# My Take on CI/CD Pipeline Setup for Microservices

CI/CD setup require a bunch of tools and infrastructure involvement. If we choose any cloud CI tool we get the agents on demand which has capabilities of building our defined requirements. Where as on On-Prem we may need to setup the tooling manually or use any configuration management tool like puppet or ansible.

So lets start talk about Continuous Integration First

This includes multiple jobs/task

- Assuming the source code is being tracked by a source control versioning tool like (git/subversion..etc) and repository to store our code base like (github,bitbucket..etc)
- Create a **Web hook** (trigger) on Source Control Management tool to notify the CI tool to kick off the build (which will raise an event and interested subscribers like our CI tool would listen for the same)
  We may need to create multiple web hooks depends on branching strategy on SCM (feature,develop,master…)

  As microservices architecture span into multiple services ,we can build a reusable pipeline, so that all the services can leverage them rather than building each pipeline for reach service / repo.

  From the web hook or trigger we can pass an argument on which repo/svc build should kick off

- Get a **service/bot account** created which should have access grant to some tools, which will require in later steps (agent,artifactory etc..)
- Clone the repository and checkout the code
- Run a package install if any and **build aka compilation** targeting to respective application runtime (ex: Node JS,.Net Core…)
- Run **unit tests** if any
- Collect the test report and submit to **code quality analyzer** tools like Sonar
- **Publish packages** if any
- Push **packages** to artifactory (Jfrog etc..)
- **Publish** the project binaries
- Run a **container build** (like docker) tagging with your app name and version with the build number (which is variable for a project) if the app is containerized
- **Push container images** to artifactory aka registry
- Perform a **security scan** with tools like checkmarx
- Create Helm packages , prepare release artifacts
- Collect artifacts if any
- We may also notify the SCM about the builds status

All the above steps are available in form of plugins in the CI tool, if it’s a custom requirement get a plugin built or just ssh to agent run the necessary job.

# My Take on CI/CD Pipeline Setup for Microservices

Continuous Deployment / Continuous Delivery

- Assuming you got all the infrastructure setup ready required for deployment (K8s cluster ..etc)
- Deployment can be done with out using any CD tools but it hard to manage to versions and roll backs, which in turn end with a lot of scripts. So better to go with the deployment tool which would offers great features
- Usually a real time application is hosted on multiple environments (Dev,Test,Prod..), each may have a different settings and configurations. So we need a manifest files for respective environments to hold the config. These manifest reside along with our code base
- Add a remote trigger if the tool support, which will notify the CD process to kick off the deployment, after a green build on a main branches like 'develop' / 'master' deploy will trigger
- Add an auto promotion trigger which will promote the release to further environments, incase of continuous delivery we off the trigger for PROD, where we define a gates / approvals
- Some tools offers CD support in form of plugins, which may not have all the features we expect, so we need develop a plugin or choose a CD tool which offer features we're looking for
- For containerized apps we require an orchestrator to spin off and manage containers, without an orchestrator it would work but tools like k8s comes with great features like self healing, scaling, rolling updates etc..
- Lets take k8s as an example, it comes with an great API to interact and to trigger our deployments. Using kubectl which is a client we can deploy by targeting manifest files(yaml or json). Which has all the necessary kube rest object like service,deployment …
- Microservice architected application would end up with many small services where each does a single job , we may tend to deploy all the relevant services as a stack. This can be done by tool like helm which will package all the manifest and run.
- Same steps are applicable for all the environments
- As we expect changes in our code and DB, We require a DB change management tool kind of liquibase. Which run our scripts and maintain the log history. Rollbacks can be done easily
- Lastly trigger the Integration tests, which will validate our deployment
- Get some application monitoring tool set up, tool like Grafana emits the metrics and dashboard can be built with that. Where we can monitor the live traffic and take a decision on scaling the app.
- On every infrastructure node we require to have a hardware monitoring tool which will notify the intended team on failure or unusual behavior