/videos/learn-cloud-native-series-what-is-devops) or [site reliability engineering (SRE)](https://www.redhat.com/en/topics/devops/what-is-sre) approach.

A CI/CD pipeline introduces monitoring and [automation](https://www.redhat.com/en/topics/automation) to improve the process of application development, particularly at the integration and testing phases, as well as during delivery and deployment. Although it is possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through automation.

**CI/CD PIPELINE USING GIT, JENKINS AND MAVEN:**

**GIT:**

**CI/CD pipeline set-up is simple**: GitHub Actions is made by and for developers, so you don’t need dedicated resources to set up and maintain your pipeline. There’s no need to manually configure and set up CI/CD. You don’t have to set up webhooks, you don’t have to buy hardware, reserve some instances out there, keep them up to date, do security patches, or spool down idle machines. You just drop one file in your repo, and it works.

**Respond to any webhook on GitHub**: Since GitHub Actions is fully integrated with GitHub, you can set any webhook as an event trigger for an automation or CI/CD pipeline. This includes things like pull requests, issues, and comments, but it also includes webhooks from any app you have integrated into your GitHub repository. Let’s say you’re going to use any one of the many tools that are out there to run part of your development pipeline. With GitHub Actions, you can trigger CI/CD workflows and pipelines of webhooks from these apps (even something simple, like a chat app message, if you’ve integrated your chat app into your GitHub repository, of course).

**JENKINS:**

Jenkins is an open-source Continuous Integration server that helps to achieve the Continuous Integration process (and not only) in an automated fashion. Jenkins is free and is entirely written in Java. Jenkins is a widely used application around the world that has around 300k installations and growing day by day.

**Features:**

* Jenkin will build and test code many times during the day.
* Automated build and test process, saving timing, and reducing defects.
* The code is deployed after every successful build and test.
* The development cycle is fast.

**MAVEN:**

**Maven** is a [build automation](https://en.wikipedia.org/wiki/Build_automation) tool used primarily for [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) projects. Maven can also be used to build and manage projects written in [C#](https://en.wikipedia.org/wiki/C_Sharp_(programming_language)), [Ruby](https://en.wikipedia.org/wiki/Ruby_(programming_language)), [Scala](https://en.wikipedia.org/wiki/Scala_(programming_language)), and other languages. The Maven project is hosted by the [Apache Software Foundation](https://en.wikipedia.org/wiki/Apache_Software_Foundation), where it was formerly part of the [Jakarta Project](https://en.wikipedia.org/wiki/Jakarta_Project).

Maven addresses two aspects of building software: how software is [built](https://en.wikipedia.org/wiki/Software_build), and its dependencies. Unlike earlier tools like [Apache Ant](https://en.wikipedia.org/wiki/Apache_Ant), it uses conventions for the build procedure. Only exceptions need to be specified. An [XML](https://en.wikipedia.org/wiki/XML) file describes the software project being built, its dependencies on other external modules and components, the build order, directories, and required [plug-ins](https://en.wikipedia.org/wiki/Plug-in_(computing)). It comes with pre-defined targets for performing certain well-defined tasks such as compilation of code and its packaging. Maven dynamically downloads [Java](https://en.wikipedia.org/wiki/Java_(programming_language)) libraries and Maven plug-ins from one or more repositories such as the Maven 2 Central Repository, and stores them in a local cache.[[2]](https://en.wikipedia.org/wiki/Apache_Maven#cite_note-maven2repo-2) This local cache of downloaded [artifacts](https://en.wikipedia.org/wiki/Artifact_(software_development)) can also be updated with artifacts created by local projects.

**DOCKER:**

Docker, a subset of the [Moby](http://mobyproject.org/) project, is a software framework for building, running, and managing containers on servers and the cloud. The term "docker" may refer to either the tools (the commands and a daemon) or to the **Dockerfile** file format.

It used to be that when you wanted to run a web application, you bought a server, installed [Linux](https://opensource.com/resources/linux), set up a LAMP stack, and ran the app. If your app got popular, you practiced good [load balancing](https://opensource.com/article/21/4/load-balancing) by setting up a second server to ensure the application wouldn't crash from too much traffic

Dockers help developers to build their code and test their code in any environment to catch bugs early in the application development life cycle. Dockers help streamline the process, save time on builds, and allows developers to run tests in parallel.

Times have changed, though, and instead of focusing on single servers, the Internet is built upon arrays of inter-dependent and redundant servers in a system commonly called "the cloud". Thanks to innovations like Linux [kernel namespaces and cgroups](https://opensource.com/article/19/10/namespaces-and-containers-linux), the concept of a server could be removed from the constraints of hardware and instead became, essentially, a piece of software. These software-based servers are called [containers](https://opensource.com/article/18/1/history-low-level-container-runtimes), and they're a [hybrid mix](https://opensource.com/article/18/11/behind-scenes-linux-containers) of the Linux OS they're running on plus a hyper-localized runtime environment (the contents of the container).

**ANSIBLE:**

Ansible is a powerful tool for IT automation and can be used in a CI/CD process to provision the target environment and to then deploy the application on it. Jenkins is a well-known tool for implementing CI/CD. Shell scripts are commonly used for provisioning environments or to deploy apps during the pipeline flow.

### **Design goals:**

The design goals of Ansible include:

* Minimal in nature. Management systems should not impose additional dependencies on the environment.
* Consistent. With Ansible one should be able to create consistent environments.
* Secure. Ansible does not deploy agents to nodes. Only [OpenSSH](https://en.wikipedia.org/wiki/OpenSSH) and [Python](https://en.wikipedia.org/wiki/Python_(programming_language)) are required on the managed nodes.
* Reliable. When carefully written, an Ansible [playbook](https://en.wikipedia.org/wiki/Ansible_(software)#Playbooks) can be [idempotent](https://en.wikipedia.org/wiki/Idempotent), to prevent unexpected side-effects on the managed systems.It is possible to write playbooks that are not idempotent.
* Minimal learning required. Playbooks use an easy and descriptive language based on [YAML](https://en.wikipedia.org/wiki/YAML) and [Jinja templates](https://en.wikipedia.org/wiki/Jinja_(template_engine)).

**KUBERNETES ON AWS:**

Kubernetes is open source software that allows you to deploy and manage containerized applications at scale. Kubernetes manages clusters of Amazon EC2 compute instances and runs [containers](https://aws.amazon.com/what-are-containers/) on those instances with processes for deployment, maintenance, and scaling. Using Kubernetes, you can run any type of containerized applications using the same toolset on-premises and in the cloud.

AWS makes it easy to run Kubernetes in the cloud with scalable and highly available virtual machine infrastructure, community-backed service integrations, and [Amazon Elastic Kubernetes Service (EKS)](https://aws.amazon.com/eks/), a certified conformant, managed Kubernetes service.

**HOW KUBERNETES WORKS?**

Kubernetes works by managing a cluster of compute instances and scheduling containers to run on the cluster based on the available compute resources and the resource requirements of each container. Containers are run in logical groupings called pods and you can run and scale one or many containers together as a pod.

Kubernetes control plane software decides when and where to run your pods, manages traffic routing, and scales your pods based on utilization or other metrics that you define. Kubernetes automatically starts pods on your cluster based on their resource requirements and automatically restarts pods if they or the instances they are running on fail. Each pod is given an IP address and a single DNS name, which Kubernetes uses to connect your services with each other and external traffic.

**INTEGRATE KUBERNETES:**

CI/CD workflows and cloud native systems have a few goals in common: they both try to increase development velocity, optimise software quality and maintain its operability. CI/CD automates many steps from when code is developed to the point it is released in production. Similarly, Kubernetes automates container deployments across various infrastructure environments and ensures efficient resource utilisation. Therefore, it naturally makes sense for organisations to set up CI/CD pipelines that leverage the Kubernetes platform.

GitLab is one of the most popular solutions if you are looking to do exactly that. It comes with a built-in container registry and can integrate with Kubernetes in three ways:

* Software built in a GitLab CI pipeline can be deployed to Kubernetes as part of the CD stage
* Kubernetes can manage batch job executions that are linked to a GitLab instance
* GitLab instances themselves can be run on a Kubernetes cluster