Creating an automated build and deployment pipeline helps us in the early deduction of any issue and gives chance to fix it at the start.

**Continuous Integration:**

**Cxcxccxcxcc**

Integration Test

Code Quality

Package

Unit Test

Code Commit

Code Comit: Git/GitHub, Bit Bucket  
Unit Test : Junit, NUnit/xUnit  
Code Quality : SonarQube  
Package : MsBuild/Maven/Pycham are used to create package in different plateform  
Integration Test: Silenium

**Continuous Deployment:**

Deploy

Automated Test

Code Integration

**Continuous Delivery:**

Automatic Deploy on Next ENV.

Continuous Integration

Continuous Deploy

Testing Approval

**In Continuous Delivery, there is automated process to release build on different servers like dev/stage/UAT/Prod after testing approval.**

Create source code, checked in Github, Connect Azure DevOps with Github by select source file location at the time of pipeline creation.  
**Note:** Azure DevOps also has Azure Repo which is Git based and offers repository. Azure Devops can be integrated with Azure Repo/Github/Bit bucket or any private Git.  
When you select source location in pipeline, it automatically deduct docker file in source folder and suggest creating and pushing Docker image. You can also go with Starter Pipeline which create YML file and checked in Github. It automatically run Job to build code. You can see raw Log as well created in Build job.  
This build pipeline can be run when you commit code in GIt or it can be manually run as well.

**Git Command:  
Git init // Initialize the git repo, Create .git folder in source code folder  
Git Add \* // Add files in local .git folder as staging server  
Git Commit –m “commit message” // Commit in local Git  
Now add remote git repo to your origin  
git remote add origin** [**https://github.com/ptripathi-17/learningNotes**](https://github.com/ptripathi-17/learningNotes) **git remote –v //used to check mapped origin rep  
Git Push –u origin master // Push to Remote Repo  
git log  
git status  
Git pull /// pulling image from Git to local  
git commit –m “rename 4 pipeline” //renaming and commiting file in git  
  
Note:** Before commit code in Git, run git pull command to sync and merge your local with remote Git

**git init  
git add README.md  
git commit -m "first commit"  
git branch -M main  
git remote add origin** [**https://github.com/ptripathi-17/AzureDevOpsPipeline.git**](https://github.com/ptripathi-17/AzureDevOpsPipeline.git) **git push -u origin main**

**In YMAL File:** You can configure any branch. By default it is master. Agent is a server which is required to build the application. Agent comes from Agent pool. In a project setting, you can also create your own data center machine in Agent Pool.

**Stages**: Azure pipeline has multiple stages like Build,Dev,QA,Prod which denotes different environments.   
**Jobs**: Azure DevOps has multiple Jobs which are responsible to execute task. We can create multiple Jobs and if these jobs are not having dependencies and we have multiple agents available then it can run parallel so each job requires a different agent or server to run task. You can also create/Edit Jobs in YML file.  
**DependsOn**: When you want depend one Job to another Job and want to run in sequential order then specify dependsOn keyword in Job

Jobs:  
- job: Job1  
 steps:  
 - script: echo job1  
- job: Job2  
 **dependsOn: Job1** steps:  
- script: echo job2

Job

Step (Task)

Stage

**Multiple Stages:**

Stages:  
**-stage: Build**  
 Jobs:  
- job: Job1  
 steps:  
 - bash: echo job1  
-**stage: Devdeploy**  
 Jobs:  
- job: Job2  
 steps:  
 - bash: echo job2

**Variables**: Variables are used to pass dynamic values to different environments. You can create variable at pipeline level using Azure interface or you can create variable in Ymal file at Job/stage level.

Stages:  
-stage: Build  
**variables:  
Environment: BuildVar**  
 Jobs:  
- job: Job1  
 steps:  
 **- bash: $(environoment)DeployJob  
-**stage: Devdeploy  
**variables:  
Environment: DevVar**  
 Jobs:  
- job: Job2  
 steps:  
 **- bash: $(Environment)echo job2**

**Pre-defined Variables:** Azure DevOps has some pre-defined variables which has its own values and can be used to get specific details about that. For ex. Agent variable, Build Variable, System Variable etc. These variable could be used to debug the scripts if any problem in running .

Jobs:  
- job: Job2  
 variables:  
 Environment: DevVar  
 steps:  
 - bash: $(Environment)echo job2 **-bash: $(Build.BuildId)  
 -bash: $(Build.SourceDirectory)  
 -bash: $(System.DefaultWorkingDirectory)  
 -bash: $(Build.ArtifactStagingDirectory)**

**There are some pre-defined task which can be added by task list available in interface or to customize task you need to add YMAL. Below is ex of installing java,python,mavon**

-Job: firstjob  
 steps: **-bash: java -version  
-bash: node -–version  
-bash: python –-version  
-bash: mvn -version**

**Run task on Multiple Agents/Server:** We can add multiple OS in agent pool and run tasks on them.  
  
strategy:  
 matrix:  
 Linux:  
 operatingSystem : ‘ubuntu-latest’  
 mac:  
 operatingSystem : ‘macos-latest’

pool:  
 vmImage: $(operatingSystem)

**Artifacts:** Build need to be published artifacts when we want to deploy same build on multiple stages/servers. There is pre-defined task available to publish artifacts.

**Deployment Task:** Deployment task is special type of task and available to deploy. You need to mention deployment keyword in place of job in Yaml file.

Stages:  
-stage: Dev  
jobs:  
**- deployment: devDeploy  
 environment: Dev** //environment will create two different labels for diff. deploy stage  
 **strategy:  
 runOnce:  
 deploy:** steps:  
 -script: echo deploy to dev  
-stage:QA  
 jobs: **- deployment: QADeploy  
 environment: QA  
 strategy:  
 runOnce:  
 deploy:** steps:  
 -script: echo deploy to QA

**Note:** You can set approval in any environment through interface to proceed pipeline.

**Build Docker Image and Push to Docker Hub:**

To push docker image to Docker hub, you should make connection between Azure DevOps and Docker Hub. Go to project setting and create service connection for Docker Hub. Docker file should also be present in source code folder.   
Now, go to pipeline section and create new pipeline, Select your Github directory as source directory and inn configure step, Devops suggest to select Docker configuration. You select and create pipeline. Edit Yaml file and new task build and push and provide path of service connection of Docker Hub. This pipeline will perform all the task what is written in docker file and build then push to Hub.

# **Integrate Terraform in Azure DevOps**

When you want to provision servers automatically through pipeline the you need to integrate Terraform with Azure DevOps.

For terraform integration, you need to install plugins from below link then you would be able to integrate terraform commands like Init/Plan/apply through Azure Devops using Task in YAML file.

* Terraform 1 (<https://marketplace.visualstudio.com/items?itemName=ms-devlabs.custom-terraform-tasks>)
* Terraform 2 (<https://marketplace.visualstudio.com/items?itemName=charleszipp.azure-pipelines-tasks-terraform>)

**Create Kubernetes Cluster(AKS) using terraform using Azure DevOps  
Note: When you create any DevOps pipeline with Azure, you need to create service connection of that particular resource/service. For Terraform pipeline this resource would be Azure Resource Manager(ARM) for Service Connection.**

* Write code for resources manager, Kubernetes in main.tf file in VS code.
* Create variable.tf file and mention all variables like client ID, secrete key, resource group, location, kubernetes name, nodes etc.
* Call variable in main.tf file from variable.tf file.
* Pass variables of clientid, secret for service principle in main.tf
* Write code in main.tf file to backend arm to store in blob.
* Install Azure cli on local and run below command to get subscription Id which will be used to create client ID and secrets
* **Az login** ---it will connect with Azure and return subs ID
* Create service account from azure CLI by passing subscription id.
* **az ad sp create-for-rbac --role="Contributor" --scopes="/subscriptions/<<azure\_subscription\_id>>"**
* After creating service account, it will return appid as client id and password as client secrete.
* Now we also need to create SSH key. Run below in azure cli
* **ssh-keygen -m PEM -t rsa -b 4096 # PEM - Privacy Enhanced Mail - Certificate Format RSA- Encryption Algorithm**
* Above command will create SSH key and will save key in .**pub** file in source code folder of terraform.
* Highly recommended that don’t place .**pub** file in same folder and move to other location to prevent from commit in GitHub

We are done with Terraform file. Now its time to connect this file with Azure DevOps pipeline.

* First of all, we need to make connection with Azure from DevOps. So go to project setting and create Service connection. In service connection, choose Azure Resource Manager(ARM) as service or connection type.
* To use terraform file and commands in devops pipeline, we need to install two plugins mentioned above.
* Go to pipeline, Select Source code github location.Create and edit ymal file.
* Add in built task : **Terraform CLI**
* When you add this task, you will see terraform commands in list option. You should start with init.
* Select systm.workingdirectory as configuration directory and append .tf file path.
* Select Azurerm as backend option.
* Pass client id, secrete key, ssh key in command option.
* All above values will be set through interface in terraform CLI inbuilt task.
* Create/pass resource group and location, storage name, container name for backend.
* Client id, screates will not be passed directly in YMAL file. You need to click on variables and then add all variables here. Pipeline will pickup variable values form here.
* SSH file is saved in file on our machine so add this .pub file in library section.
* Now go to again ymal file, add in build task :Download secure File” and add name of file.
* Give name of this task file and file in variable of terraform cli task.
* This task should be added before terraform cli task.
* Now save the pipeline and run.
* Above pipeline has INIT command but to create resources we need to add apply command.
* Edit YAML file and copy entire previous task and paste but replace command name from Init to apply.

**Azure CLI:** You can download its installer and install on your local machine. It is used to connect with Azure from local.  
Check installed version  
az –version

Now IAAC pipeline for Kubernetes has been created and triggered. So, you can connect now with AKS through Azure CLI. Below command to connect with K8S.

* Az login  
  ag aks get-credentials –name kubernetesclusterName –resource-group resourcegroupName  
  kubectlt get svc  
  kubectlt get pods  
    
    
  **Create Microservice Deployment Pipeline in Kubernetes through Azure DevOps Pipeline - Pending**Steps To Create PipeLine
* Create Service connection for Kubernetes with Azure to DevOps from Project Setting.
* Go to Pipene, click new pipeline
* Select source GitHub Repo.
* Rename pipeline and save and edit.
* Now we need 4 task to add in this pipeline.
* Build Docker Image
* Get and publish K8S files --Need to create deployment.yaml and place in source code/github
* Deploy to K8S cluster

**Note:** In the Production scenario, Separate github for the Provisioning cluster, Separate Github for each microservice.

**Create AWS Kubernetes through Terraform using Azure DevOps Pipeline**

* To call AWS from Azure DevOps, you need to install AWS plugin on your machine like you did for terraform.  
  <https://marketplace.visualstudio.com/items?itemName=AmazonWebServices.aws-vsts-tools>
* You also need AWS IAM secret key and code to connect from Azure.
* Go to Azure Devops and create service connection for AWS from project setting section.
* Select **AWS for Terraform** as destination of Service connection since we are provision EKS.
* Pass Secrete id and keys
* Create new pipeline, rename.
* Select Github directory where main.tf is placed and choose starter pipeline.
* Add **Terraform** as in built task, **AWS** as provider, **Init** as command, **append path of main.tf file with $(System.DefaultWorkingDirectory)** in Configuration directory, Choose **Service connection** and now create task.
* Now add task for terraform apply command
* Add in-built task **Terraform** and like above select all fields and in command choose **validate and apply** command

**AWS CLI :** It helps to connect with AWS from our local machine. Download and Install through MSI Installer. To know version of installed CLI use below command.  
**aws --version**

**Setup Service connection for EKS using Azure DevOps Pipeline**

* Select service connection from Azure DevOps project setting.
* Select kubernetes as destination service in configuration.
* Select Service account option in place of Azure subscrion in next step of Kubernetes Service connection.
* Here you need to fill connection name, server URL and secrets.
* To get connection url , you need to configure AWS CLI by passing access key ID and secret access key
* Run AWS login
* **Aws configure**
* Pass key and secrets, region
* Now we need to set the credentials up to connect Kuberenetes cluster from kubectl. Kube file would be updated to use Kubernetes cluster.
* **Aws eks –region us-east-1 update-kubeconfig –name=clusterName**
* Now kubectl is configured for AWS EKS and able to run EKS command.
* Now running command through Kubectl, you can get service url of cluster
* To get service url of cluster run below command.
* **Kubectl cluster-info**
* Now you need to get secrets of cluster to get it with service account.
* Run below command
* Kubectl get serviceaccounts
* Kubectl get serviceaccounts default –o yaml
* Kubectl get secret secreteName –o yaml
* It will return long details which ypu need to paste in Secrete field of service connection and now click to create Service Account for AWS EKS.

**Deploy Micro service in EKS using Azure DevOps Pipeline**

Stage 1 (Build)

Publish Artifacts to Build.ArtifactStagingDirectory

Copy Deployment.YAML from System.DefaultWorkingDiretory to Build.Artifactstaging Directory

Build Docker

Push Docker image to Repo

Stage 2 (Deploy)

Select Task Deploy to Kubernetes

Kubectl apply – deployment.yaml

Download Pipeline Artifacts to System.ArtifactsDirectory

* Create new pipeline to build and deploy pipeline for Microservice in EKS.
* Select Github repository micro service location.
* Select Build Docker Pipeline