AzureDevops-Pipeline

Status	Not Started

we will see what it is and

what you can do with azure devops platform how to use it for your software development projects and generally how

is this relevant for you as an engineer we will see different features and use cases of azure devops how it all works

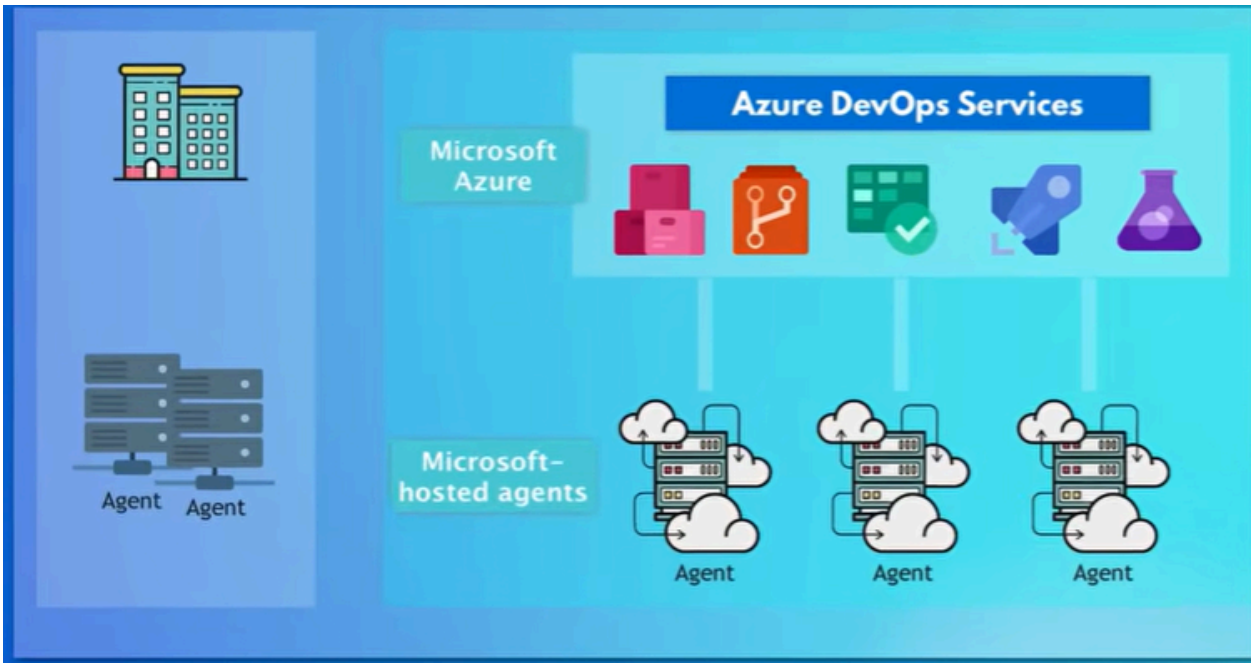
and how you can implement the whole software development and deployment lifecycle with it

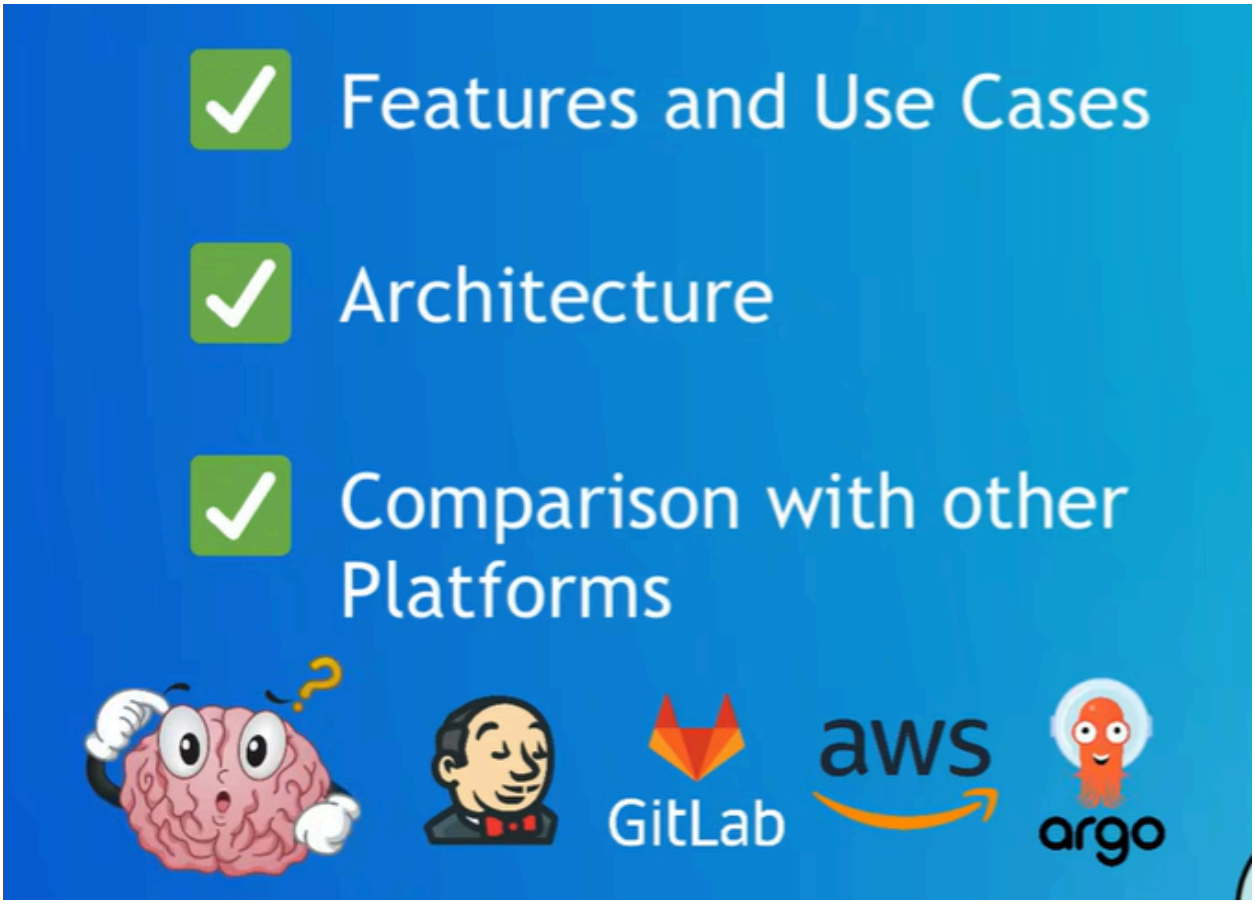
we will also review

the azure devops architecture and how it works in the background and finally we

will also compare it with alternative tools and talk a bit about which of these tools you need to learn for your

career with so many alternative options so let's get started


Azure DevOps Services
Microsoft Azure
Agent
Agent
Microsoft-hosted agents
Agent
Agent
Agent


Features and Use Cases
Architecture
Comparison with other Platforms
GitLab
aws
argo

What is Azure DevOps?

so first of all what is azure devops as the name

suggests it is a devops platform it is a software as a service offering which was

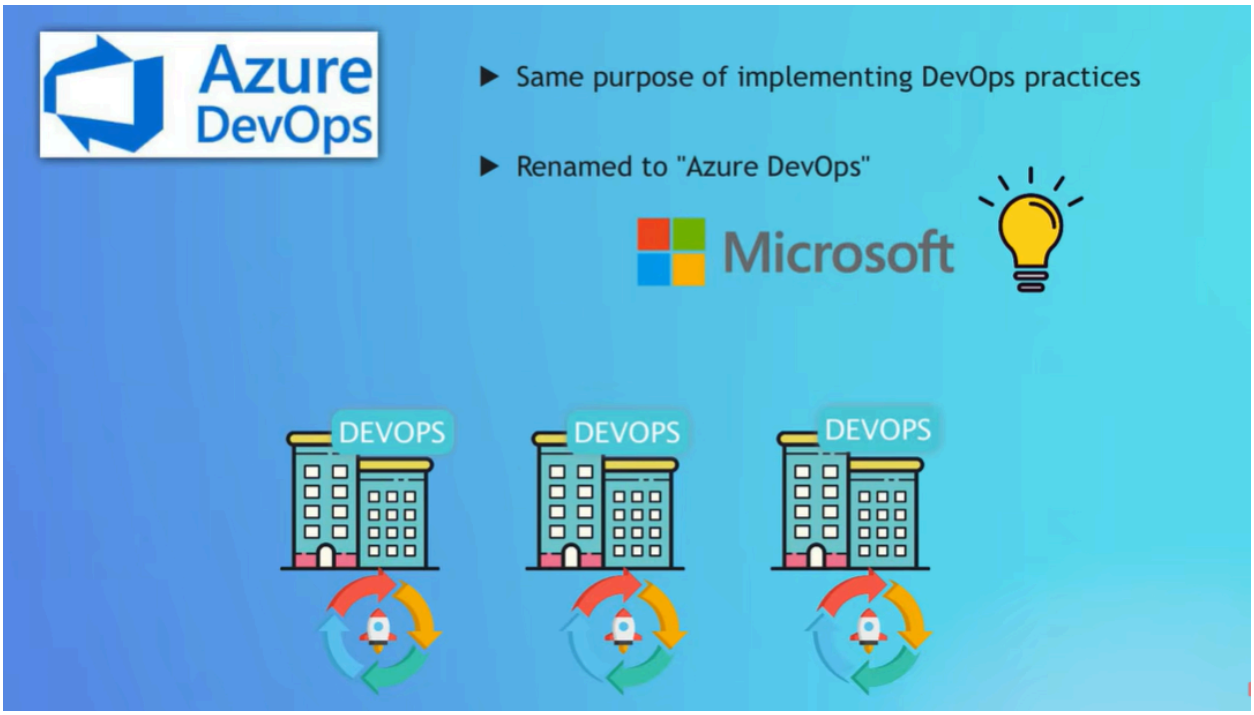
created to basically be a one-stop shop for implementing all your devops processes for your

project and it had many names before it became known as azure devops it was called team foundation server then

visual studio team services so it had a bunch of other names before azure devops because it extended from existing services and tools so it kind of got names of these tools and

even though it was for the same purpose of implementing the devops processes at that time devops

was still practiced by a smaller group of projects



so that's why the name devops never actually came up in the name of the platform itself however since devops became more mainstream and as it is now used in lots of projects worldwide it was actually renamed to azure devops which is pretty smart because devops is already a popular term everybody knows

what it is so just by the name you know that it's a platform for devops

but it still sounds too general because what is a devops platform devops is many things right so what is a platform for devops and to answer that question

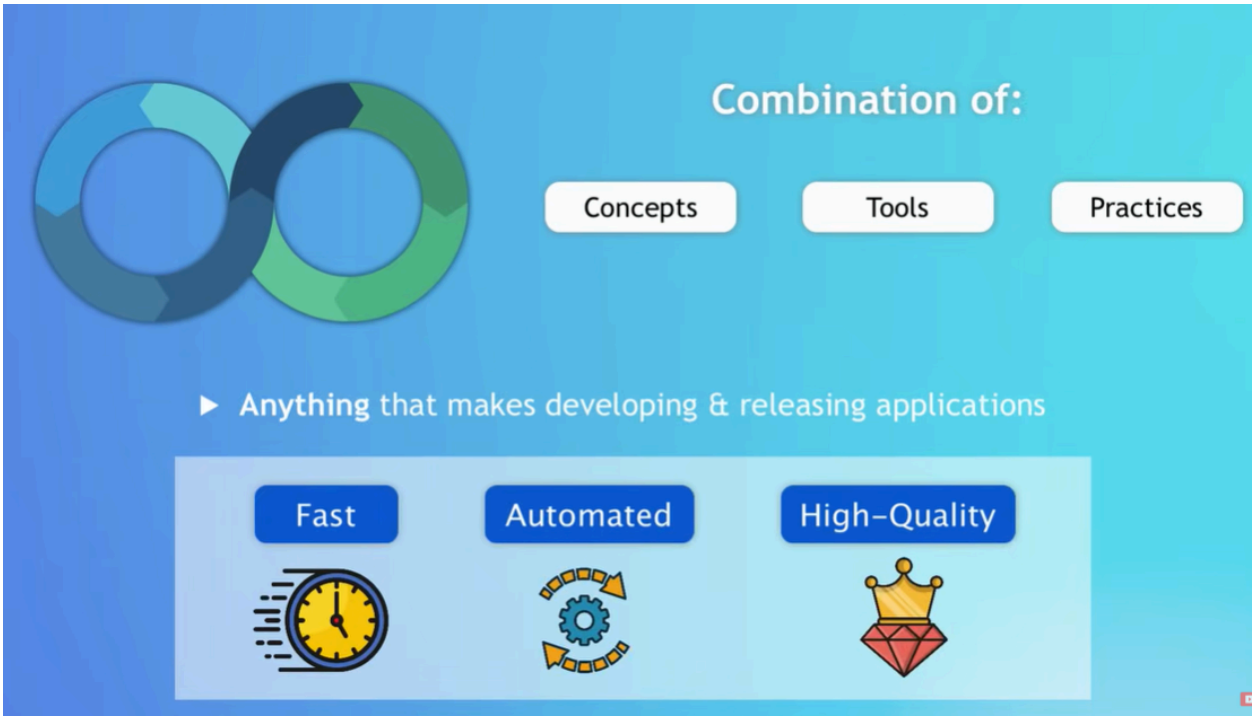
let's look at what azure devops is exactly and

how it helps in implementing devops processes well

devops as i said is many things it is a combination of concepts and tools and

basically anything that makes developing and releasing applications fast in an

automated way and with high quality possible so a project needs to implement devops practices in order to achieve



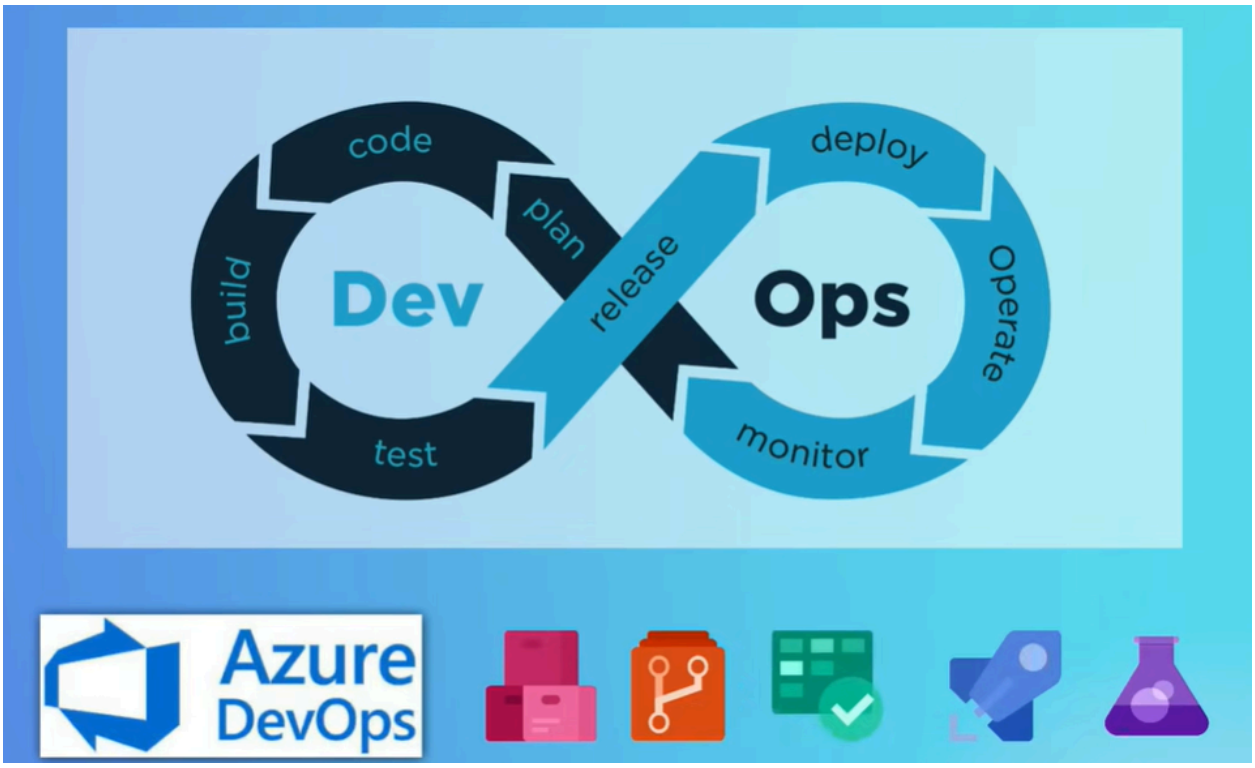
an efficient workflow it's to make

the software development lifecycle as efficient as possible by fully or mostly

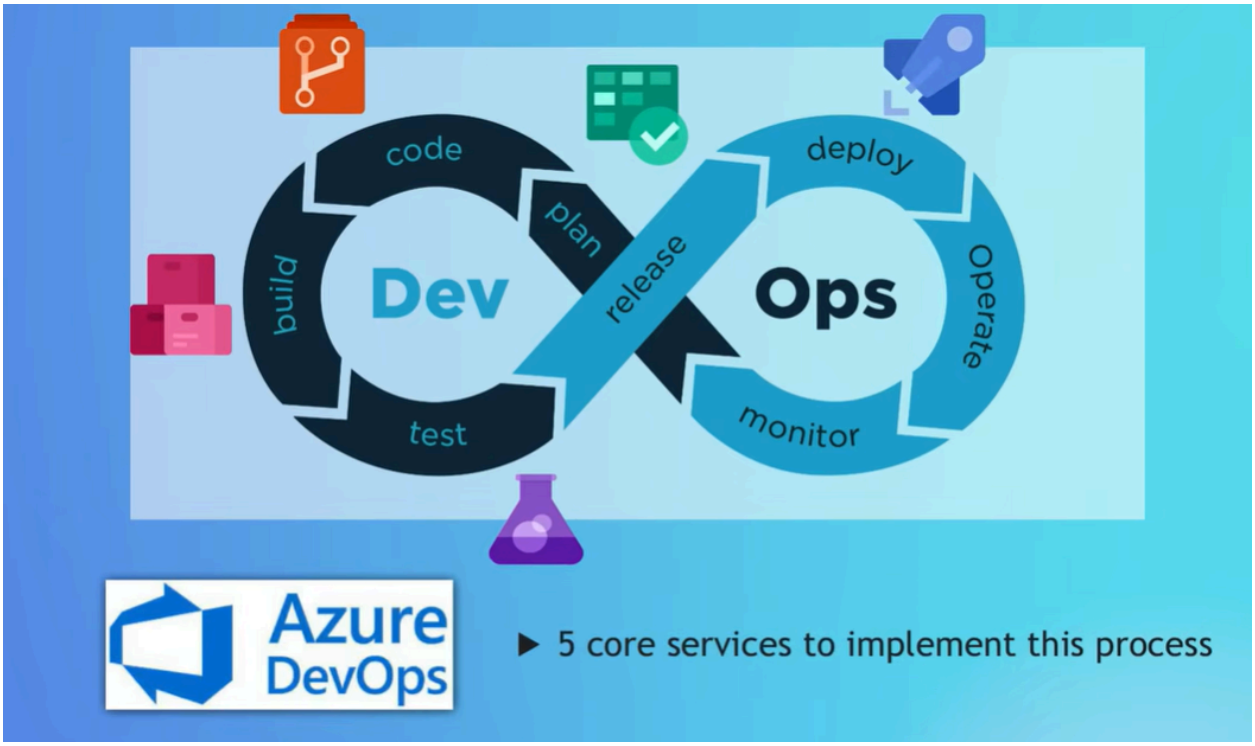
automating it and azure devops is basically a technological implementation

of that devops process which covers the whole software development lifecycle and

it has features for each part of this lifecycle



so let's go through the steps or parts of that software development lifecycle and see how the various azure devops features map to those parts



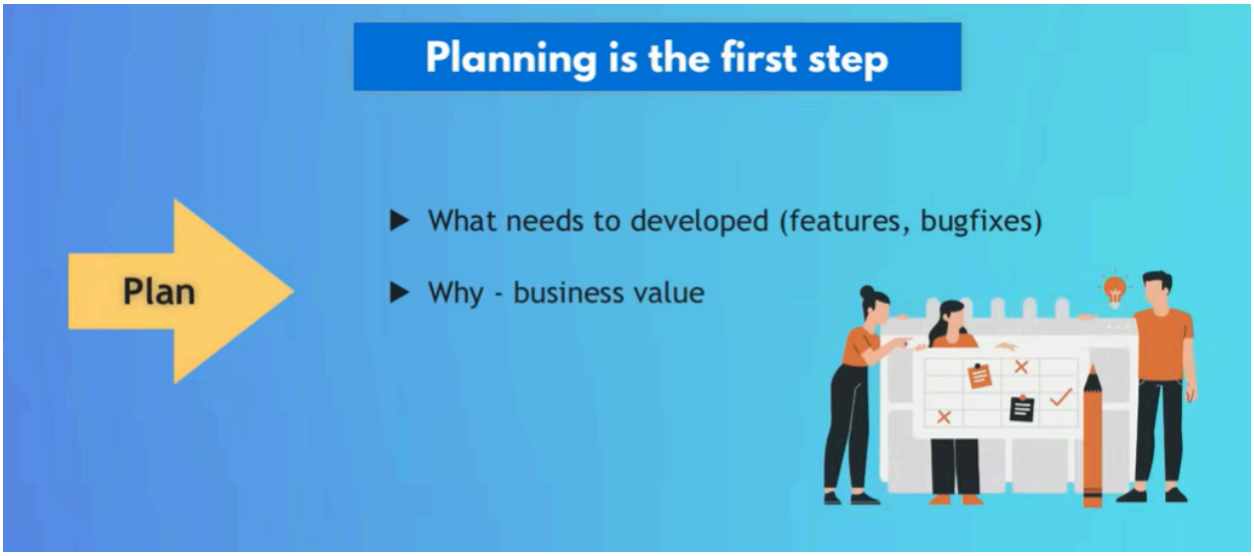
Azure Boards

now what does a software development lifecycle include it's not just developing the application

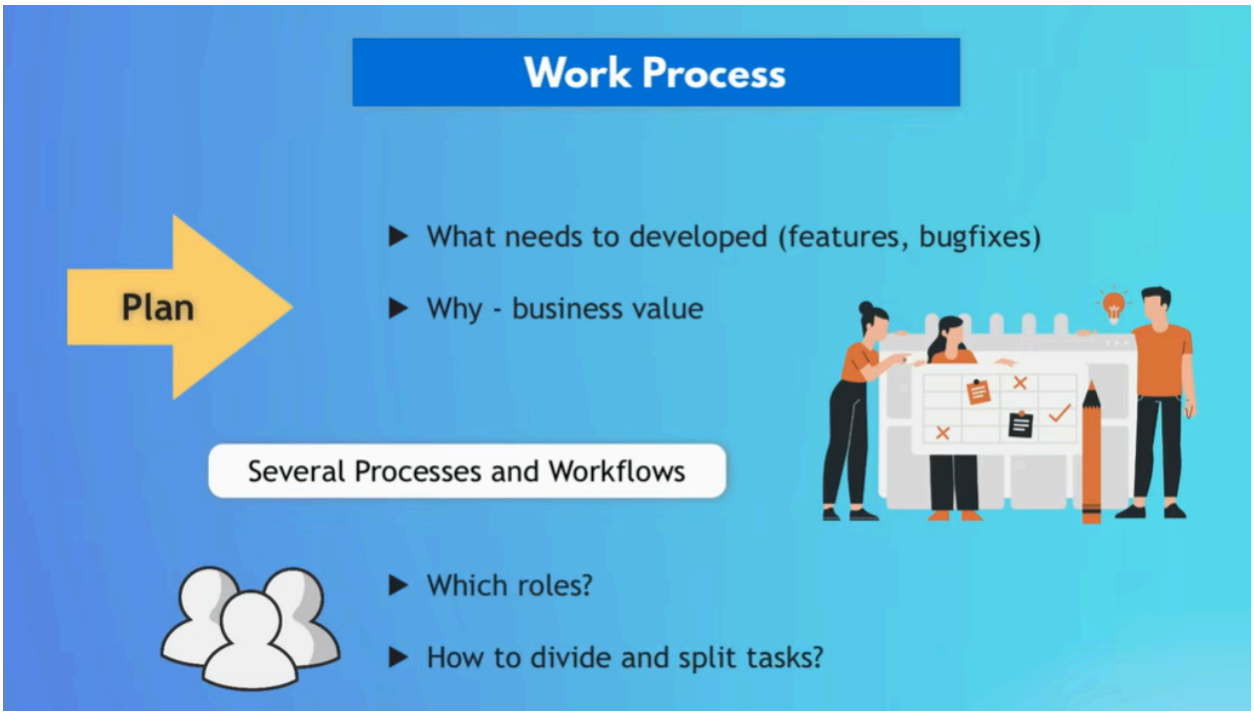
or writing the code it actually starts before you have written a single line of code

because before coding any feature or an improvement needs to be planned first

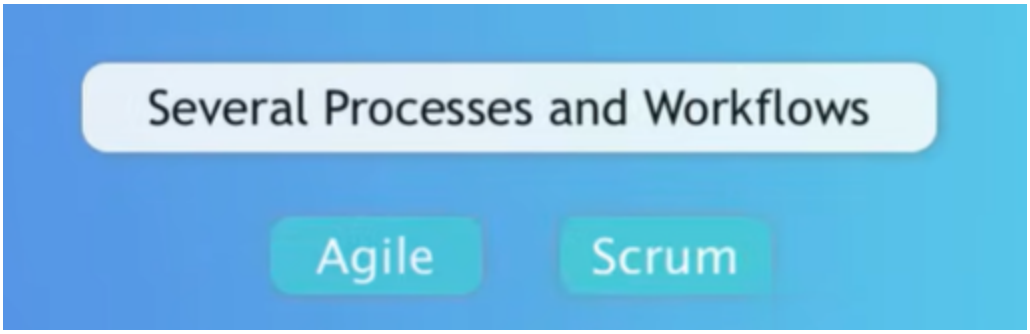
right mostly by a product manager so the first step is to define



what we are developing and why are we developing it or in other words what's the business value behind it and there are several ways of defining the workflow of how the project team will work on the application

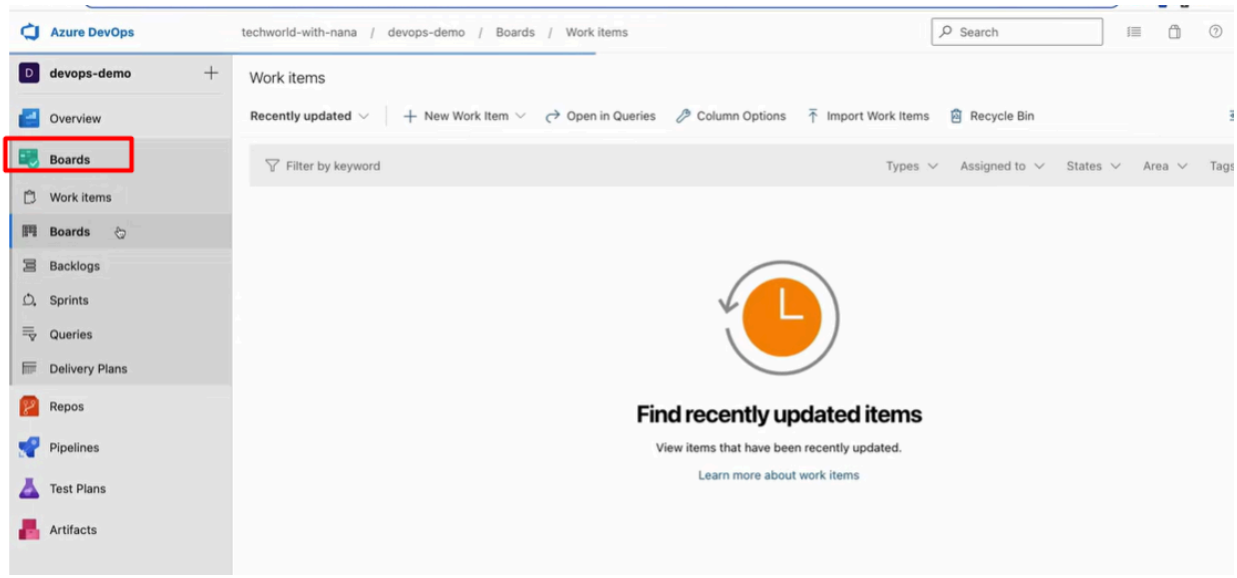


which roles they have within the team how they will divide and split the tasks etc



two of the most popular workflows are agile and scrum many modern projects use one of these approaches for their development so once you have created a project in azure devops for your

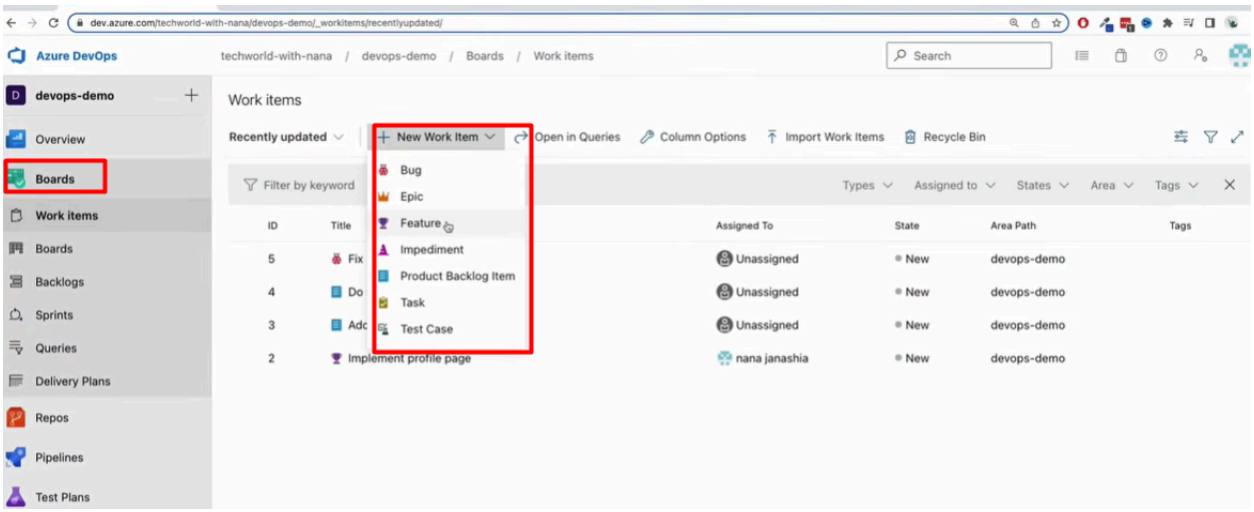
application



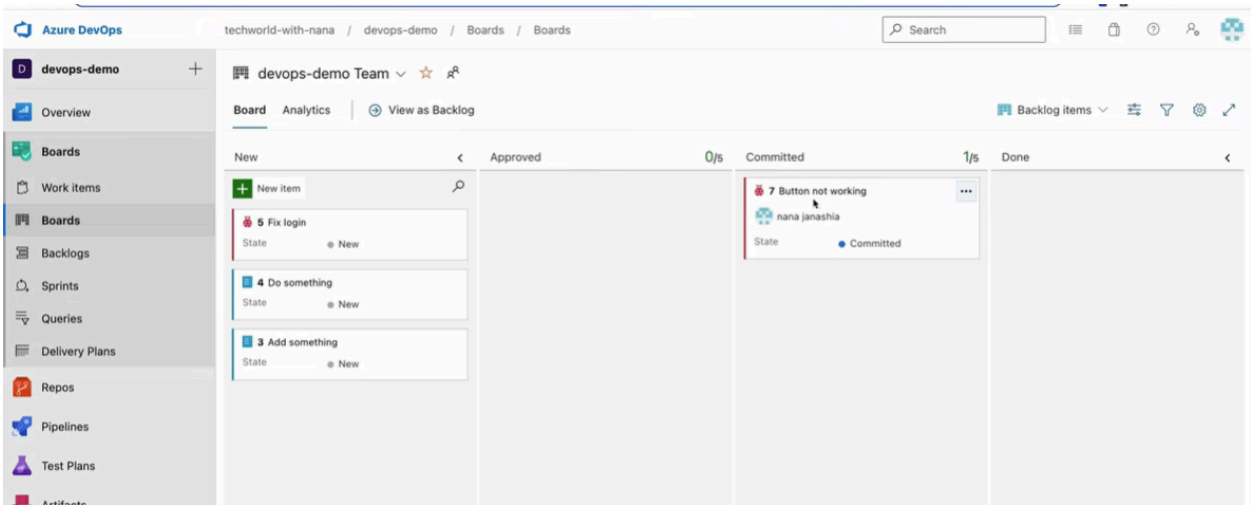
the first feature you will probably use here will be azure boards and depending on which workflow you use for your projects in your company generally like agile scrum or even some basic workflow you can choose the corresponding board for this project because

as i said azure devops or any other similar platform is basically just a tool that gives you various features to implement whatever workflows you have in your company

so on azure boards you can create tickets or tasks for features improvements or fixes for your applications

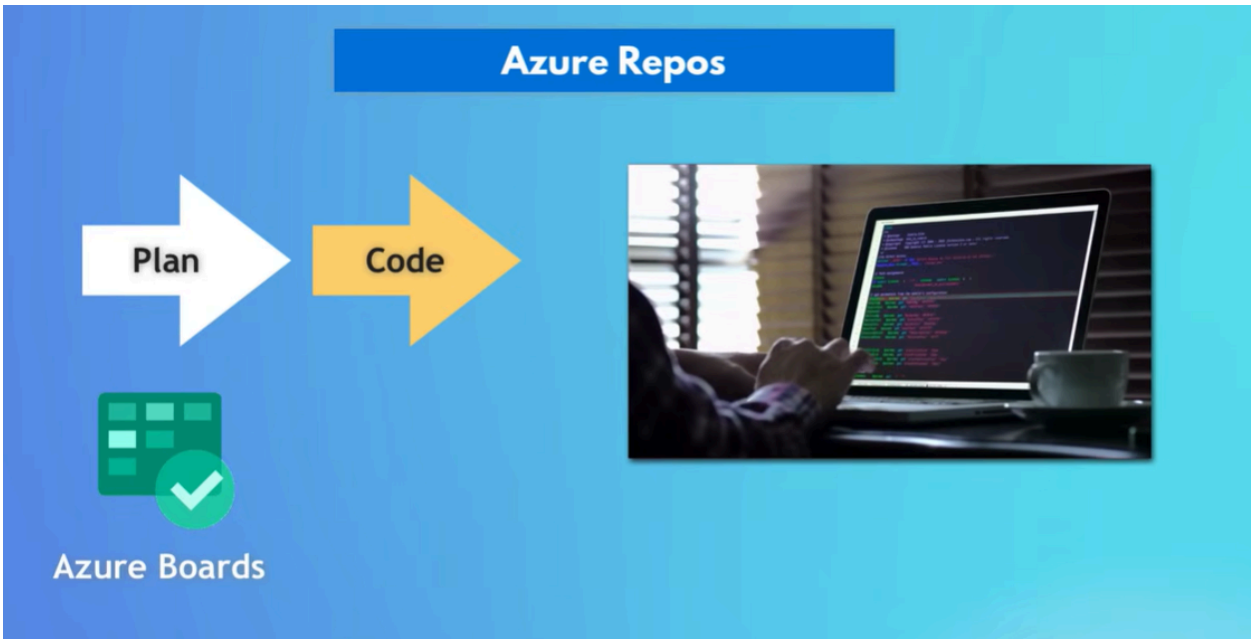


as part of that agile or scrum processes you can assign it to people to work on and you can also track progress of those features or improvements while they are being developed now developers need to take that planned task and actually develop it right in that process they may have questions about the task

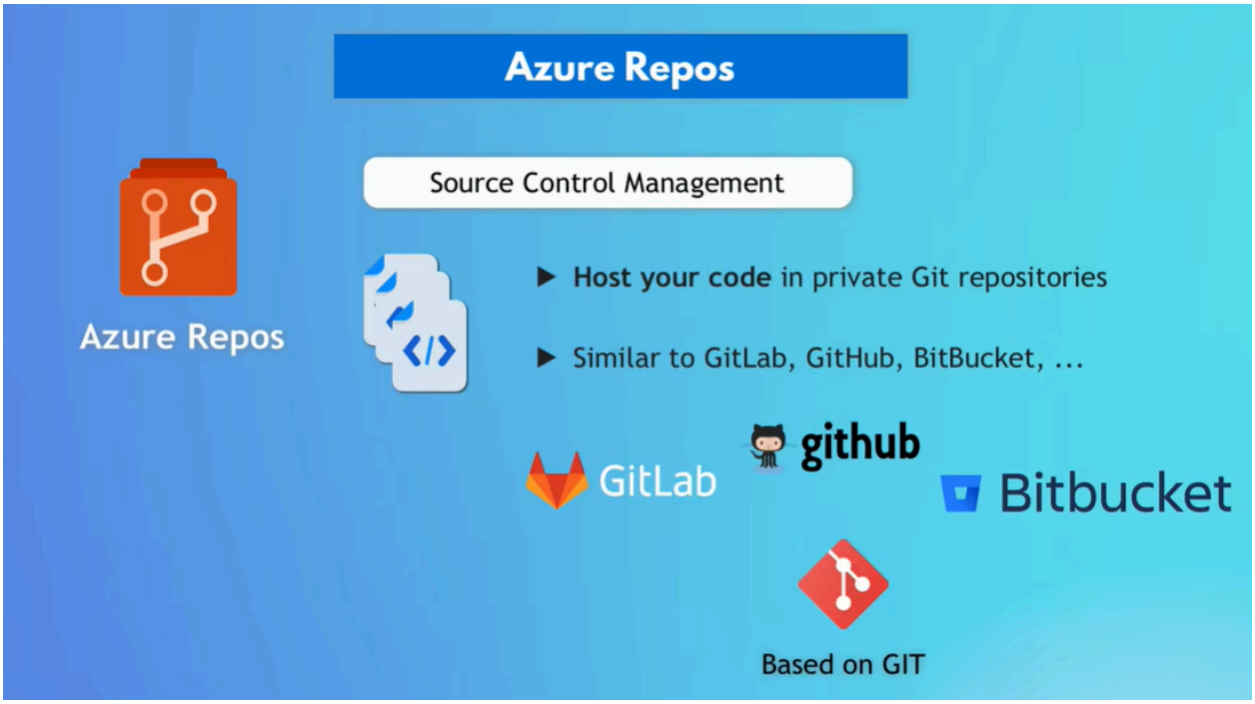


so azure boards can be used for communicating between the developers testers product owner etc within the task description plus it can be used to have an overview and transparency over the status of the feature as it is being developed who is working on it what stage it is in what's its progress status has it been deployed already etc

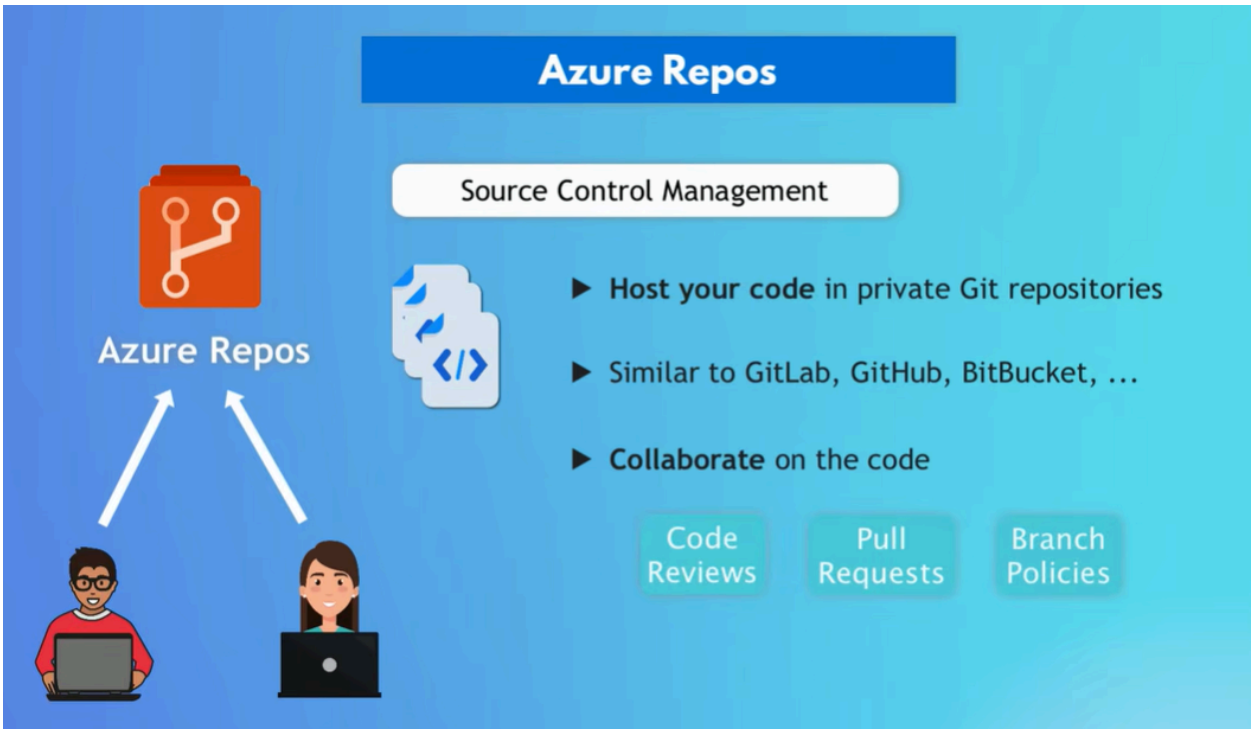
Azure Repos



now the application code that developers are creating is also part of that life cycle actually the main part of it an azure repository feature is what you can use in order to host that code now of course you know the popular code repositories like github gitlab bitbucket etc

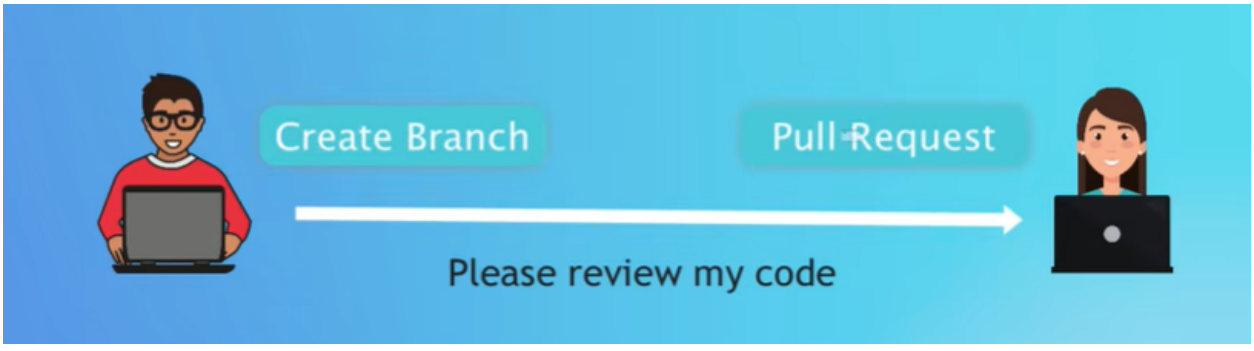


which are all based on git and that's basically an alternative to those platforms so editor repository also supports git which is the most popular version control tool so developers can host their code in azure repository and push their changes to it however the code repositories have actually evolved and became much more feature reached and

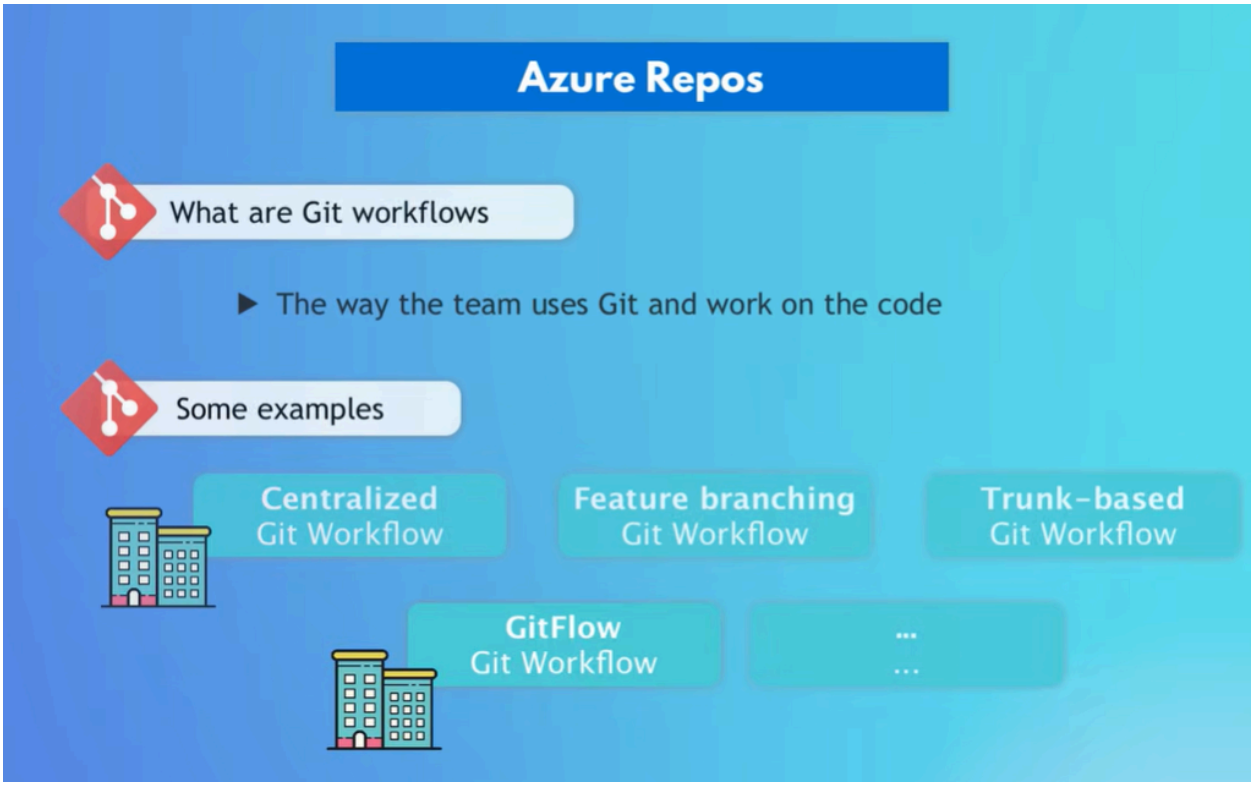


just hosting code is one of the many features that they provide so azure repository also it's not just for hosting the code as part of the devops life cycle developers collaborate and work together to develop high quality code

so in the repository as part of the git workflow you have features such as pull requests branches various collaboration features and so on

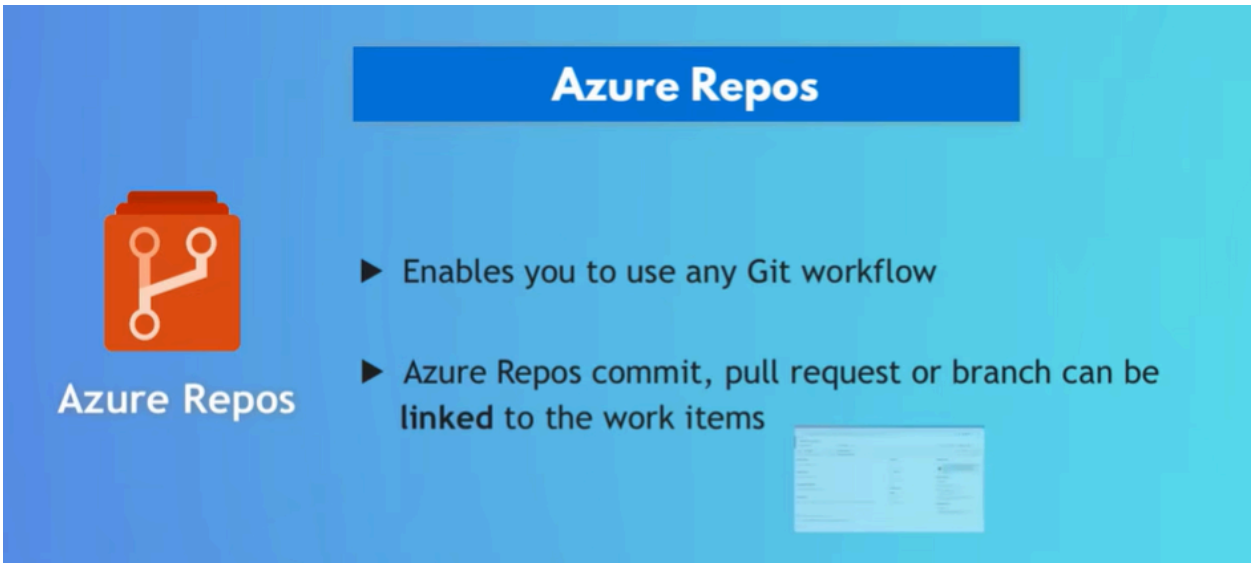


so when developer starts a task they create a temporary branch when done they create a pull request and other developers can review and comment on the pull request they can communicate and collaborate until the pull request is good enough to be merged into the main branch now this is what's called a git workflow



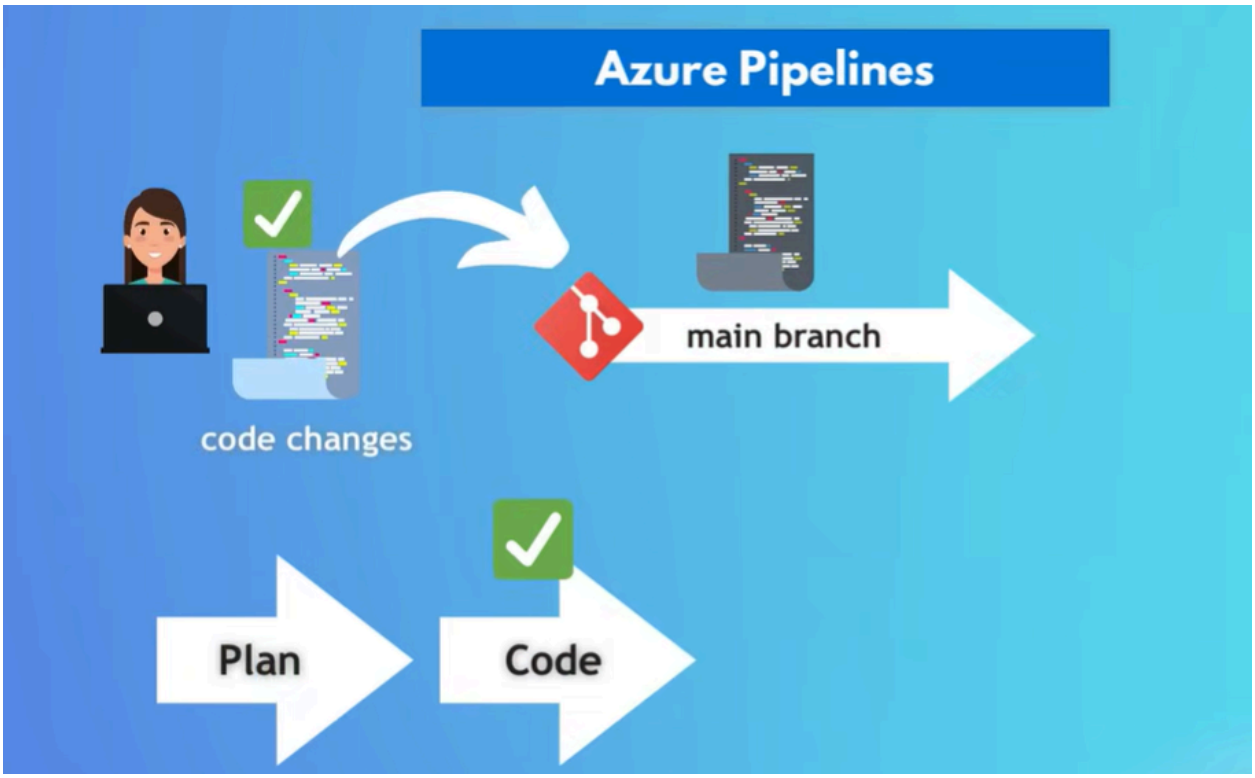
so basically how your team decides to work on the code and to make sure that the quality of code is very good and there are many git workflows or the way teams use git and all its features and different companies may use different approaches or different workflows

but the key point here is that with azure repository you have the tool that enables you to implement whatever git workflow you choose to work with plus note that the repository and branches and the pull requests are all linked back to the feature task so you see the activity and



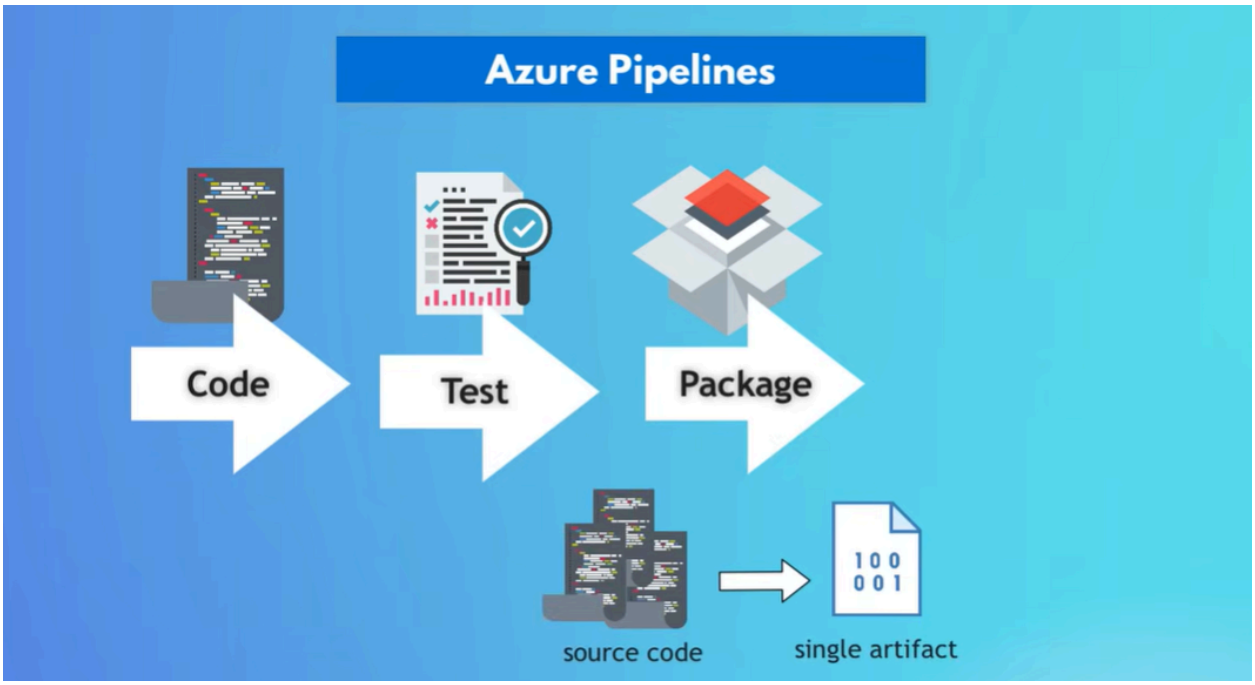
status of development there as well

Azure Pipelines (Build Pipeline - CI)

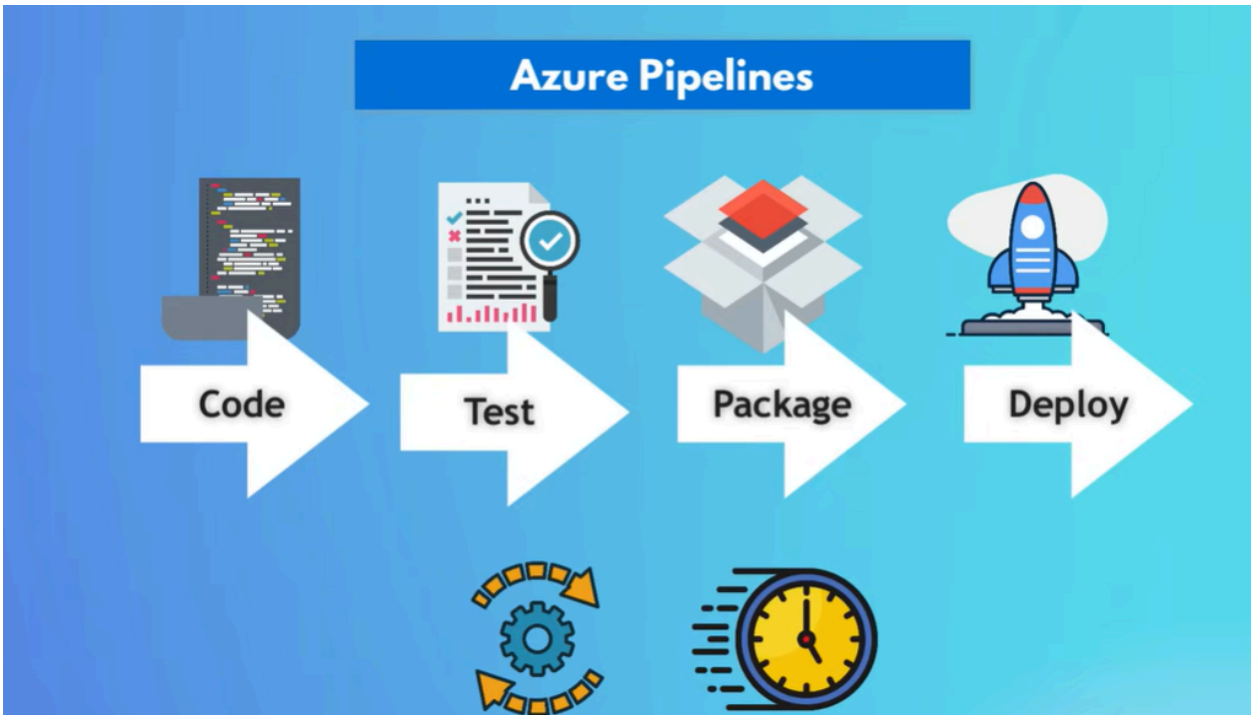


now once the feature is developed and pull request is approved and merged into the main branch it needs to be released right that's the reason we're developing the feature in the first place so we can

release it and the end users can use it so in order to release our code changes

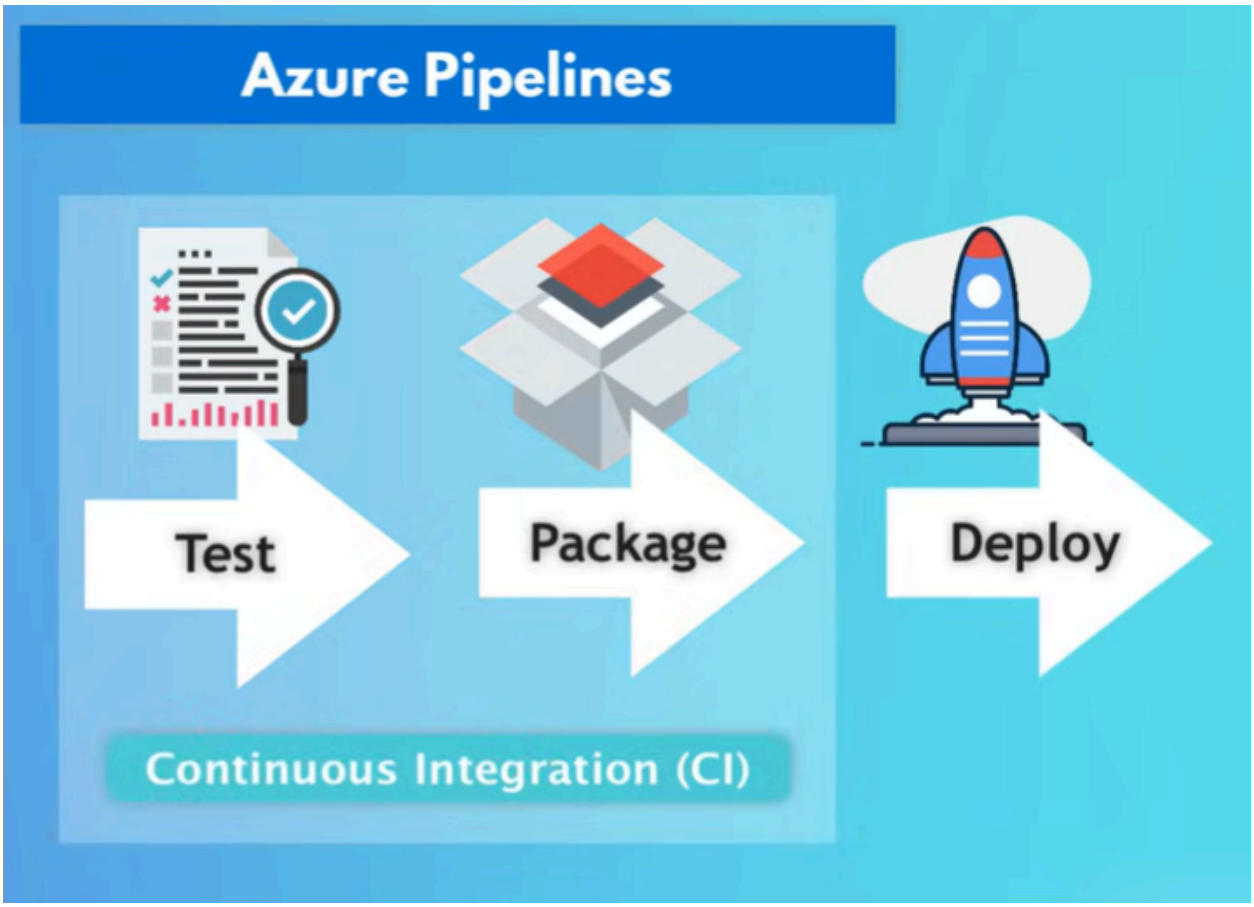


we first need to test it and package it into an artifact which is a deliverable that we can then deploy on the end environment and again devops is all about automating things and workflows



so that it's fast and efficient

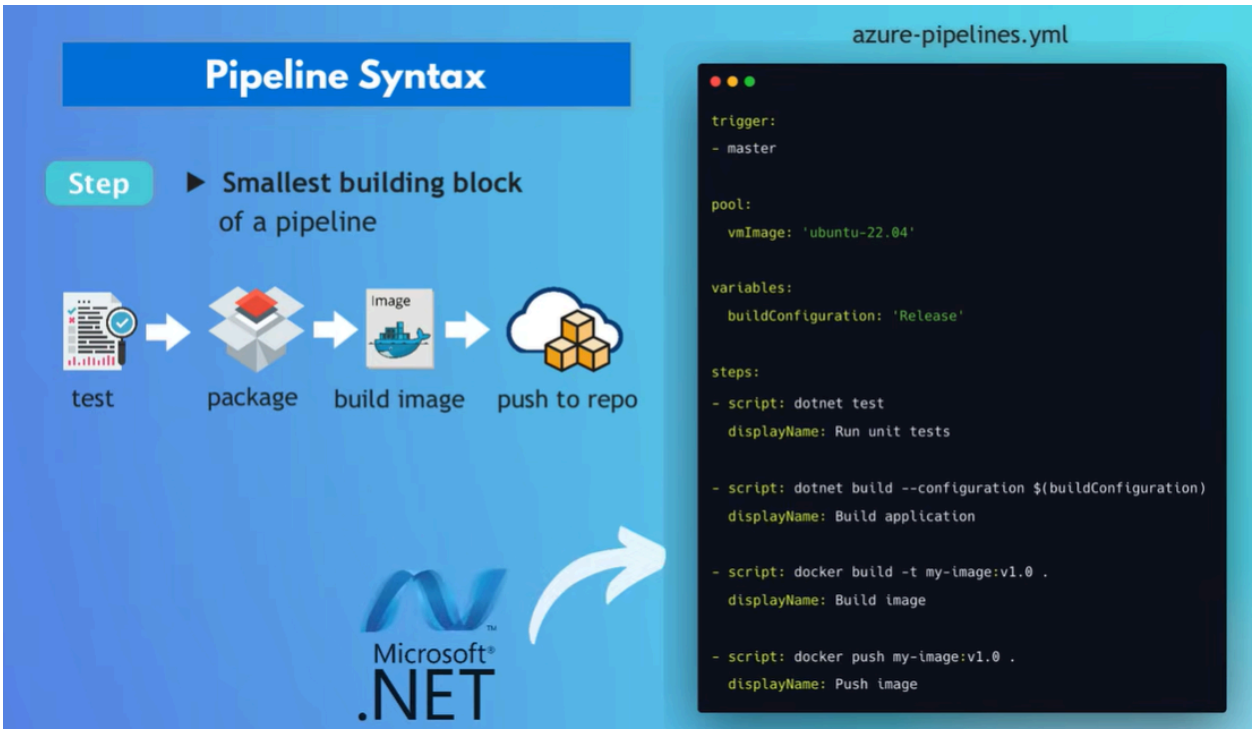
so this process of testing and building the application is done by an automated ci or continuous integration process and for building the ci pipeline azure has



what's called azure pipelines section pipelines can be written in yemo which means you can have your pipeline script

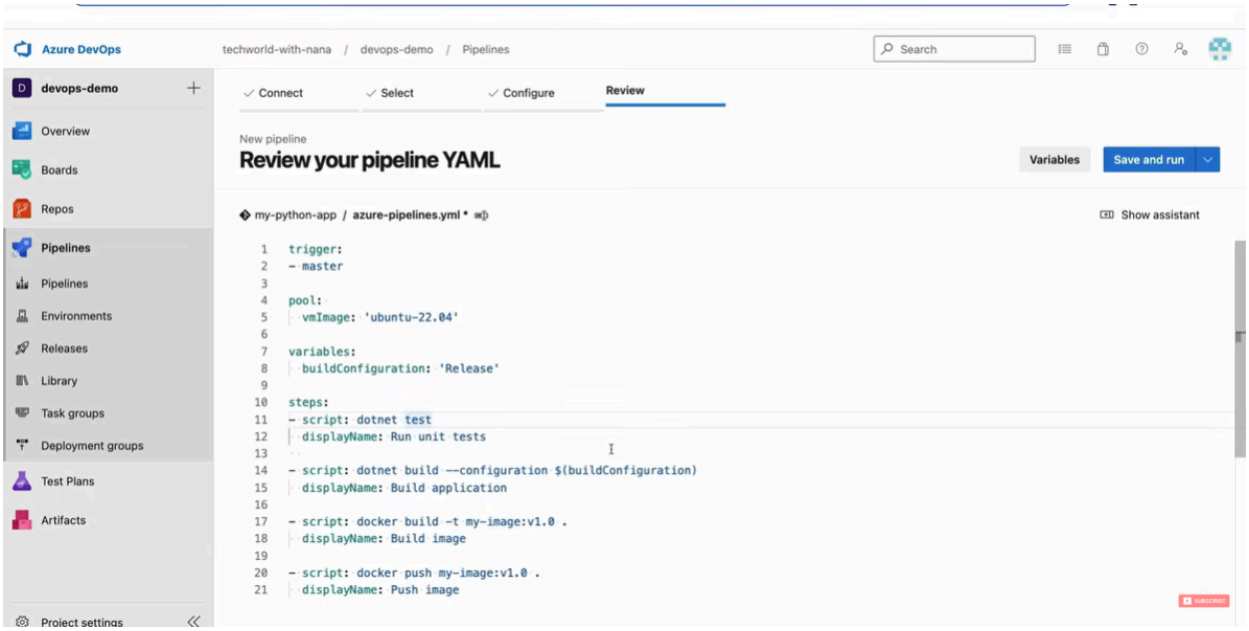
as part of your code the main building blocks of azure build pipelines are steps for example if we

want to test and package the application we may have steps to run tests package application build an image push that image to an image repository so that later it can be deployed so each step will execute a certain command to run the test to package the application build a docker image and



so on and this is an example pipeline for building a.net application

which is a common type of application built on azure devops platform since asp.net is also part of the microsoft technology stack in the first two steps in this pipeline we test and build the application with net commands and then build the image with a docker command and as you see we execute commands in script attribute however we have another option for executing step commands instead of scripting it directly in yemo ourselves so in addition to writing the scripts yourself you can use what's called a task instead



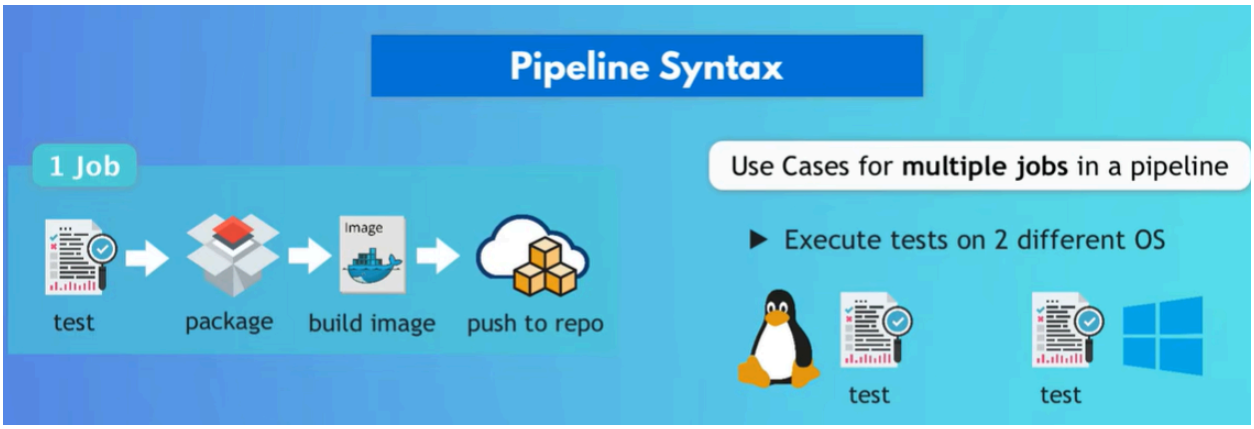
so what is the task and how does it work in azure devops you have loads of tasks already available to choose from for different use cases and they can be selected and configured using a ui you can select the tasks directly from the list of available tasks on the right side of the pipeline's yaml editor and configure any needed parameters for the task and when you have the task configured basically you fill in and set all the parameters you can add the task back to your yaml pipeline it will be automatically converted and edit as yaml code and of course you can adjust or add any additional configuration to it

so here for example if you look for asp.net task you will find one for executing various.net commands like test build etc and once configured you can add the test task in place of the script you can do the same for docker you can search for docker tasks and find the one with build and push commands and add that task to your yaml pipeline and this obviously can be convenient in many use cases because you don't have to know the commands exactly you don't even have to know or memorize the exact syntax for the pipeline and you also don't have to script the steps from scratch

so this basically gives you a simple um an easier high level approach to adding steps to your pipeline now this is a very simple scenario of a pipeline with one

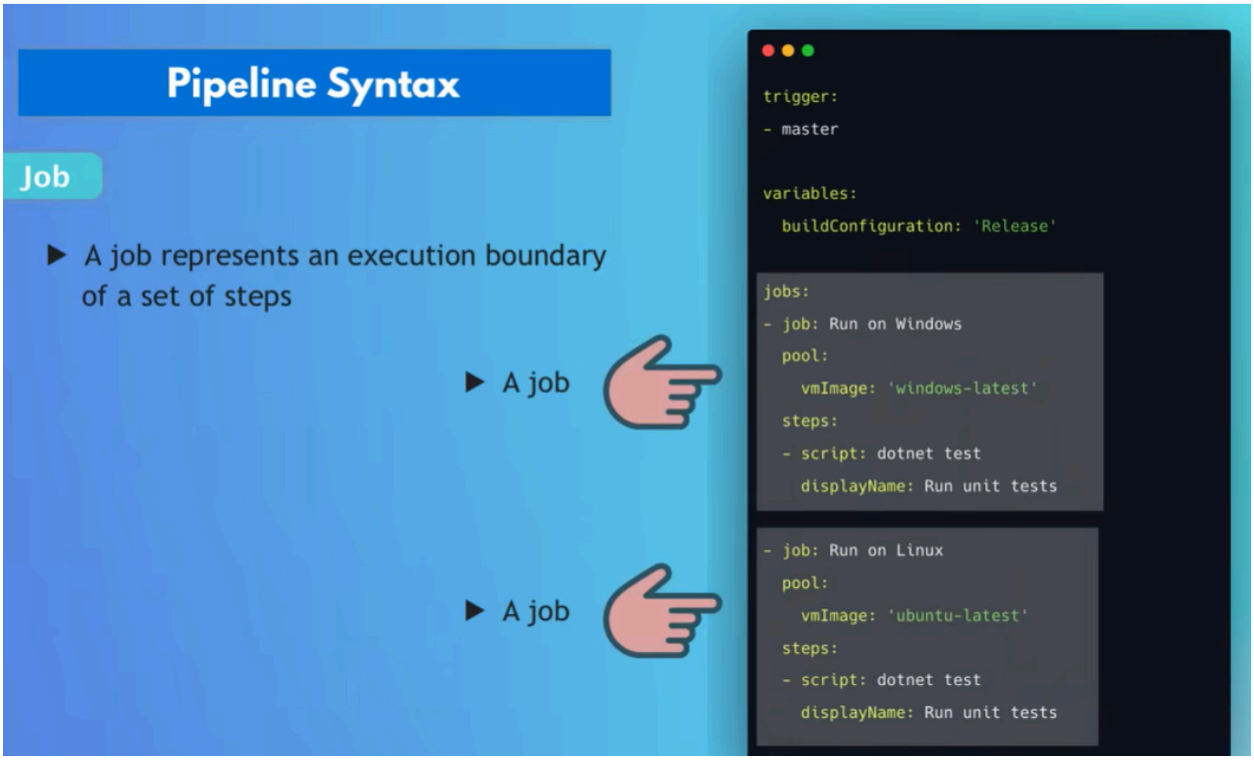
single job that has all the steps

but in practice we often have more complex scenarios where we would need multiple jobs in a single pipeline let's say we want to execute the test step on



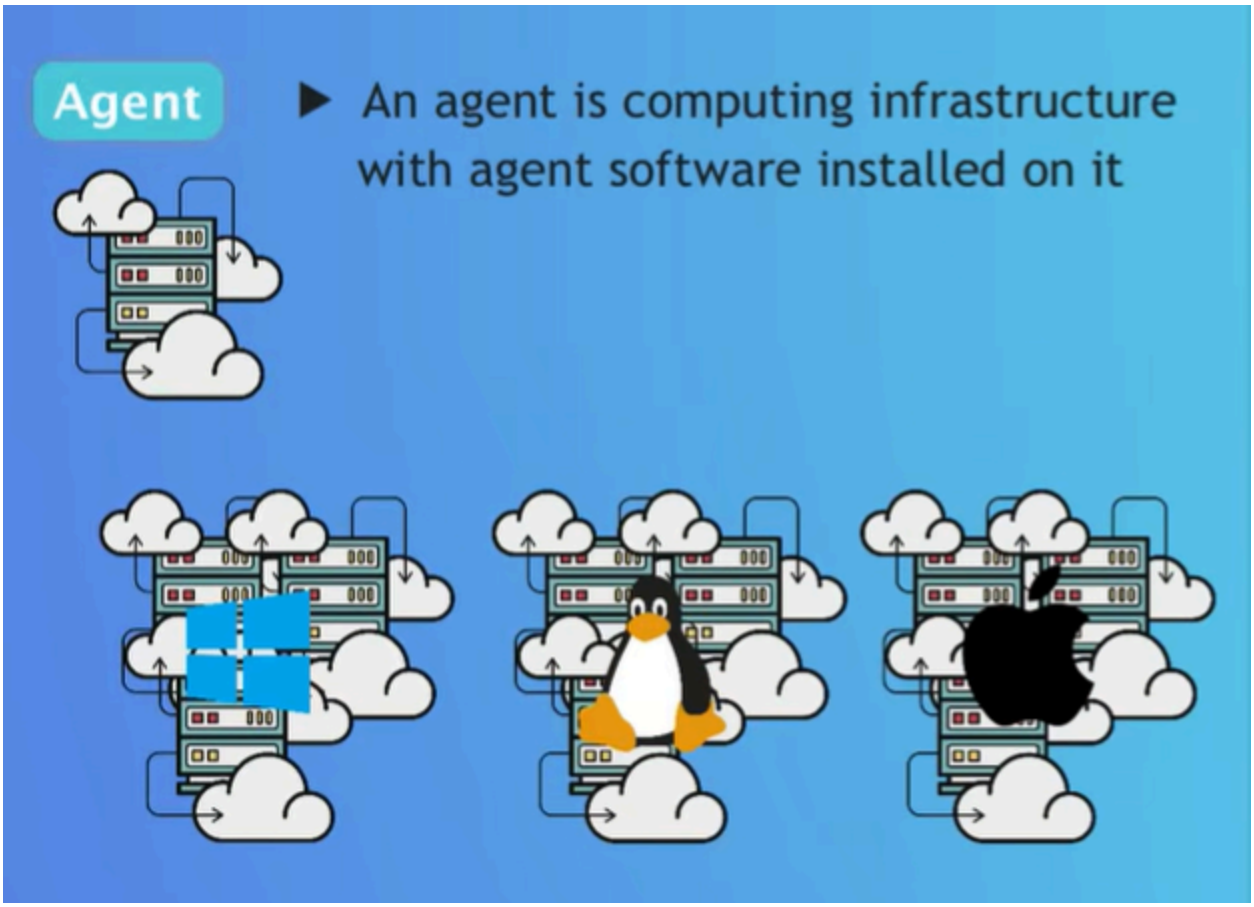
two different operating systems we want to test the application on linux and windows before we build it or let's say we test and deploy our application on linux machine but we need a database for our application which must run on a windows machine for this use case we would need two separate jobs now i'm mentioning jobs but we haven't seen a job defined in our simple pipeline yet

so what is a job exactly well all these steps actually belong to one job and can be defined like this so a job is basically a group of multiple steps however when we have only one job in our pipeline we don't need to explicitly define it that's why we could skip this in a simple pipeline but when we have multiple jobs we need to define them within jobs attribute

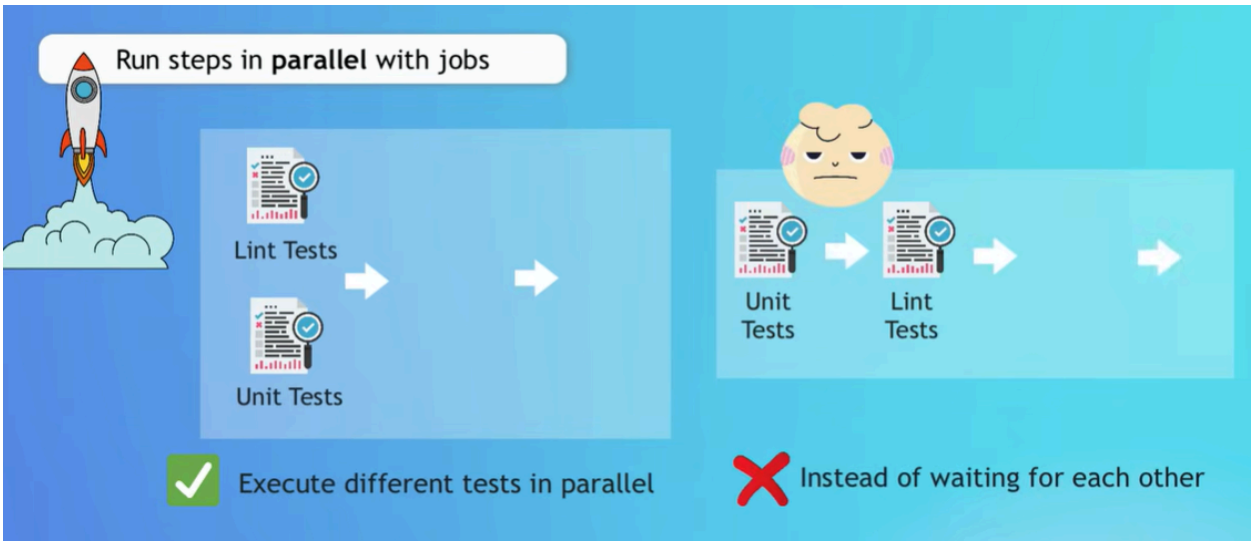


so we have multiple jobs each with multiple steps and now we can actually execute each job so all the steps within the job on a different environment also called an agent

so agent is a machine that will execute the tasks or the steps of the pipeline like running the test building an image etc and an agent is selected from an agent pool like pool of windows machines or linux machines or mac os machines etc and

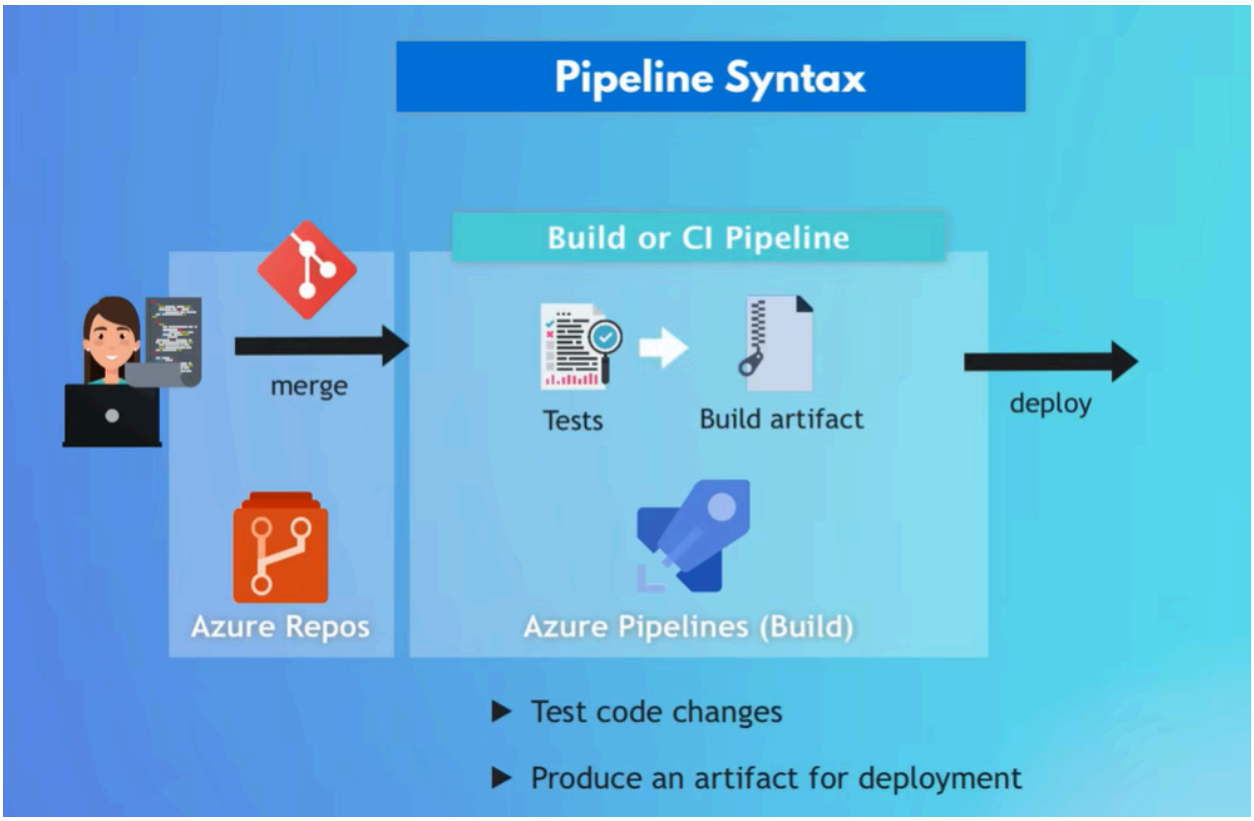


that's why we have a pull attribute in the job to define what kind of machine we want to dedicate for that job and as you see inside the pool we can define if the end image that specifies what kind of operating system we want and you can also specify what version of that operating system you want to run that step or that job another very common use case for multiple jobs would be



if we have a set of steps that can run in parallel so that the build overall is faster this could be running multiple tasks that can run at the same time testing different parts of the application so they don't have to wait for each other to complete they can all run at once

so by creating multiple jobs for all these tests you can actually run them in parallel on different environments

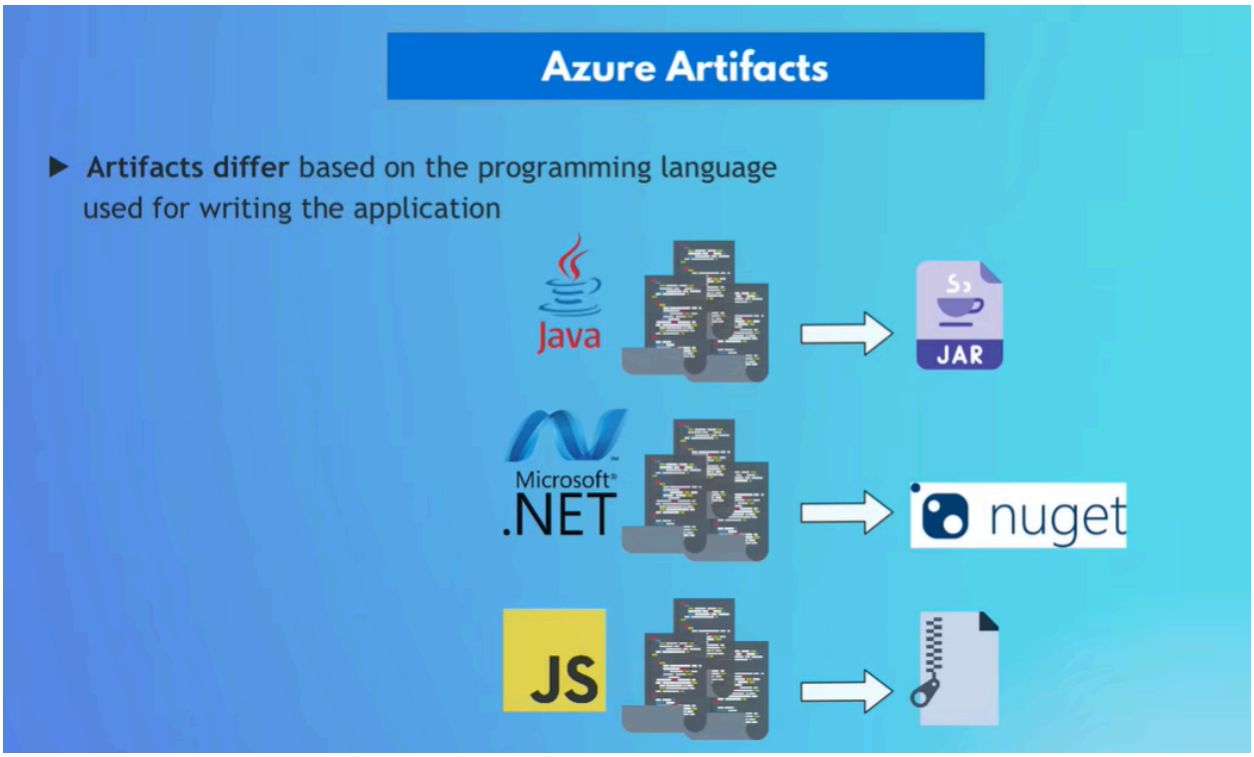


so overall the main task of this build pipeline is to test the code changes and if everything is fine produce an artifact that we can deploy

for azure devops they actually build a task extension that lets you easily use polumi in your ci cd pipelines it can be used with azure pipeline's wizard ui or the yaml configuration

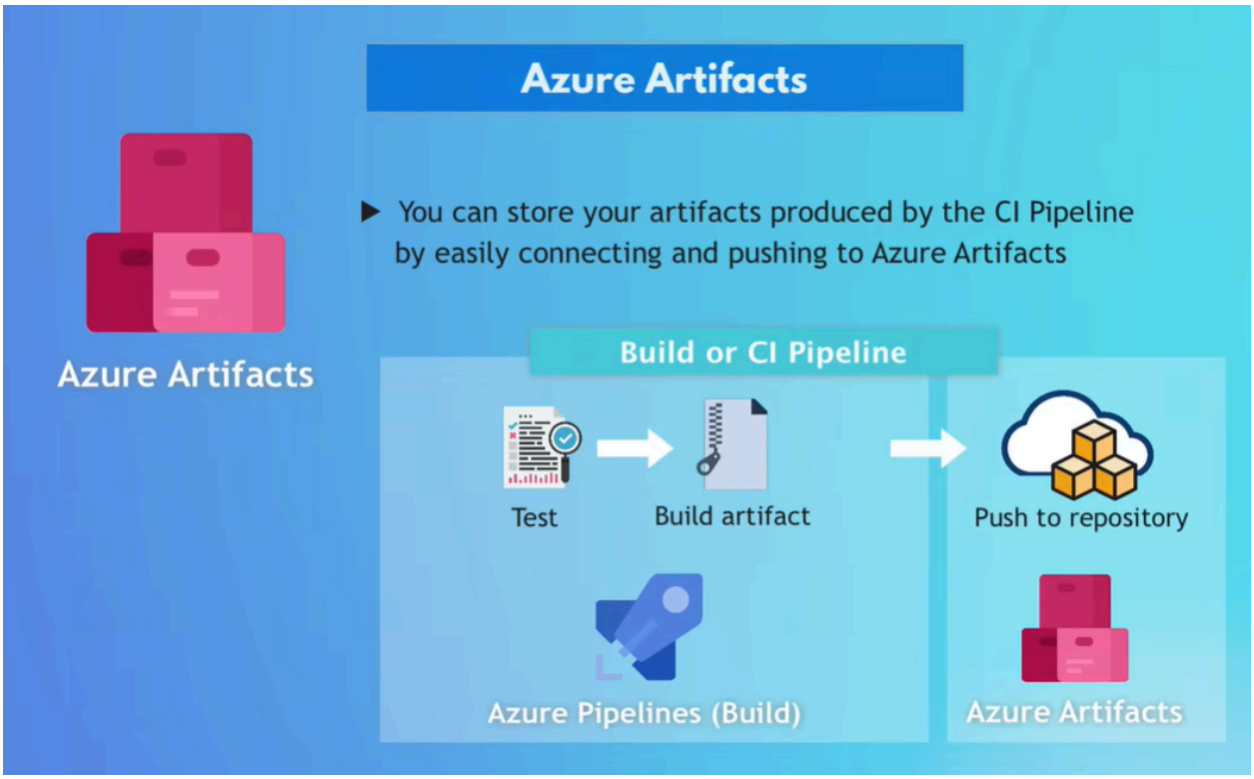
Azure Artifacts

let's move on to azure artifacts now traditionally depending on the application programming language the

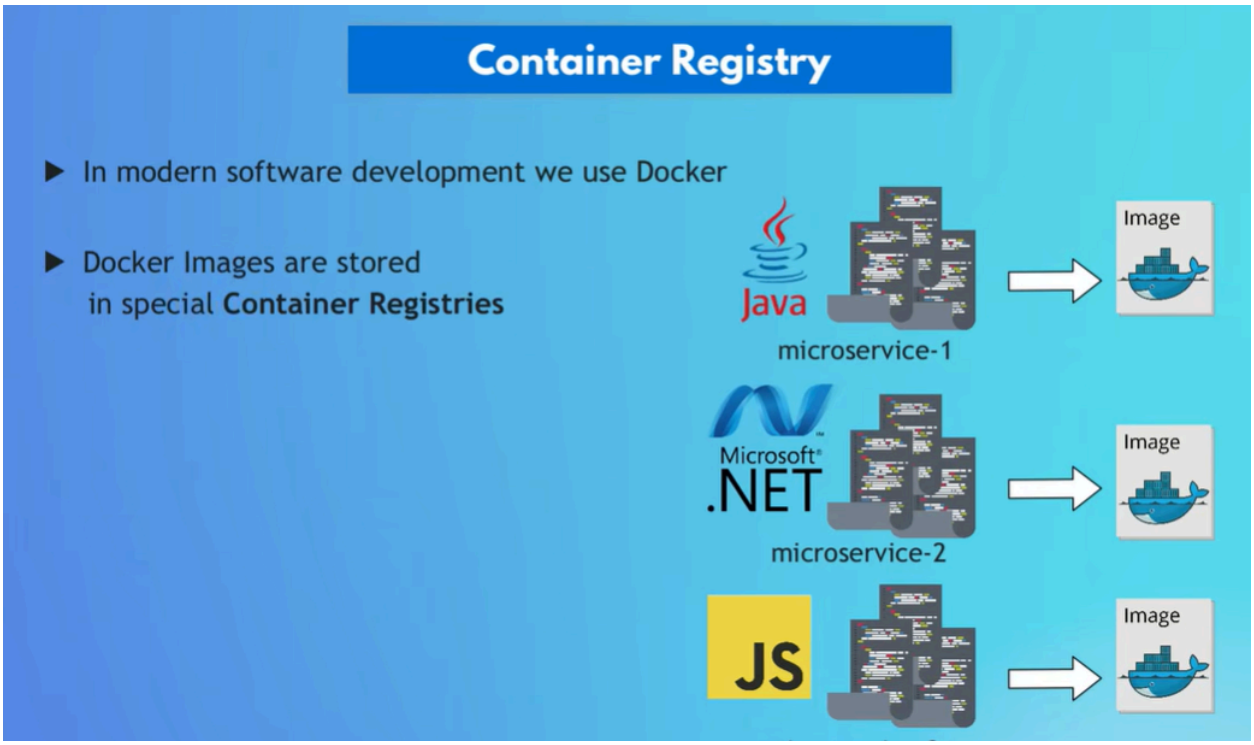


artifact produced will be different could be a jar or war file for a java application a new ga file for net a zip file tar file etc and

for storing this kind of artifacts you actually have another feature in azure devops called azure artifacts and azure artifacts actually currently supports three types of artifacts which are maven packages nuka packages and npm packages



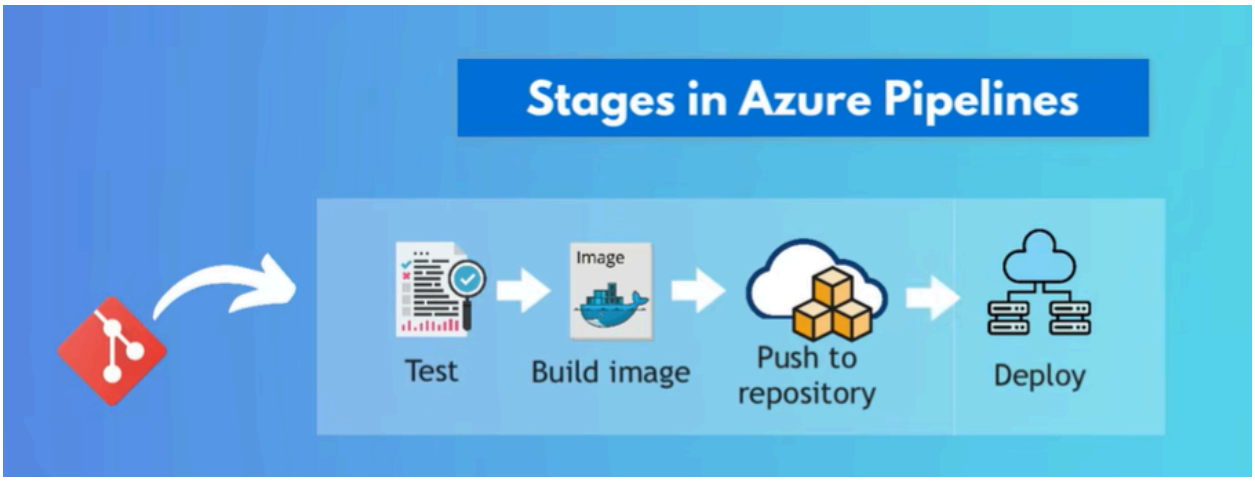
so if you're developing and building your application with any of these tools then you can store the artifacts produced in the build pipeline in the azure artifacts however in the modern software development we usually don't produce such artifacts anymore to deploy them instead we create docker images is artifacts



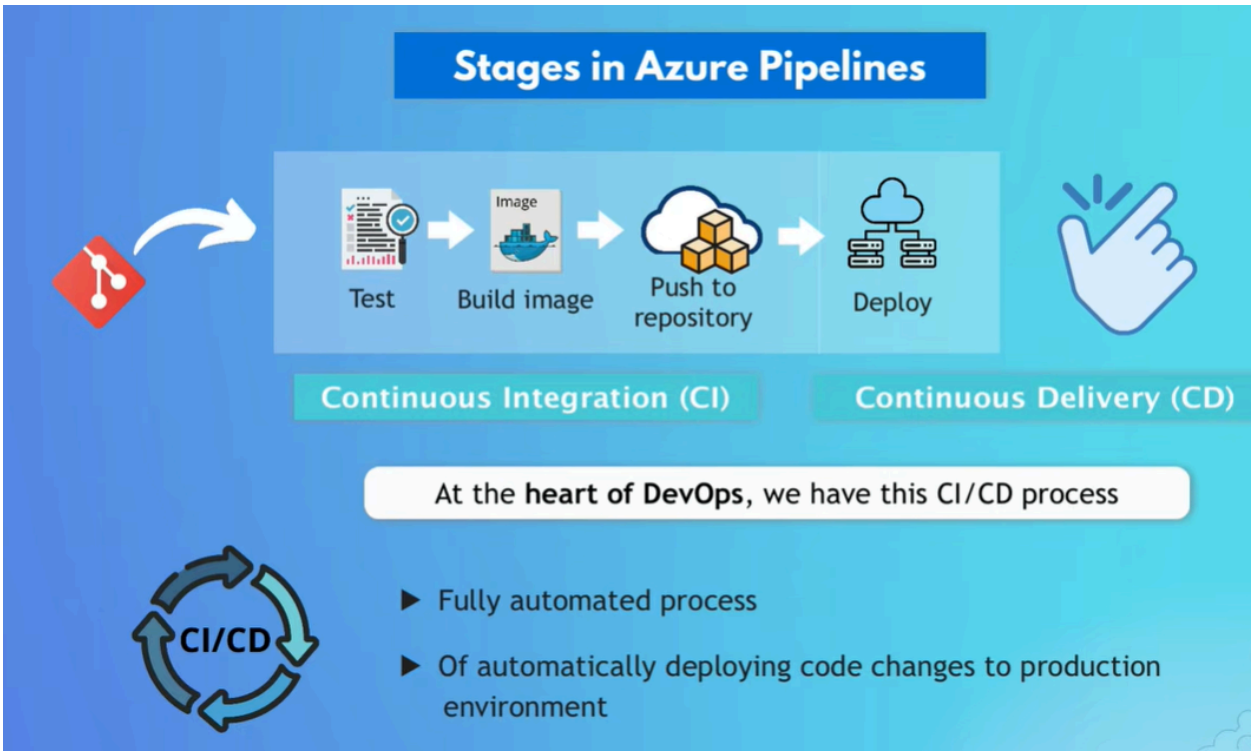
so no matter what language you use what tools you use the artifact is always the same which is a container image so if you have 10 microservices all in different languages you can still produce the same container image type of artifacts and images actually need a dedicated type of repository so if your build pipeline produces docker images you will connect your azure devops to some docker registry like docker hub azure container registry etc and basically store your images in that repository

Stages in Azure Pipelines

now let's say we successfully built our application into a docker image we pushed it to a docker repository and now it's time to deploy it to the end



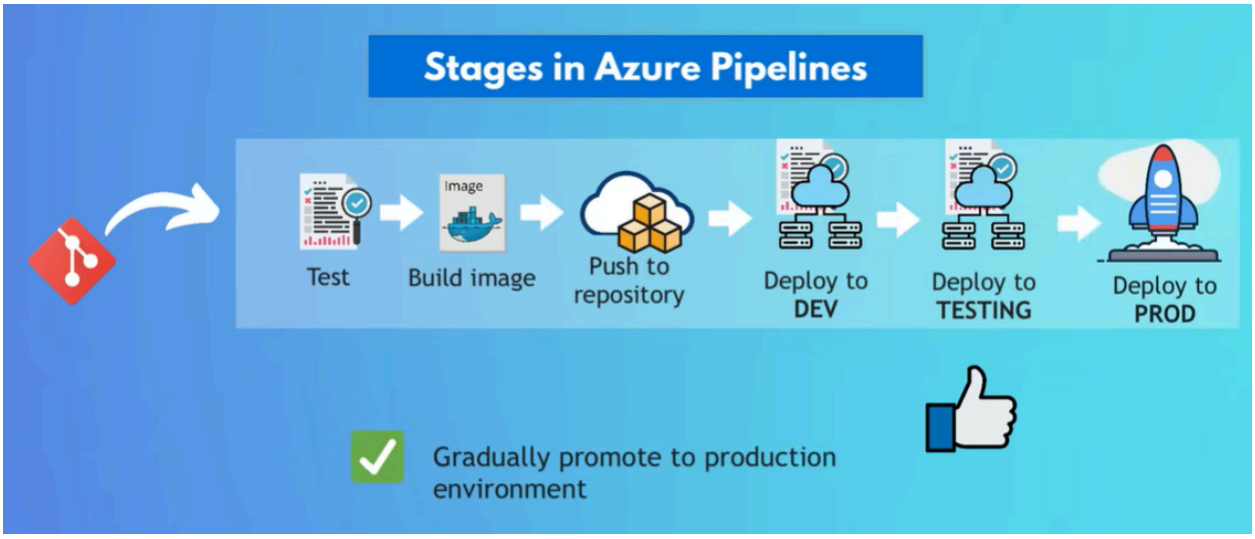
environment this would be the next stage of the pipeline which is also called the cd or continuous deployment or continuous delivery if you're not totally new to devops you already know that at the heart of devops there is the ci cd pipeline



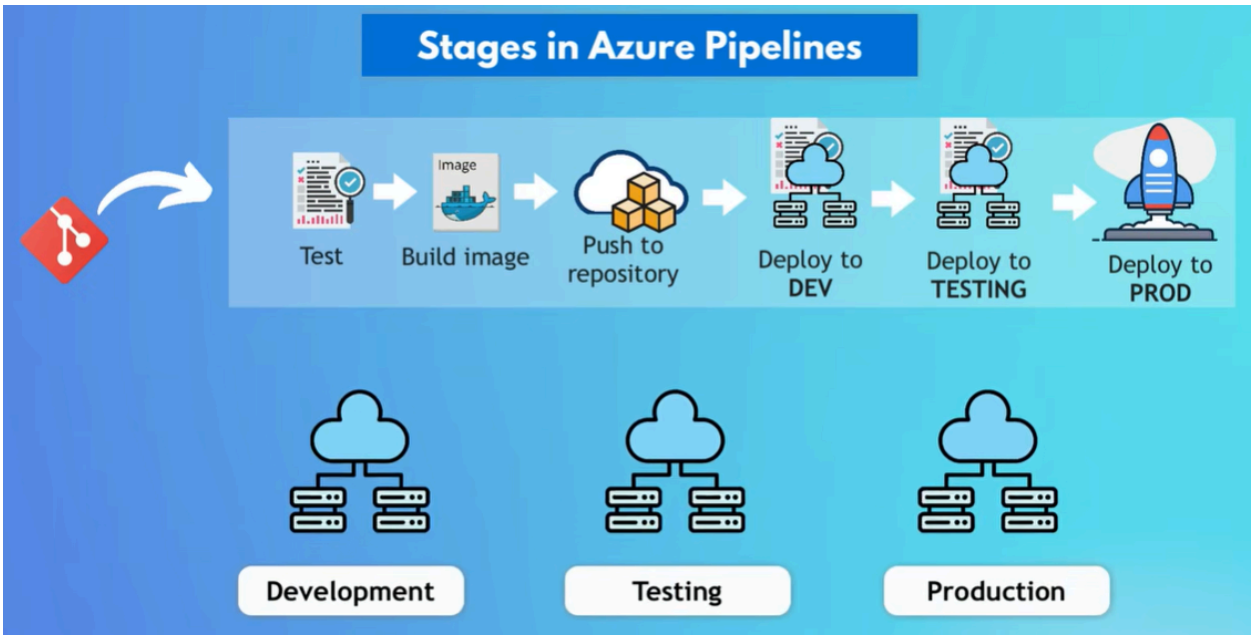
which is ideally the fully automated process of taking the code changes and deploying it all the way to the production by testing and validating all parts of those changing from is the application functioning is it secure etc and

in that process we have these two main parts which is build stage and the deploy stage again we haven't defined a stage in the pipeline yet since we only had one stage with all the build jobs when we have multiple stages

however we need to configure that as well so in order to create a complete ci cd pipeline we'll have the stages in our yaml pipeline script for building and for deploying the application and by the way we can use a specific type of job in the deploy stage which is called deployment which is specifically meant for deploy job and has some features for that purpose for example it doesn't check out the code like job type does etc

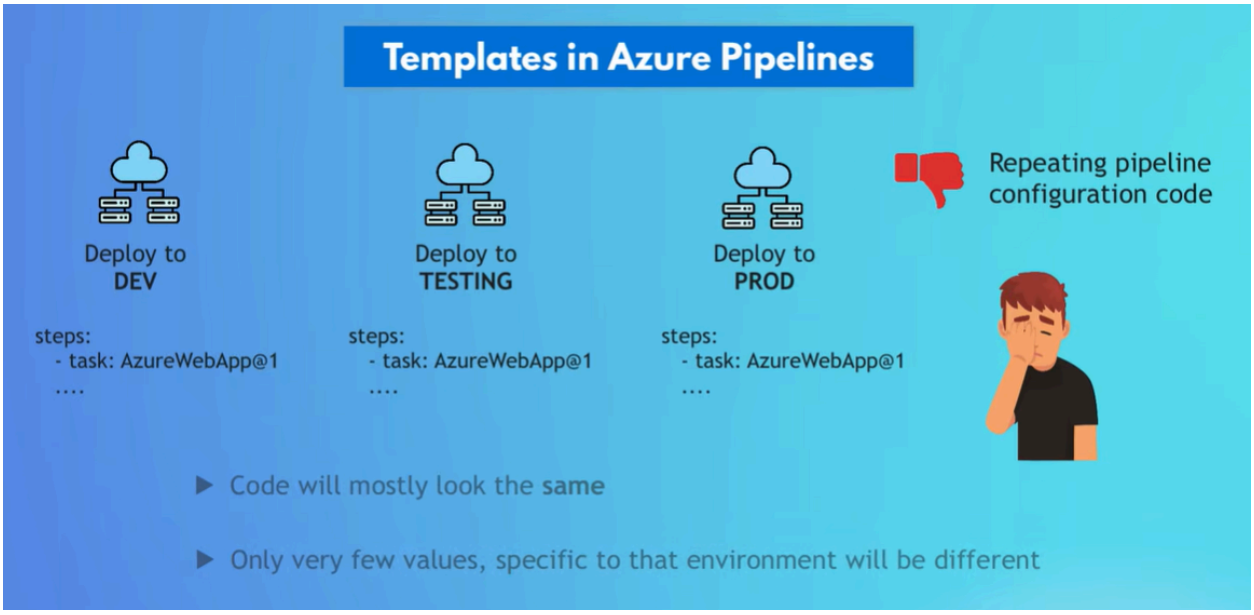


now when we are deploying the new application version usually we don't directly deploy to the production instead we deploy to intermediate environments we test it extensively and gradually promote it to the production

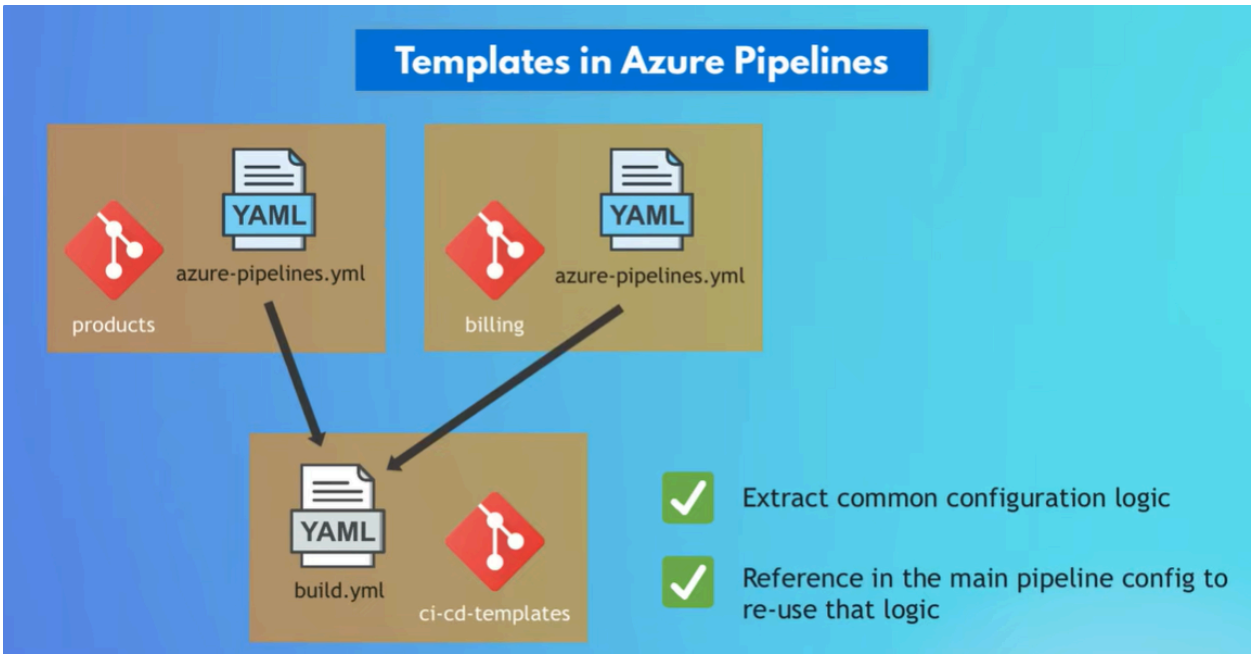


when we're almost 100 sure everything is fine common is to have development testing and production environments so we can have all these as separate stages after the build stage deployed to development deployed to testing deployed to production now the code for deployment to different

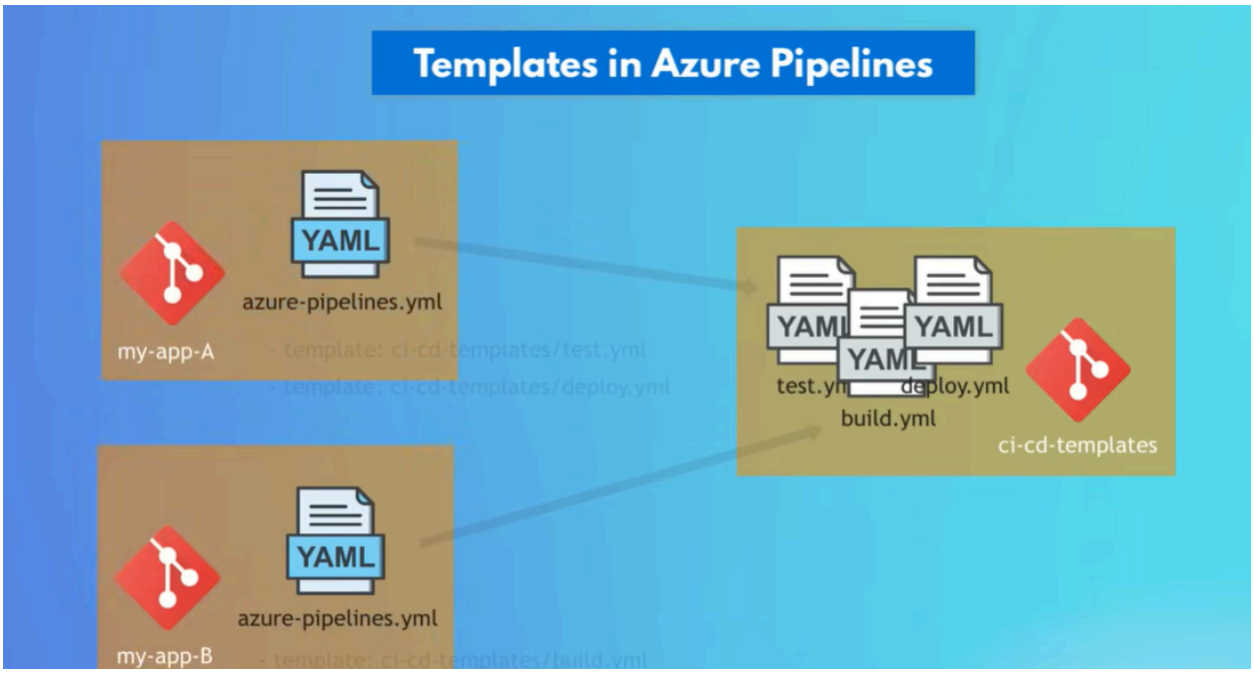
Templates in Azure Pipelines



environments will be pretty much the same except for a couple of parameters so how can we avoid repeating the pipeline configuration code in this case or maybe we have multiple applications that all have the same pipeline logic so we don't want to write the same pipeline configuration for each application in our company instead ideally we want to write that logic once properly and

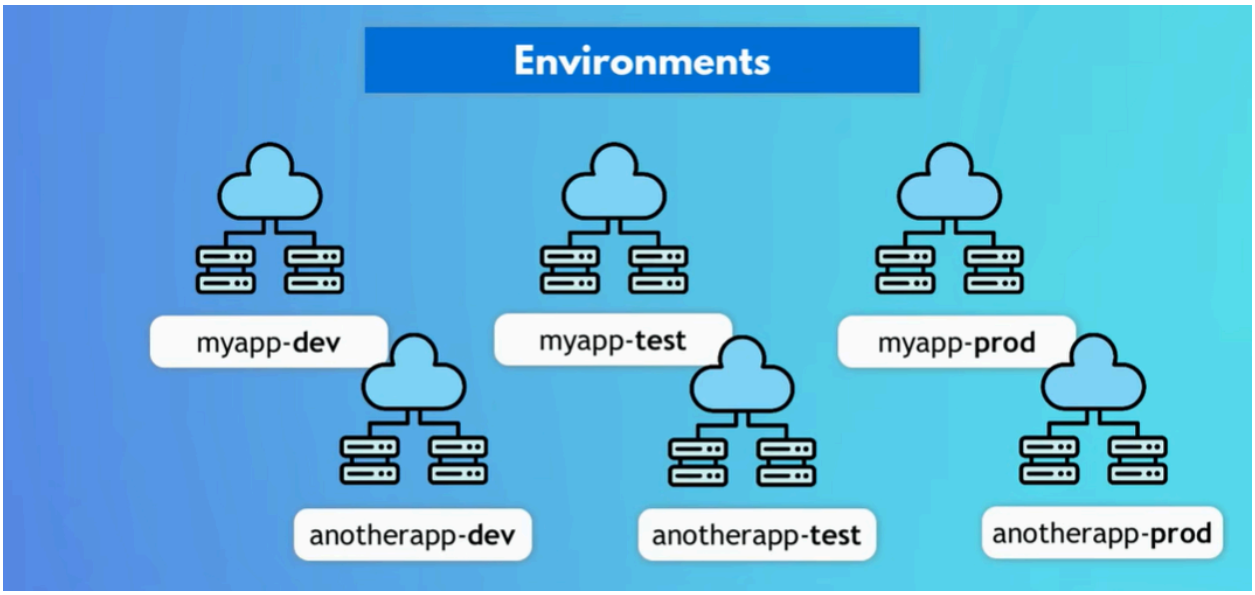


then reuse it for all the applications that may need it in azure devops pipelines yaml syntax we can actually put any code that is repeated and extract it in what's called a template which is a separate file and can be referenced in the pipeline using template attribute and it can even be configured with parameters so it's like a reusable piece of configuration that you can reference in different pipelines



so you can split your entire pipeline into multiple individual files and these files can even be stored and managed in a dedicated separate repository and as i said all the pipelines can reference them and by the way you can have a template for a job a step or stage so for any of these levels and you can also have templates within the templates creating a hierarchy like this

Environments in Azure Pipelines



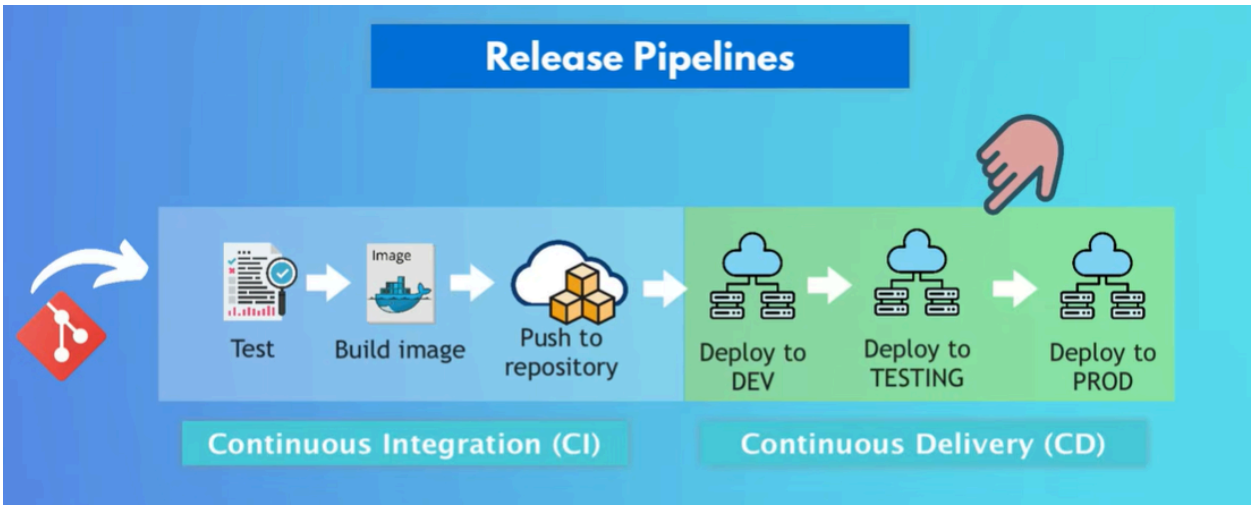
now when you have multiple environments for multiple applications it may become difficult to have an overview of what version of what branch is deployed where or when the code was last deployed to a specific environment and so on and that's where the environment feature comes in which is part of azure pipelines

we can create environments in azure devops which will map to the actual deployment environments and then you can configure in your pipeline which of these azure devops environments you want to deploy to so you kind of have this abstraction there and once the application gets deployed to these various environments you can actually view the deployment history per environment so this can actually be some additional valuable ui feature that gives you a better overview of your deployments plus the deployment status can also be linked back to the original ticket so you have that additional information for the feature or improvement to which stages or environments it has been deployed to already which again can be pretty convenient now that deployment process or

Release Pipelines (CD)

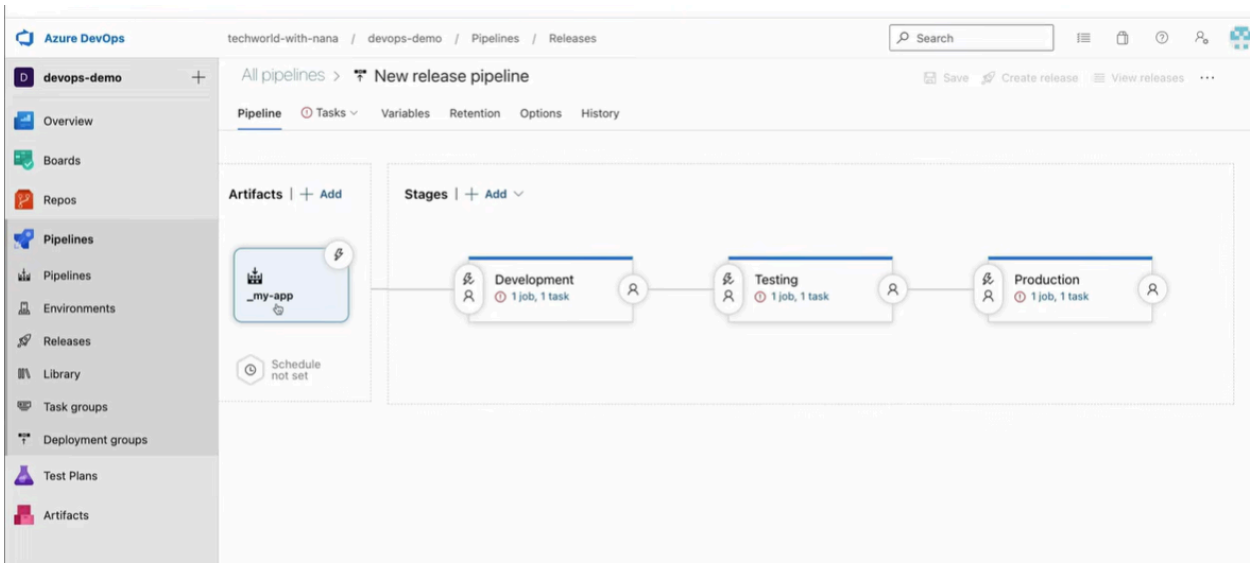
deployment part of the pipeline which as i said is called cd or continuous deployment

on azure devops can also be built as a separate pipeline called release



pipeline interesting to note that many ci cd platforms like jenkins gitlab csd etc they have one pipeline for the whole process so we have one file and one ui unit for both in this case it will be split into separate ci and cd pipelines so the way it works is that you select a build pipeline that produces an artifact or you choose the already built artifact location or source and you create a release pipeline for that artifact note that release pipelines in azure devops can only be created using the ui so you have no yaml file

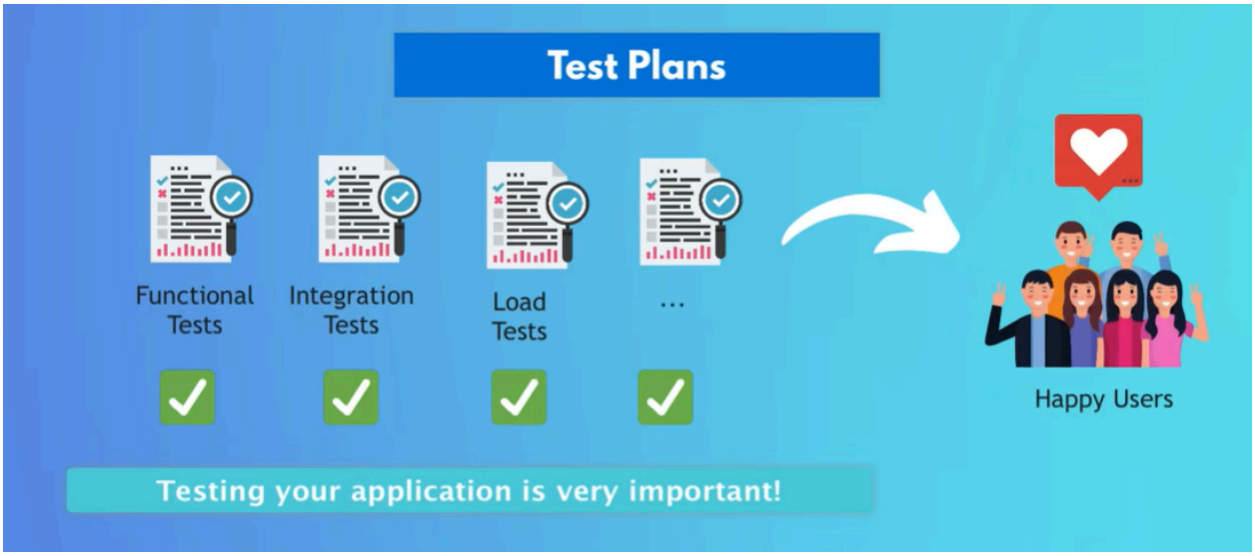
for that however the pipeline structure itself is the same you can create steps by choosing from available tasks and also have multiple stages like deployment to development testing and production



so with release pipelines you say when this build pipeline completes and successfully creates the docker image artifacts for example trigger this release pipeline so this way you chain them but as i quickly mentioned as the artifact source you can actually use not only the build pipeline output but also already built artifact from various sources now generally it's usually a better idea to always have one ci cd pipeline defined in yemel instead of splitting that into two plus you have all the benefits of scripting your pipeline and making use of the reusable templates etc so the release pipelines is probably for more specific use cases maybe when you want to deploy existing artifacts from the artifact repository directly but as i said usually you should have one pipeline for the complete cicd process

Test Plans

now an important part of an application release process is testing and you need to extensively test your code changes before deploying it to production and of course the more complex the application the more tests you need



so in azure devops you actually have a dedicated section for tests and here you can actually create a unified central view of all the test cases or many of your test cases that need to be checked before giving a green light to production deployment and here you can create manual test cases or plans

so when a new feature is being released a tester can go through these steps and test the application but this could also be automated tests which will be executed as part of the ci cd process and the test reports from the pipelines can be published and viewed here and

the main advantage of this is that you have own centralized place with an overview of all the test plans whenever releasing your application whether these are automated tests that were run in the pipeline or manual tests from developers product owners testers etc and again you can see the results of those test executions in your feature descriptions to decide whether you can release the changes or not and you can even view and run the test cases related to a feature directly from the convent board

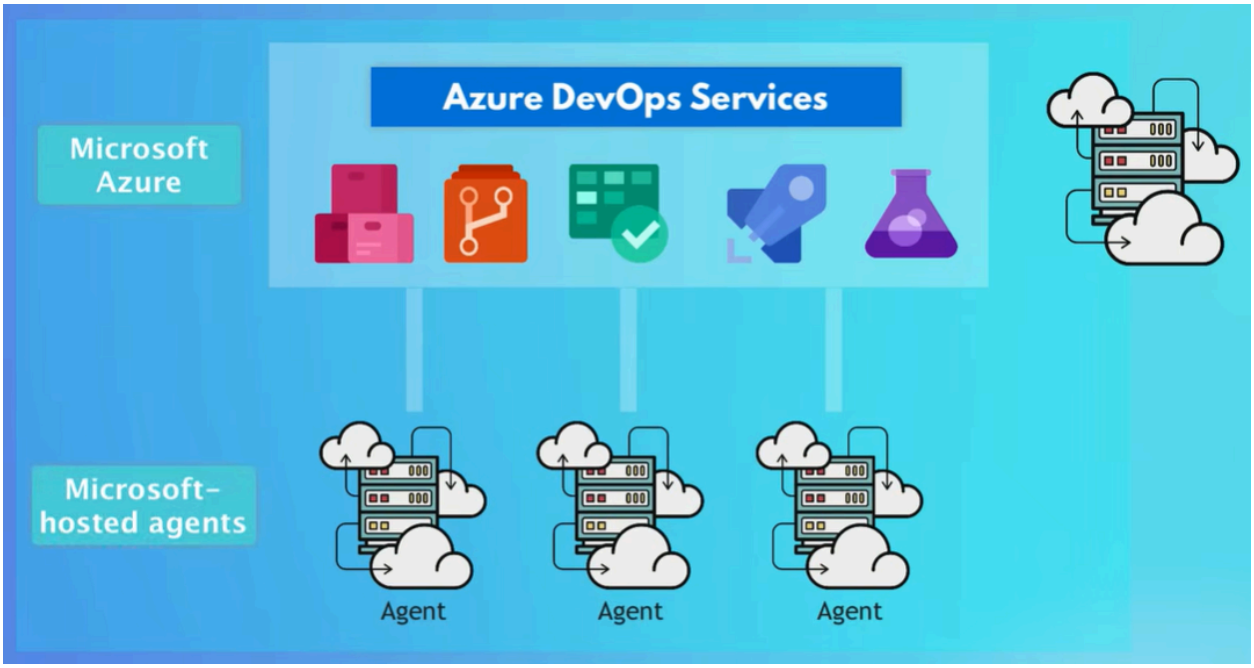
Azure DevOps Architecture

now the pipelines execute tasks like running tests building an image etc and as i mentioned you can execute these tasks on different machines with different operating systems

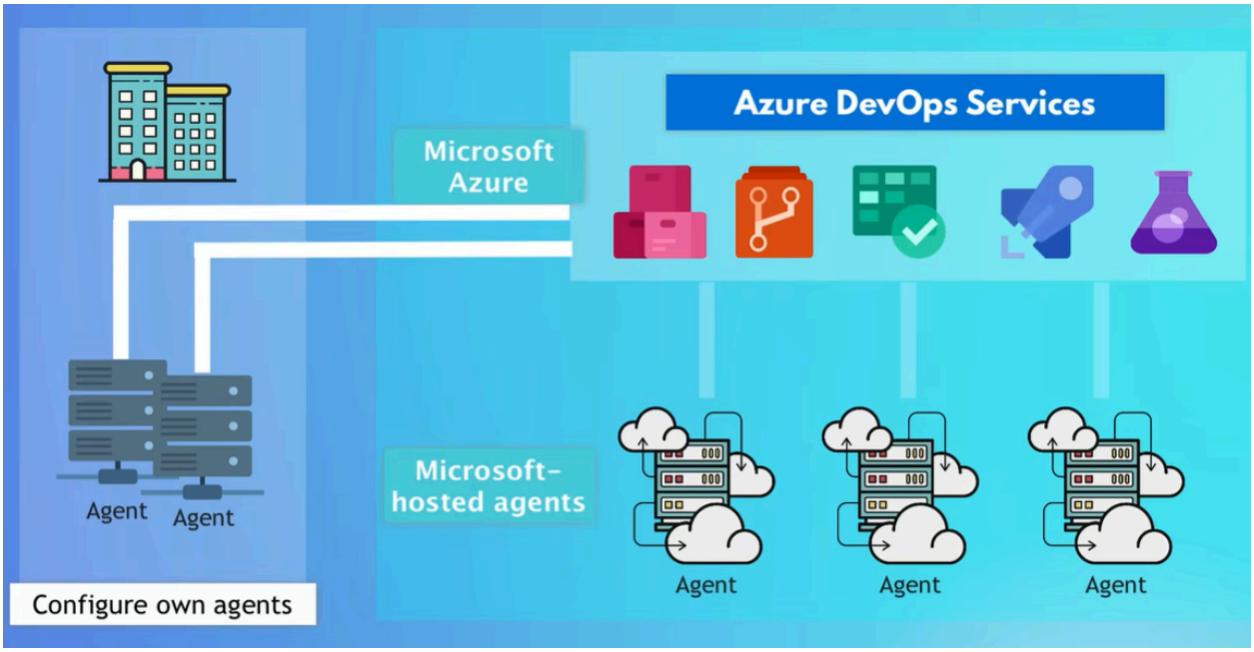
so where are those machines exactly and how do we get access to them to run our tasks and which machines and which environment do we get to answer this question

let's look at the azure devops architecture on a high level at the core we have what's called the azure devops services a software as a service or the managed online solution from azure and

that's the main part where configurations are made pipelines and repositories are created and stored etc so these are all dedicated machines for those things but the pipeline tasks themselves run on separate machines called agents which are connected to the azure devops services platform



now who manages these agent machines well azure offers managed agents as well so you can let microsoft actually manage the whole setup for you including the main service which holds the configuration plus the machines that actually execute the pipelines



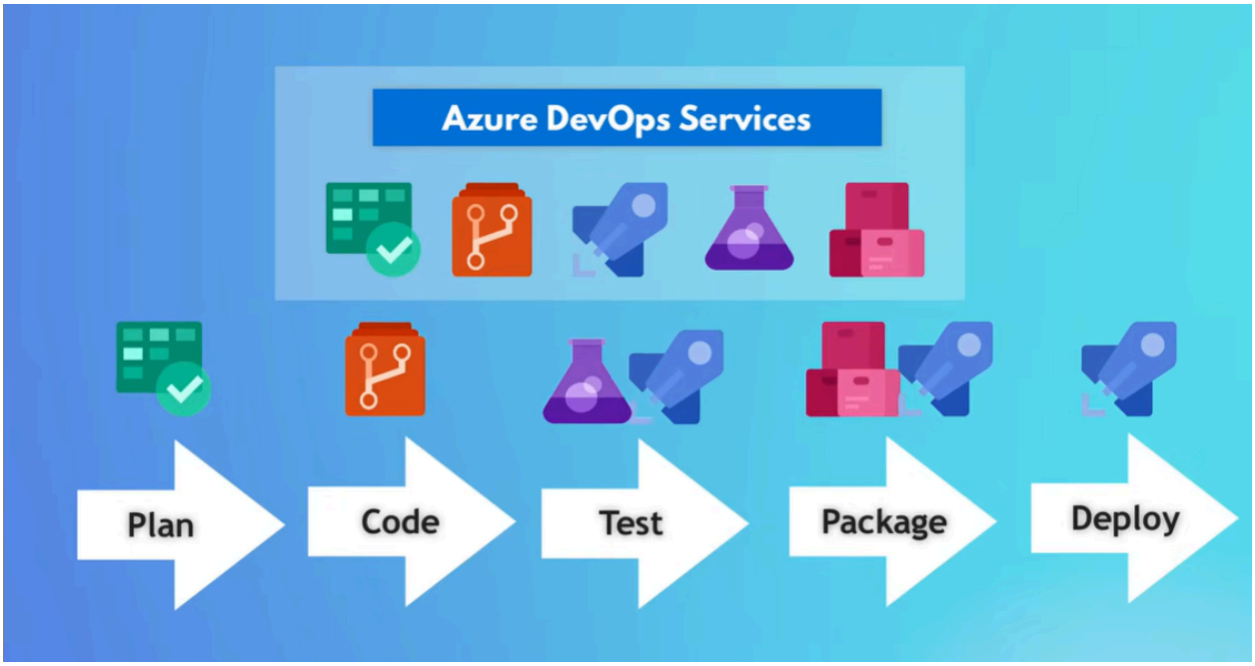
however in practice many companies need control over these machines plus they want to remain flexible and save costs maybe they have these machines on premise or even on another cloud platform so they want to make use of this so you have an option to configure your own agents and connect them to the azure devops platform or you can even have a mixture of both and this is actually pretty similar architecture to what other similar platforms like eclipse icd etc look like so nothing really extraordinary here and since this is a

Pricing Model

managed service of course you have to pay for these services and using for these resources but azure devops does have a free tier to get started with that basically includes a certain amount of free resources that you can use to get started with including using managed agents for your pipeline jobs great so till now we actually saw

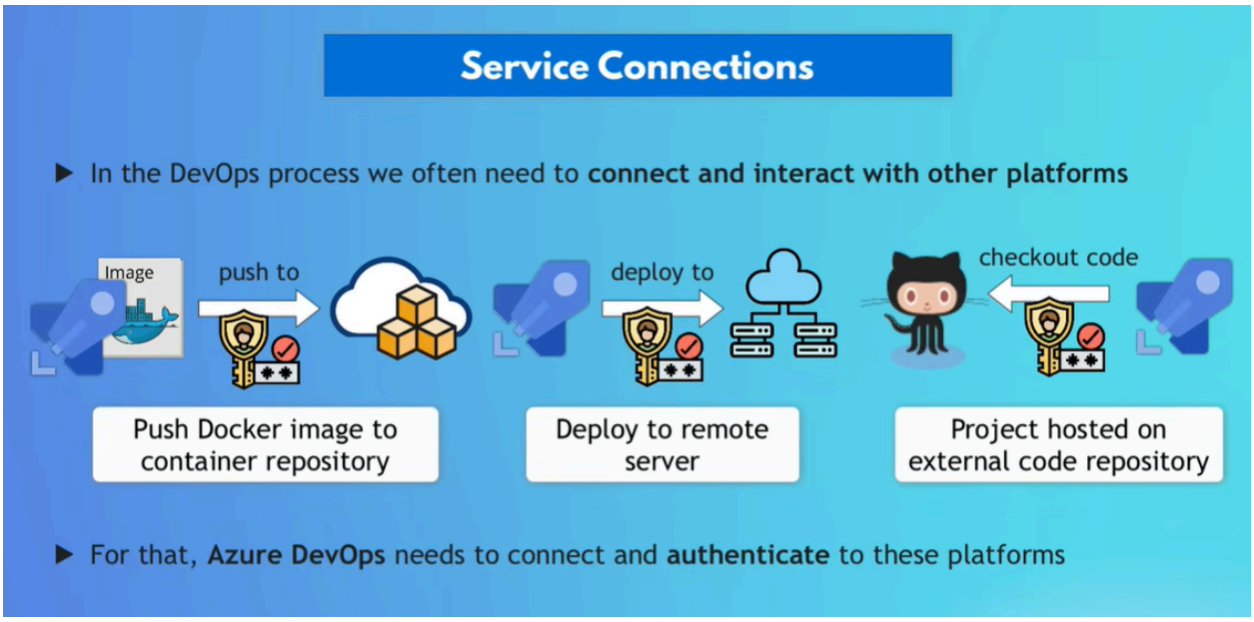
Service Connections

various features of azure devops that map to different parts of the application development life cycle starting from planning the task all the way to developing and deploying it to the end environment now throughout these processes we actually have tasks that we execute on



other platforms for example when we build and push an image that needs to be pushed or stored in an external image repository right which is outside of azure devops or

when we deploy to a remote server it will be on some cloud platform like azure aws or even on-premise or maybe we deploy to kubernetes cluster etc plus we may have the pipeline connected to the external code repository in github instead of using the azure repository



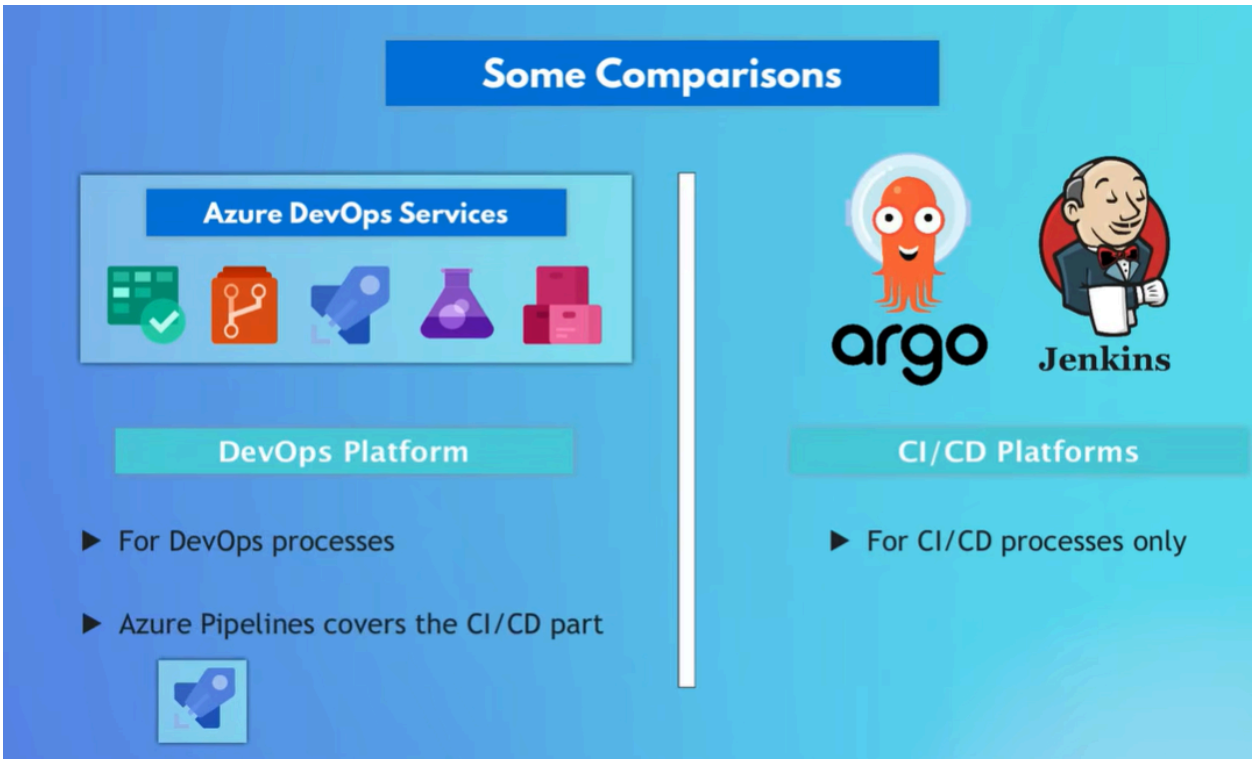
so for all these tasks azure devops needs to connect to those platforms right and normally you have credentials like username and password or access token from these platforms that you need to make available in azure devops so that it can connect and authenticate itself with those platforms now for those use cases in azure devops you have what's called service connections feature which makes managing access to external platforms much easier first of all it's less configuration effort because you don't have to create these credentials in the respective surveys and then replicate in azure devops instead the credentials are created automatically when azure devops connects to those services and

second advantage is that it's more secure actually because service connections use short-lived credentials which as i said gets generated on the fly when the connection is established so you don't have to worry about rotating or invalidating credentials and so on and the service connections can be created in the project settings section

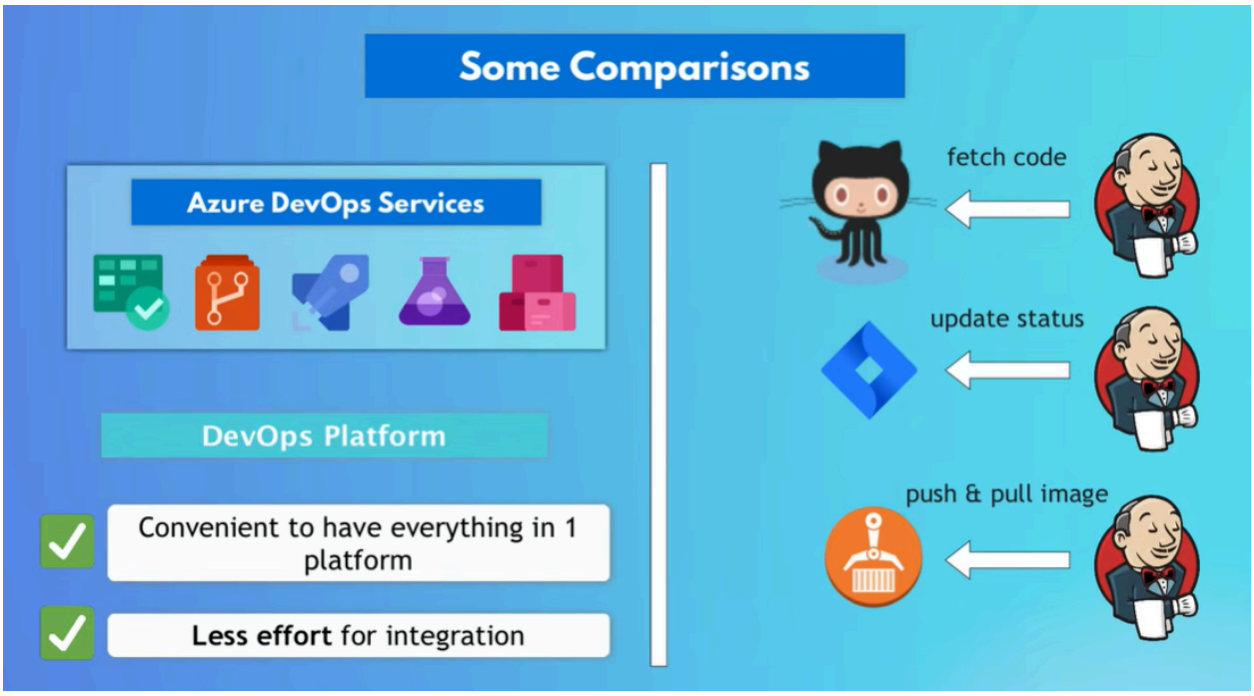
so you have a separate section for administering the project here you can manage settings for all the features like boards repositories pipelines test plans and artifacts plus as i mentioned you can use self-hosted agents to run your pipelines and this is also where admins can configure these agents as well

Comparison with other platforms

the various features of azure devops and what the azure devops platform even is you're probably wondering if this is so great why aren't all the projects using it or is it that great and what is the difference from other similar platforms like aws or gitlab etc and which one are you supposed to learn should you become an expert in azure devops and ignore all other tools



so let's look at comparison with similar tools and answer the question about which one to learn first let's compare it with traditional ci cd tools like jenkins or modern ones like argo cd circle ci etc the main difference here is that these are exclusively ci cd tools right so jenkins circle ci etc they're specifically built to create and manage cicd processes

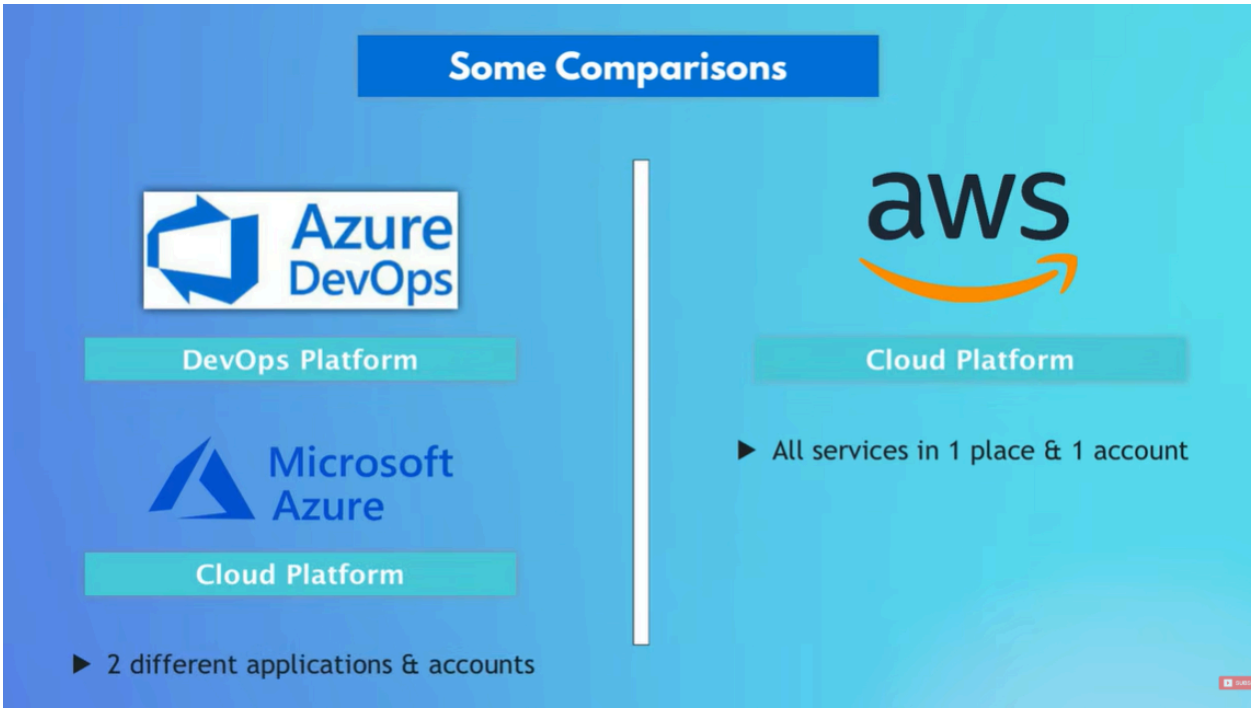


but azure devops actually strives to be the complete devops platform not only the ci cd so any feature you need for covering the whole devops process including the ci cd is in one place which can be extremely convenient as i said because in devops you need multiple tools for different parts of the process like jenkins for build code repository jira board artifact repository etc

which means you need to integrate these tools together so you have an effort in putting all these tools together like connecting git repository with jenkins connecting jenkins with jira to update status of feature tasks etc so when you use a platform that offers these services in one place obviously it's more convenient because they are already integrated and plus you get a better traceability meaning you have links between all parts of the process feature task has links tocorresponding feature branch or pull requests to its pipelines maybe the artifact that was produced with the version and so on so you have a better overview because you have linked data from all features a direct comparison to azure devops is however gitlab because gitlab which started off as a git repository actually made a turn and decided to create an all-in-one devops platform as well

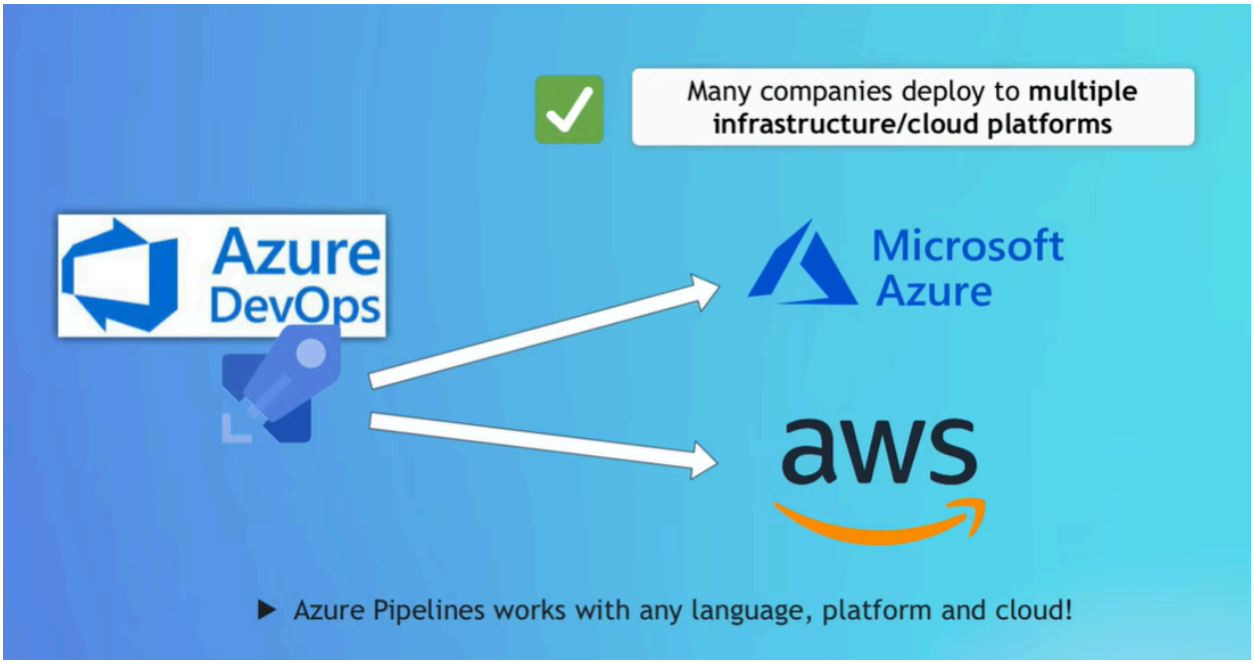
and to be honest many of the features and use cases are pretty similar between gitlab and azure devops or generally how the things work and

however aws is way bigger and way more encompassing than just the devops processes and here i want to mention an interesting note about comparing azure with aws and where azure devops platform actually plays a role in that



as we know azure and aws are both cloud platforms where you can create and configure your complete virtual infrastructure create virtual servers and use a bunch of other services as well but while aws has all its services on one place with one account azure platform and azure devops platforms are more separated so you have two separate accounts for them and you can manage them separately and even use each platform without the other however they are both obviously microsoft products part of the same ecosystem so they have some integration so essentially they are still connected so for example the hosted runners for azure devops run on azure platform as well as the code on azure devops repositories are also hosted on azure platform and you can also integrate the azure active directory which is one of the azure services in your azure devops account

so they're two separate platforms but integrated with each other for various use cases and that means when you want to deploy to azure virtual machines azure app services etc or azure kubernetes service from azure devops you basically have to connect to just like you would to any other cloud platform to deploy to it and



an interesting use case in many projects many companies is that projects who use azure devops actually deploy to multiple infrastructure environments or cloud platforms so they may deploy from azure devops pipeline to azure virtual machines and aws virtual