**AZURE DEVOPS**

**DEVOPS:**

SaaS platform to implement all devops operations.

Covers the entire SDLC.

Components:

**1.Boards:** is used to implement the SDLC that we follow. May be scrum, agile or other.

* It consists of features, tasks, bugs and can be assigned to different people.
* Brings developers and testers together.
* Can see the status of the task. Is it approved, pending or finished?

**2.Repos:** Source control management

* Continuous communication and collaboration
* Only when all developers check and approve, pull can be done.
* Once the code is built, it should be tested, released, packaged and deployed

**3. Pipelines:**

Test -> package -> create a docker image -> push to docker repo & use during deployment

* Runs Jobs. Job : group of multiple steps
* If one single job , no need to explicitly mention the job in the pipeline.
* Else need to specify the ordering of the jobs
* Jobs are run using agent.
* Agent is selected in an agent po(basically a server) l which can be windows, Linux or MacOS
* Multiple jobs for different environments is to run the tests parallelly
* If everything went well,
* create artifacts and make it ready for deployment

**4.Azure Artifacts:**

* Allows developers to share their code effectively and manage all their packages at one place
* When an application needs some packages to support it , they install it from here.
* Ex: Maven package for Java appln , npm for node js appln etc.
* It can be in different formats depending on the type of application
* Ex: Jar if its java, nuget if .net, zip if js

After the build, we will deploy it to 3 different environments: development , testing and production

**YAML** scripting helps in creation of azure pipeline templates which can be reused for any job in any environment

**Clone vs Fork:**

1.Clone: copies the entire project to the local repo i.e to persona’s computer

2.When changes are made this can be pushed only if he has the contributor authority

3.When changes need to be made by the newly joined developer on the same project without the need of contribution

1.Fork makes a new copy of the project in the remote repo itself i.e in the separate github account.

* This is used when a developer needs to make some changes to the project without affecting the original project.
* All the commits are saved separately in a branch.
* It can create a pull request, since it doesn’t have the write access to push the code to the original repo.
* Original project owner gets a pull request, he can review and comment it. He can approve the pull request and merge it.
* Now all the commits made initially are reflected into the master branch.

2.When changes are made permission need to be given explicitly to access the forked project

3.When the entire project needs to be started from scratch

**Fetch, Pull & Merge:**

Fetch: tells that there are changes in the remote repo without bringing the changes to the local repo (when changes are made in the remote repo rather than local)

Merge: to bring the changes in the remote repo to local we use merge

Pull: Copies the copy of the remote repo to local repo

Git pull = git fetch + git merge

**Agent pool:**

Every pipeline runs on an agent

**Azure pipelines** is default agent pool from Microsoft consisting of various agents

All the jobs either build or release are running on the same server provided by the agent i.e., azure pipelines from Microsoft

**Agent** is a server on which we are going to run the jobs

**2 types of agents:**

1.Microsoft hosted agent: provided by Microsoft and a default agent

Each time previous VM is discarded and a new VM is assigned by default.

Configuration and upgrades need not be taken care of.

2.Privately hosted agent/self-hosted agent: agent which can be added on our own. It can have different accesses like read, write, manage, build, release and test. Agent that’s set up and managed on our own to run jobs.

Why do we need self-hosted agents when we have default?

When we want to install dependant software’s for builds or deploys, we prefer self-hosted agents.

**Pipeline:**

* A set of automated processes which helps in building or deploying the code to production env.
* It combines CI and CD to build, test and ship code to any target
* Cloud or on premise
* To run jobs either individual or multiple

8 stages of pipeline:

**Development:** Plan, Code, Build and Test

**Operations:** Release, Deploy, Operate and Monitor

**Few DevOps tools:**

Git, Azure , Jenkins , ansible , docker ,Nagios ,selenium

**Few CI/CD tools:**

Jenkins, Codeship , GitLab

**Few Continuous Deploy tools:**

Jenkins, Azure pipelines, AWS code deploy

Containers:

Package the SW code, configurations, dependencies into a single unit. Multiple containers can run on a single machine and share the same OS.

**Ex: Docker , Azure Kubernetes, Asp.Net**

What is the use of selenium in azure devops?

For continuous testing .regression and functional.

Release: Collection of changes for the user to experience

Deploy : Shifting of applications from one env to another

Build pipeline vs. Release pipeline:

Jenkins:

Jenkins is a open source on premise tool which is used for CI/CD.

Changes made to code, tracked by the Jenkins, builds the code, if successful creates the executables and deploys it

**Jenkins vs Pipelines:**

|  |  |
| --- | --- |
| 1.Open source on premise | 1.Cloud or on premise |
| 2. Has wide range of plugins to support | 2. Supports only Microsoft supported plugins |
| 3.Huge support since open source | 3.Support is less |
| 4.Integration is easy with any products | 4. Integration with non-Microsoft products is difficult |
| 5. This has scripted pipelines coded in Groovy | 5. Supports any application like node.js, python, java etc. |
| 6. One pipeline for whole processing | 6. Two different pipelines for release and deployment |

Pipeline is suitable for faster deployments

Jenkins is suitable for complex deployments

Agent Working:

When a pipeline is run, entire source code of the repo is downloaded to the agent. It will have folders a(artifacts), b(binaries), c(source code) and test results.

Initially agent accesses the repo , downloads the code and runs the specified jobs.

Inside a pipeline, we will have jobs created. Agent runs one job at a time.

Azure roles:

1.Owner: access to change the resource and grant permissions to another user.

2.Contributor: access to change the resource.

3.Read: Read only access to resource.

Docker Container:

* Is a VM environment used in application development.
* To create, run and deploy applications that are isolated from hardware environment

Docker Image:

* It’s a snapshot of the docker container at specific point of time
* It’s immutable
* It can be shared, deleted and duplicated
* Since it’s immutable, it’s used for testing new SW or configurations
* It has everything that’s needed to run an application, conf files ,env variables etc.
* When an image is deployed to docker environment, it can be executed as a docker container

**Linking git to azure:**

Go to Settings -> git connection -> add the git account ->authorize -> save the repos you want to work on.

Go to Repo -> Files -> Import repo -> select the repo from the git ( choices available) ->import

To Clone the repo to local:

Copy the clone repo path from code under the project

1. In Bash terminal enter : **git clone repo path**
2. To check the files enter: **ls**
3. To open any file : **nano File.txt (**Make changes and save it)
4. To check the status : **git status**
5. To add the committed changes : **git add .** (stashing)
6. To commit the changes: **git commit -m “commit msg”**
7. To push the file : **git push**
8. Check the changes in the remote repo
9. To check in which file path you are : **use pwd cmd in Bash**
10. To navigate to the project folder : **cd Project Name**

To pull the changes from remote to local:

Make changes in the file in remote repo

**Commit the changes**

1. On bash : **git status** = up to date
2. On bash : **git fetch** = some changes made
3. On bash : **git pull**
4. On bash : **nano File.txt** = updated changes are seen

Else

After **git fetch** , perform **git merge** instead of **git pull** , result is the same.

**Adding new file/folder in remote repo:**

1. Added new File2.txt in remote repo. Committed the changes remotely
2. git status : up to date
3. git fetch : some changes made
4. ls : lists only File1.txt
5. git merge : merges changes from remote
6. ls : lists both the files File1.txt and File2.txt

**Branches:**

1. Create a new branch and update the file .
2. Create a pull request to merge the changes to the master branch
3. Approve and complete the process
4. Can sustain the new branch after it has been merged to master or can delete it if no longer needed
5. Now check the master branch, the file will have the updated changes

**Feeds:**

Is a folder group ,manage, storing of packages and decides whom to share it with

**Views:**

Sharing options available for the artifacts like @local,@prerelease ,@release

Beta – still at testing

**Build artifacts:**

Are the files that your build produce .Eg: .exe or .dll

**Pipeline artifacts:**

Stores build output

**Connecting to Azure Pipeline:**

Application itself is an artifact and it is released into a server and then available for testing

**Exploratory Testing :**

Exploring the application and providing the screenshots and videos of the bugs and raise a bug to the developer

Feature Flags:

It allows us to show or hide the features in an app

Exploratory Testing:

1. We should add the plugin to chrome **Test & Feedback**
2. Connect it to the azure DevOps organization by providing the URL (Organization URL)
3. Once done , navigate to the application page
4. Has options like screenshot , screen record , capture the entire screen , capture the application if running behind, add a new test case , create a bug etc.
5. Can see all the screenshots added in the history .
6. Once you add a new test case , it gets added to the history , clicking on the same will take you to the newly added test case inside the azure DevOps test plan.

This makes it easy for testing and also saves time .Can add more work items there itself

Why should we use azure pipelines?

1. Supports multiple OS
2. Supports multiple languages
3. Supports version control system
4. Works with open source projects
5. Deploys to multiple environments at the same time

**Pipeline Architecture:**

Trigger -> triggers the pipeline.

Pipeline - > contains multiple stages & can be deployed to multiple environments.

Stage -> Each stage has multiple agents

Agents -> Each agent has one / more jobs

Job -> Each job has one or more steps . Job can be agentless also

Step -> can be a script or a task

Task - > this is the piece of script that performs some action

Artifacts -> is a package or a file generated from the run of tasks

**YAML :**

* Human readable data serialization language. Operations can be performed using GUI also .
* YAML gives more automated approach since scripting can be done for the jobs to run be run one after the other.
* Output of one job is an input for the next and can be lined in this fashion
* Configuration file where the data is stored & transferred
* Template for the Azure pipelines

Pipeline

Stage A

Job 1

Step 1.1

Step 1.2

Job 2

Step 2.1

Step 2.2

Stage B

Job 1

Step 1.1

Step 1.2….

**To update the changes to the already existing repo other that master branch:**

1. Clone the repo to local

Git clone “repo path”

1. By default, you will be in **master** branch
2. Git branch -a -> this gives the list of branches present
3. Git branch branchname – to create a new branch
4. Switch to the required branch

Git checkout branchname

1. Now the cloned local repo will have all the files from the remote repo present in that branch
2. Make any changes locally
3. Git add filename
4. Git commit -m “commit msg”
5. Git push -u origin
6. Changes will get reflected in the remote repo

New branch will have all the contents present in the main/ master branch

**To delete a file both in local and remote repo**

1.Git rm filename

2.Git status

3. git commit -m “removed a file”

4. git push -u origin branchname

**To merge a branch into master branch:**

1. git checkout newbranch
2. git pull – to pull the lastest changes
3. git checkout master (checkout to the branch where you want the new changes to be merged)
4. git pull
5. git merge newbranch
6. git push

Now master branch will have all the files present in newbranch

**To clone the repo from the specific branch:**

git clone --single-branch --branch branchname“repo path”

**To get new changes in the repo first use git fetch and then use other commands:**

git branch -a = lists all the branches in the remote repo

git fetch = fetches all the new changes in the remote repo

git pull = pulls all the new changes to local

git clone = creates a new copy of the remote repo into the local

**When multiple developers are working on the same project:**

1. First clone the project from the repo
2. If u make any changes, add to the stashing area, commit the changes and push it to remote repo
3. If another developer makes any changes, **pull the latest change** into the local you are working on and continue to work and push.

**When multiple branches are present, don’t forget to checkout to the respective branches to see the project files in the local repo.**

**Pull Requests in azure:**

Pull requests - > is used to indicate other members about the code merge from one branch to another branch

1. There is active, completed, mine and abandoned pull requests
2. Abandoned -> pull requests rejected by the other team members
3. Can specify any no of commits
4. Can mention any no of team members to verify the pull request
5. Team members can approve, approve with suggestions and reject the pull requests
6. Once pull request is approved , it can be merged by keeping the other branch or deleting that
7. Merge can be done either by retaining all the commits or merging all commits to one single merge commit
8. Can see the history of the changes or the commits made in the history section.

Git bash commands:

cd ~ -> when we have navigated to the other directories use this command to head back to the home directory

cd .. -> brings one level down

cd .. -> brings further one level down

Git Reverting the changes:

File.txt

**Content:** hhhhhhhhhhhhhhhhhhh

Committed

**Changes made content:** zzzzzzzzzzzzzzzzzzz

Not committed

To revert back git command back:

**1.git diff file.txt:** This shows the old and current content of the file.

**2.git checkout file.txt:** This command reverts the data back to the file.

**Open the file and see the content:** hhhhhhhhhhhhhhhhh

**To unstage the files from staging area:** i.e. (to remove the file added to the staging area ):

1. Git rm --cached -r .
2. r = revert . = all files

**To avoid the files getting added to the staging area:**

Write the files which are avoided to be added to the staging area into the  **.gitignore file and save the file**

1. **Create .gitignore file .**
2. **Open .gitignire file**
3. **Add the files and save**

**Now use**

1. **git add .**
2. **git status : Only other files gets added to the staging area.**
3. **Git commit -m “msg”**

**To delete all the files inside the directory:**

**rm -i dirName/\***

**Rebase:**

Second way of combining the work between branches. It can be used to make a nicer linear set of commits.