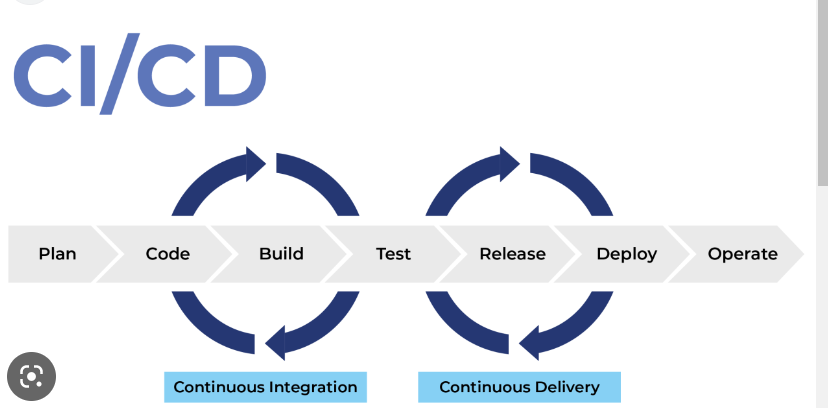
Continuous integration (CI) executes the sequence of steps required to build and test the project. CI runs automatically on every change committed to a shared repository, offering developers quick feedback about the project’s state.

Continuous delivery is an extension of CI. Its goal is to automate every step required to package and release a piece of software. The output of a continuous delivery pipeline takes the form of a deployable binary, package, or container.

Continuous deployment is an optional step-up from continuous delivery. It is a process that takes the output from the delivery pipeline and deploys it to the production system in a safe and automated way.



* **Unit tests**: validate that functions or classes behave as expected.
* **Integration tests**: are used to verify that the different components of an application work well together.
* **End-to-end tests**: check an application by simulating user interaction.
* **Static tests**: finds defects in code without actually executing it.
* **Security tests**: scans the application’s dependencies for known security issues.
* [**Smoke tests**](https://semaphoreci.com/community/tutorials/smoke-testing): fast tests that check if the application can start and that the infrastructure is ready to accept deployments.

**Plugins in Jenkins  
-----------------------------**

**1.Github**

**2.Amazon EC2**

**3.Build pipeline**

**4.Docker**

**5.Kubernetes**

**6.Mailer plugin**

**7.Docker**

Graphical user interface, application

Description automatically generated

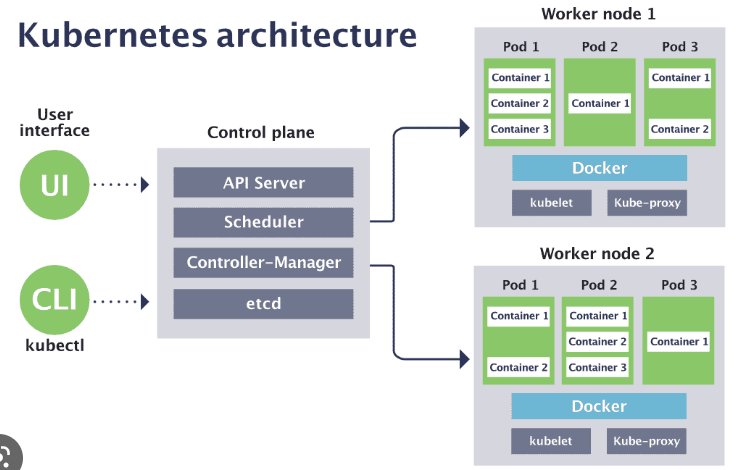
Timeline

Description automatically generated

**GIT all commands**

**-------------------------------**

[**https://dzone.com/articles/top-20-git-commands-with-examples**](https://dzone.com/articles/top-20-git-commands-with-examples)



**Executor node: (This runs on master node)**

* Kube-proxy: This service is responsible for the communication of pods within the cluster and to the outside network, which runs on every node. This service is responsible to maintain network protocols when your pod establishes a network communication.
* kubelet: Each node has a running kubelet service that updates the running node accordingly with the configuration(YAML or JSON) file. NOTE: kubelet service is only for containers created by Kubernetes.

**Master services:**

* Kube-apiserver: Master API service which acts as an entry point to K8 cluster.
* Kube-scheduler: Schedule PODs according to available resources on executor nodes.
* Kube-controller-manager:  is a control loop that watches the shared state of the cluster through the apiserver and makes changes attempting to move the current state towards the desired stable state

How to fix Kubernetes not ready state error

**MemoryPressure**—node is running out of memory.  
**DiskPressure**—node is running out of disk space.  
**PIDPressure**—node is running too many processes.

Kubectl describe node nodename

Then journalctl -u kubelet – to restart the service

<https://komodor.com/learn/how-to-fix-kubernetes-node-not-ready-error/#:~:text=A%20Kubernetes%20node%20is%20a,be%20used%20to%20run%20pods>.

<https://www.redhat.com/sysadmin/kubernetes-troubleshooting> - for OOM errors

<https://yankeexe.medium.com/how-rolling-and-rollback-deployments-work-in-kubernetes-8db4c4dce599>

<https://katharharshal1.medium.com/kubernetes-objects-pods-replicasets-and-deployments-44f16b38d9f7>

As is turns out, ReplicaSets are actually what Deployments create when you deploy them. You can deploy ReplicaSets directly, but they don’t have rolling update functionality on their own. They instead deploy pods, and it’s the Deployments job to manage the rolling updates (via rolling the update of the ReplicaSets), which in turn causes a rolling update of the pods.

Stateful applications **save data to persistent disk storage for use by the server, by clients, and by other applications**. An example of a stateful application is a database or key-value store to which data is saved and retrieved by other applications

A stateless application is **one which depends on no persistent storage**. The only thing your cluster is responsible for is the code, and other static content, being hosted on it. That's it, no changing databases, no writes and no left over files when the pod is deleted.

Jenkins task

we are doing deployement using jenkins like for example to create war files.

Github repo to checkout.

configuration:

1.discard old build to 10

2.log rotation

Poll SCM periodically polls the SCM to check whether changes were made (i.e. new commits) and builds the project if new commits where pushed since the last build, whereas build periodically builds the project periodically even if nothing has changed.

Deploy container dockerfile to Kubernetes

* 1. Create dockerfile and write below commad:
  2. Text

     Description automatically generated
  3. Docker build -t apacheserver .
  4. Docker run -d –-name apache1 apacheserver1
  5. Docker container ls
  6. Git clone urlname – to copy the code from git
  7. Create a directory mydir/helloworld
  8. Docker build -t path- this will locate dockerfile in the path
  9. Docker images
  10. Docker run -p 8080:8080 path
  11. curl <http://localhost:8080-> **Image build and creation done**
  12. push dcoker image now
  13. docker push *REGION*-docker.pkg.dev/${PROJECT\_ID}/hello-repo/hello-app:v1
  14. Kubernetes create deployment helloapp --image=pathname
  15. kubectl scale deployment hello-app --replicas=3
* kubectl get pods
* NAME READY STATUS RESTARTS AGE
* hello-app-784d7569bc-hgmpx 1/1 Running 0 90s
* hello-app-784d7569bc-jfkz5 1/1 Running 0 90s
* hello-app-784d7569bc-mnrrl 1/1 Running 0 95s

1. kubectl expose deployment hello-app --name=hello-app-service --type=LoadBalancer --port 80 --target-port 8080
2. kubectl get service
3. if new version is release then first push to repo then use below command to upgrade
4. kubectl set image deployment/hello-app hello-app=REGION-docker.pkg.dev/${PROJECT\_ID}/hello-repo/hello-app:v2
5. other way you can use yml file to create deployment:

|  |
| --- |
| apiVersion:apps/v1 |
|  | kind: Deployment |
|  | metadata: |
|  | name: helloweb |
|  | labels: |
|  | app: hello |
|  | spec: |
|  | selector: |
|  | matchLabels: |
|  | app: hello |
|  | tier: web |
|  | template: |
|  | metadata: |
|  | labels: |
|  | app: hello |
|  | tier: web |
|  | spec: |
|  | containers: |
|  | - name: hello-app |
|  | image: us-docker.pkg.dev/google-samples/containers/gke/hello-app:1.0 |
|  | ports: |
|  | - containerPort: 8080 |
|  | resources: |
|  | requests: |
|  | cpu: 200m |
|  | # [END container\_helloapp\_deployment] |
|  | # [END gke\_manifests\_helloweb\_deployment\_deployment\_helloweb] |

Default rolling strategy

1. [**Rolling deployment**](https://spot.io/resources/kubernetes-autoscaling/5-kubernetes-deployment-strategies-roll-out-like-the-pros/#a1)—replaces pods running the old version of the application with the new version, one by one, without downtime to the cluster.
2. [**Recreate**](https://spot.io/resources/kubernetes-autoscaling/5-kubernetes-deployment-strategies-roll-out-like-the-pros/#a2)—terminates all the pods and replaces them with the new version.
3. [**Ramped slow rollout**](https://spot.io/resources/kubernetes-autoscaling/5-kubernetes-deployment-strategies-roll-out-like-the-pros/#a3)—rolls out replicas of the new version, while in parallel, shutting down old replicas.