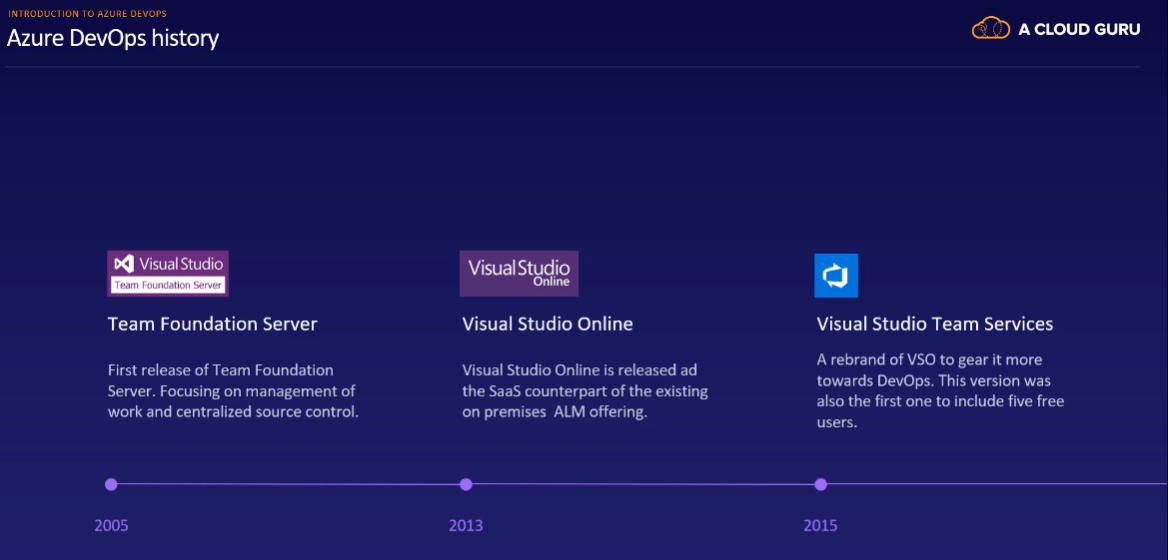
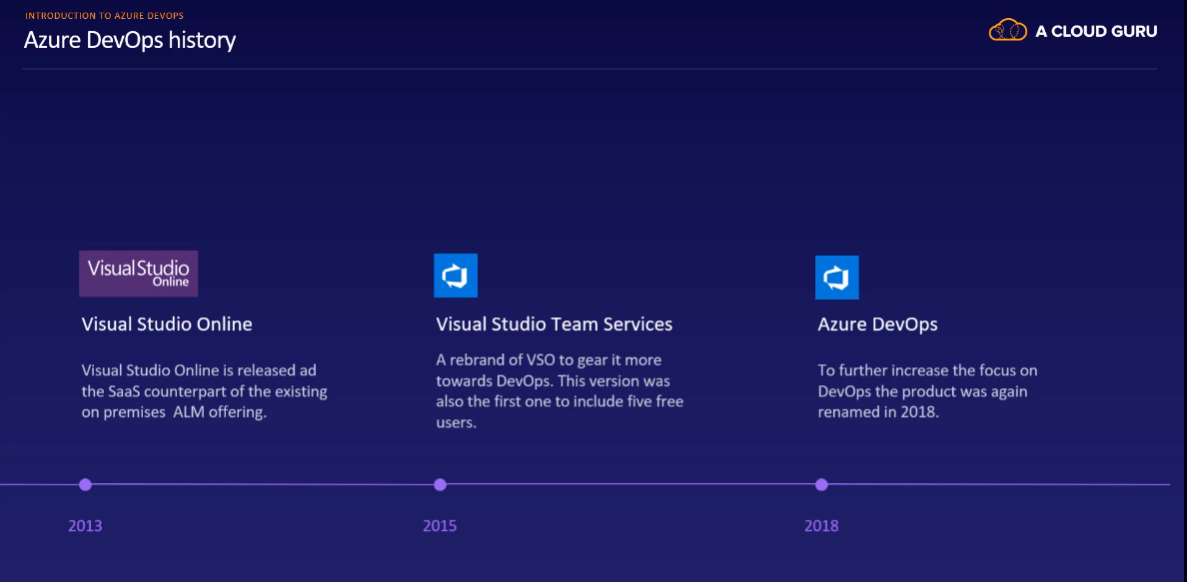
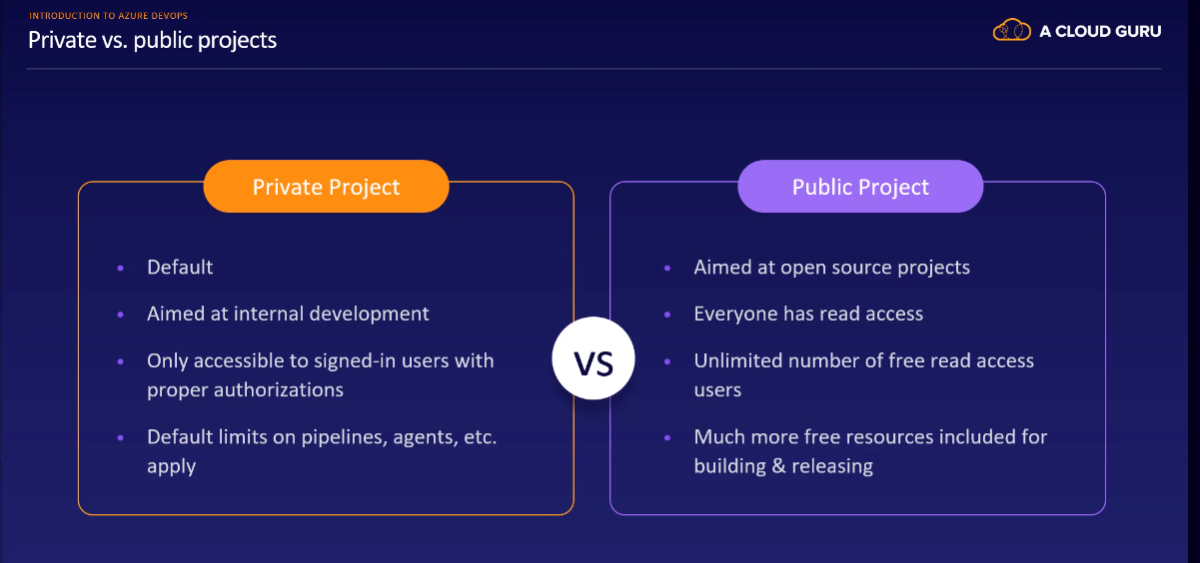
**AZURE DEVOPS**

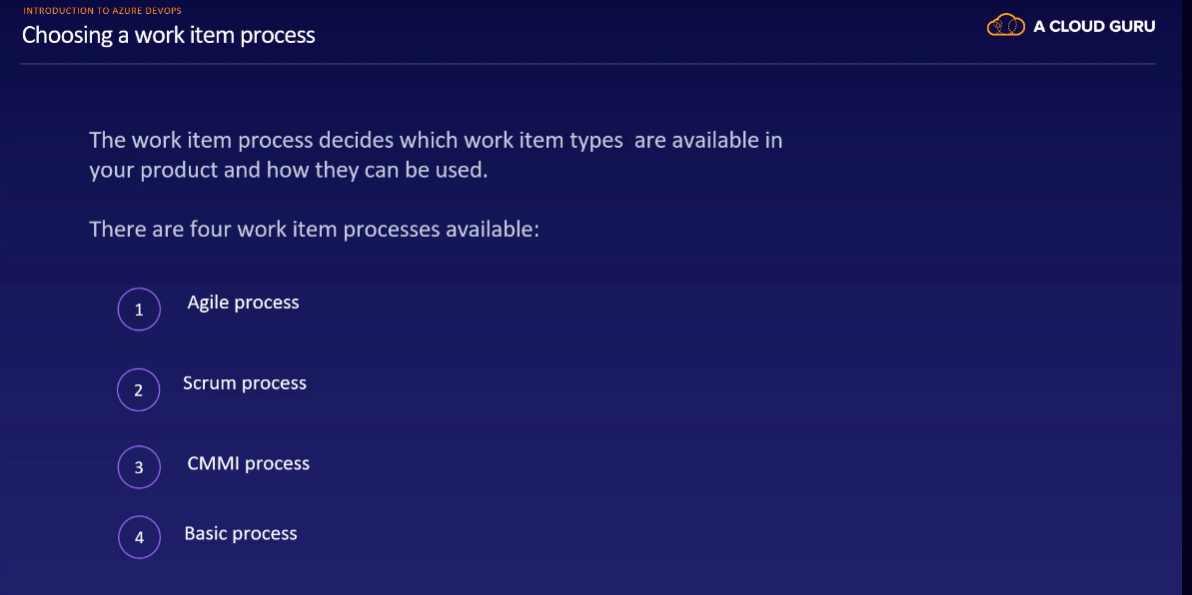


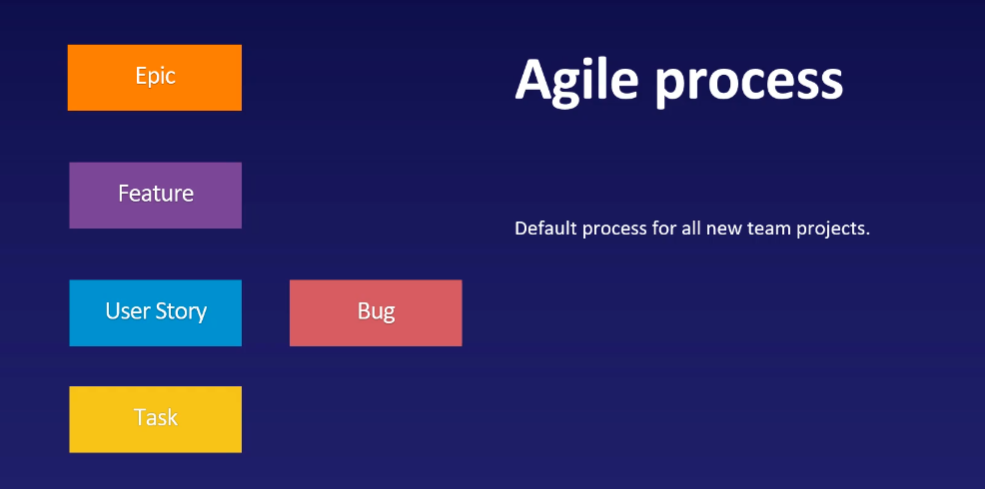
****

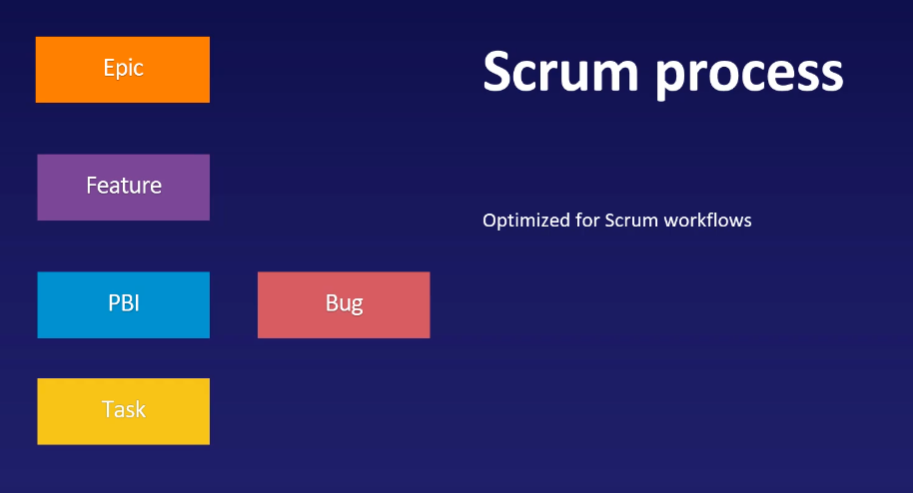
* Within an organization, you can create one or more team projects.
* Each team project will get all Azure DevOps products (boards, repos, pipeline, test plan, artifacts)
* Within the team project, you can create one or more teams.
* In teams we will have members.
* Sharing or relating items is not possible between team projects.
* Team projects are strongly separated containers and should be considered as incrossable security boundaries.

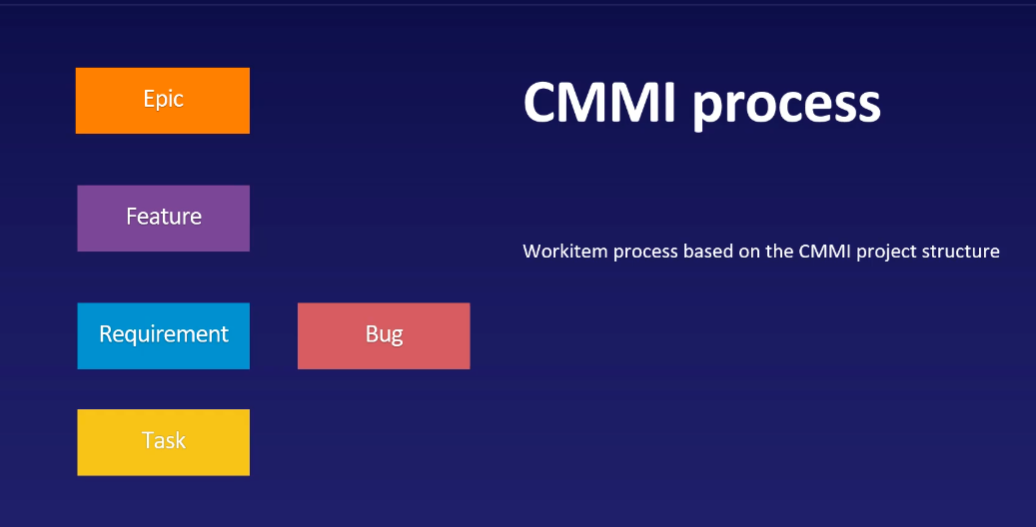
****

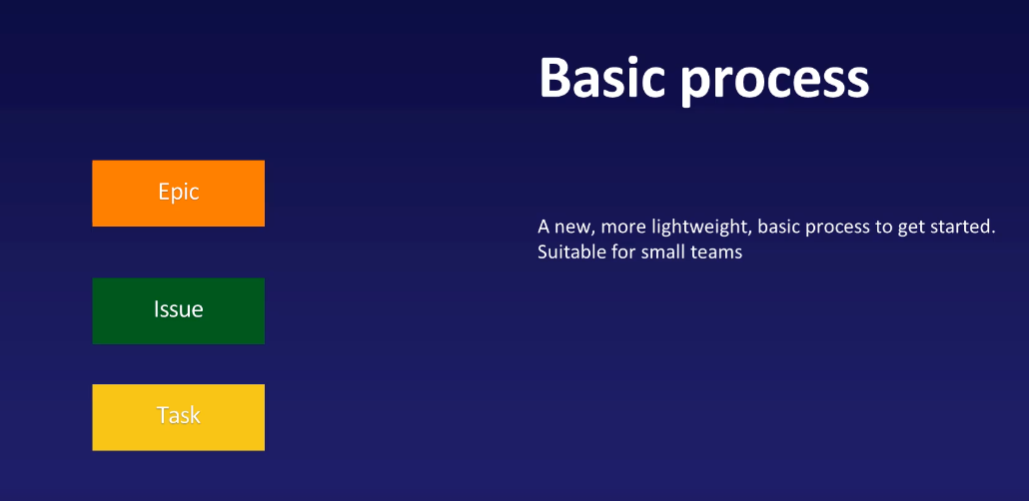
****

****

****

****

****

****

****

* Stakeholder license is free.

****

**Work Item::**

Work Item is used to refer to any piece of work that is noted down in Azure boards.

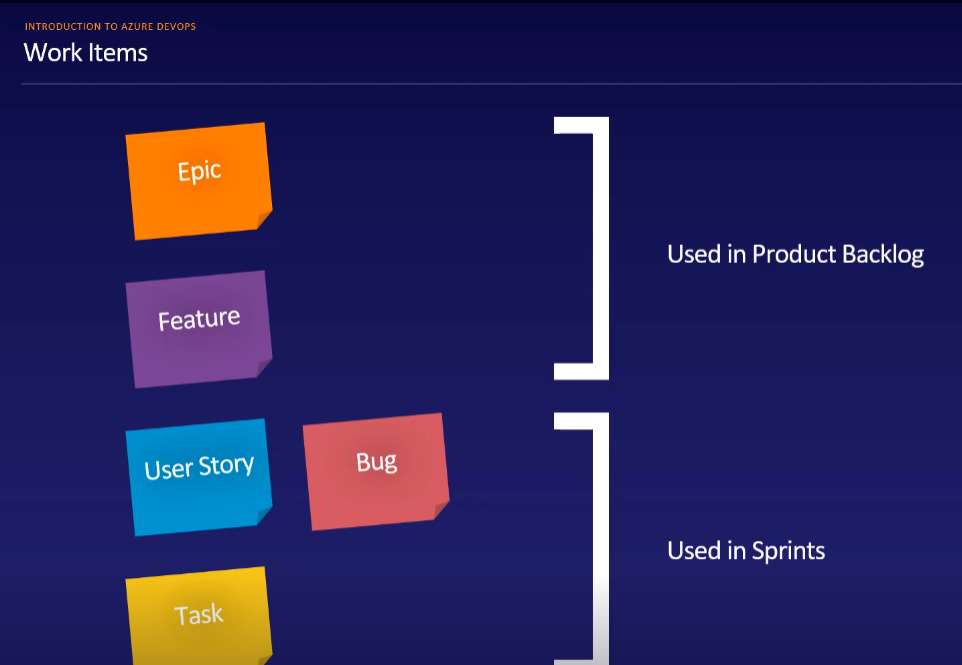
User Story :::

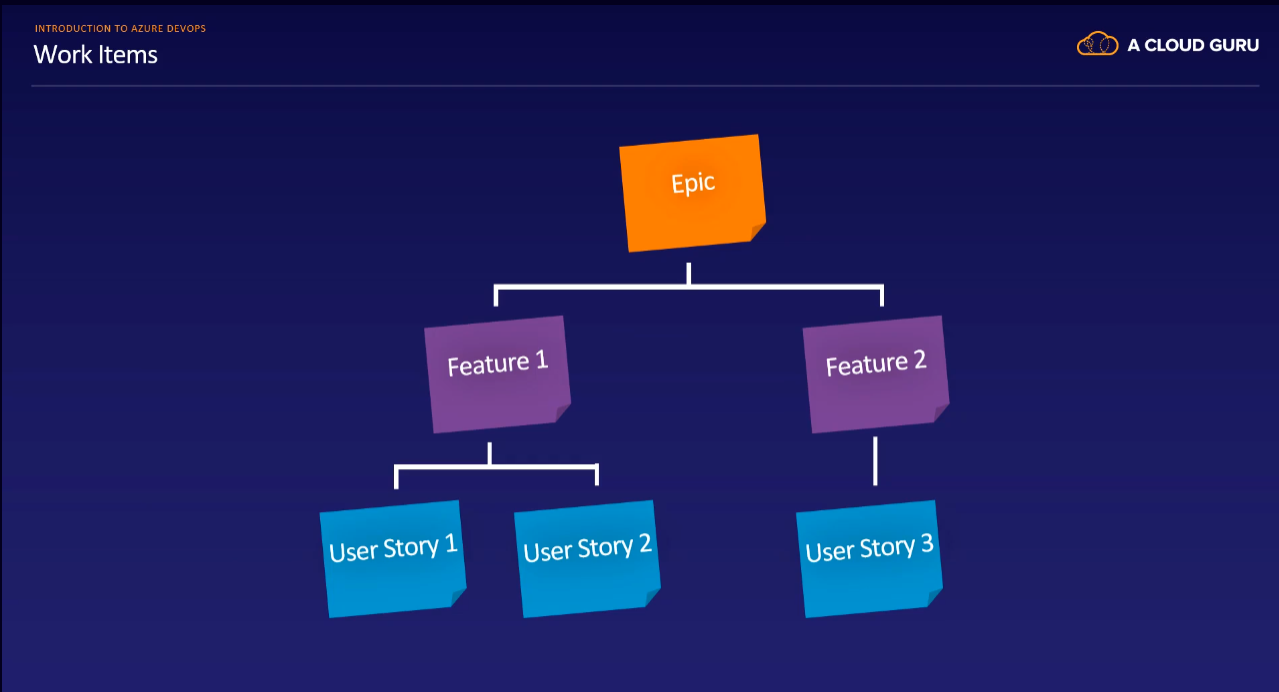
User Stories are the lowest type of Work Item and should be used to describe a single piece of work that would deliver value to an end-user.

Feature :::

A Feature is a more high-level description of something that a product owner would like to see realized in his product and would consist of multiple User Stories.

Epic :::

Epics are often used for long-term road mapping. 



Area paths ::

Area paths are used to group work items based on the functionality group they belong to.

Iteration Paths:::

Iteration Paths are used to describe a sprint or iteration.

Bugs::::

In all work item processes, except the basic process, you can choose whether to add bugs as an extra work item type or not, using configuration.

If you do so, they are on the same level as user stories.

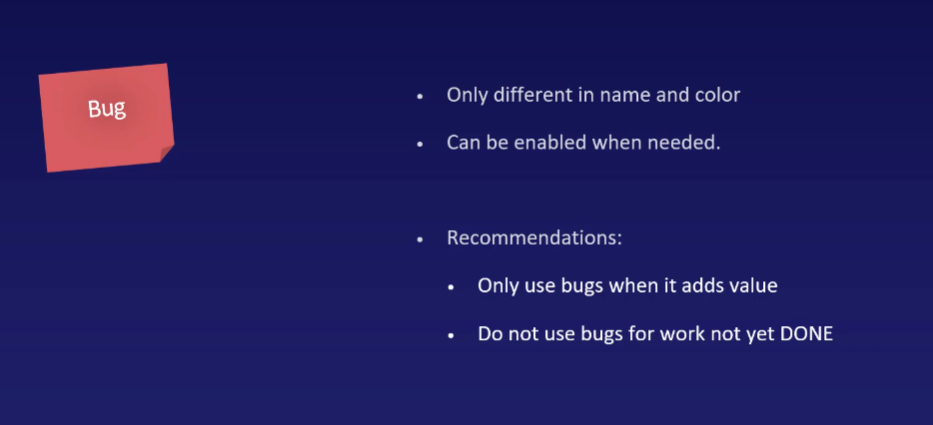
The Basic process does not have a work type item designated for bugs.

Other processes (Agile, Scrum, and CMMI) bugs are a work type item.

Recommandations to use:::

To keep track of the number of bugs found and created or fixed, over a period of time, or in an iteration. Or maybe you are using bugs to track issues reported by users, detect regressions, and build release nodes.

I would recommend you to use the bug type only for bugs that are discovered in work that was already declared done.



Queries::

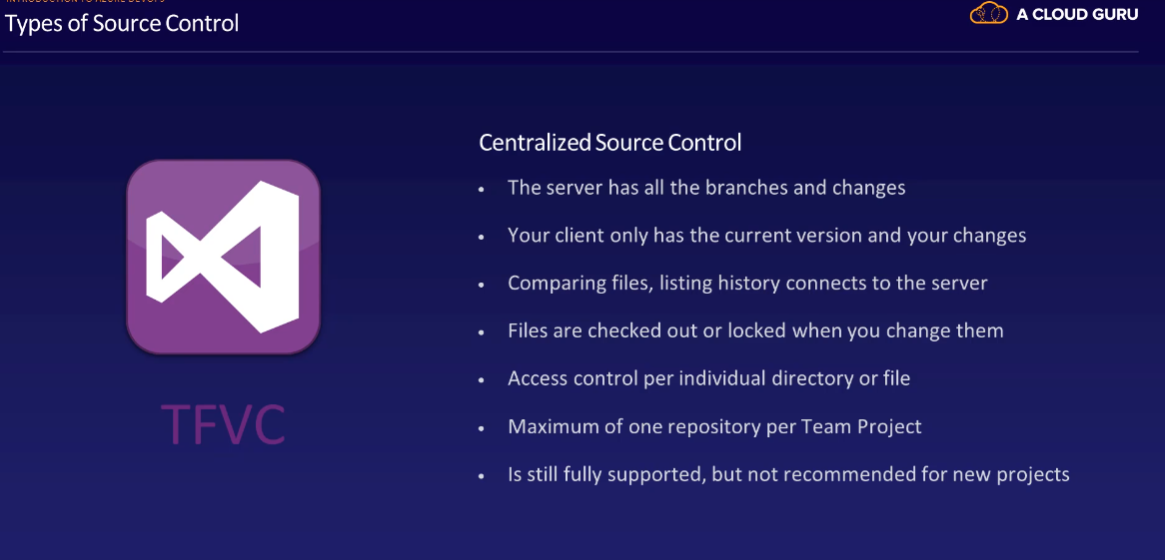
Queries are the way to generate custom lists of work items, which we might need when looking for something.

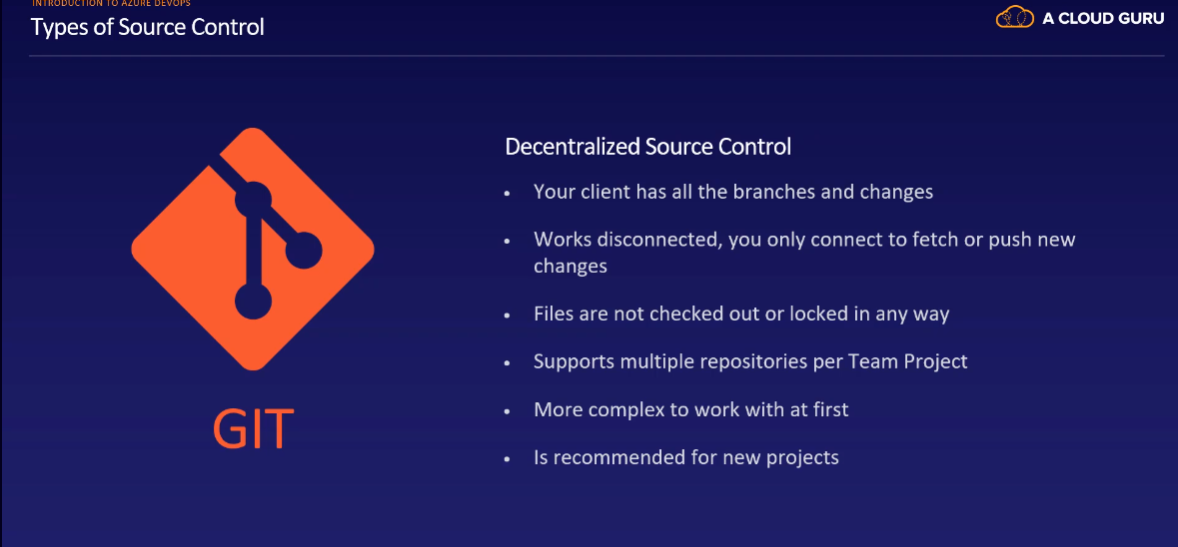
We can create our own queries, so like we can save them.

So for that Query we will get one url (that is valid upto 3 months)

We can send it through email also.

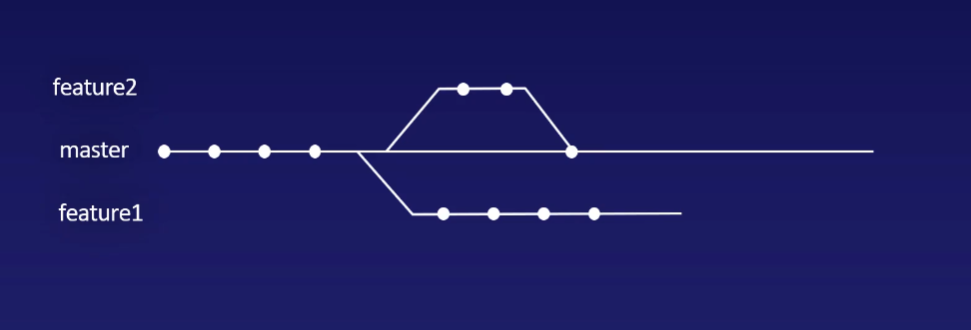
AZURE REPOS

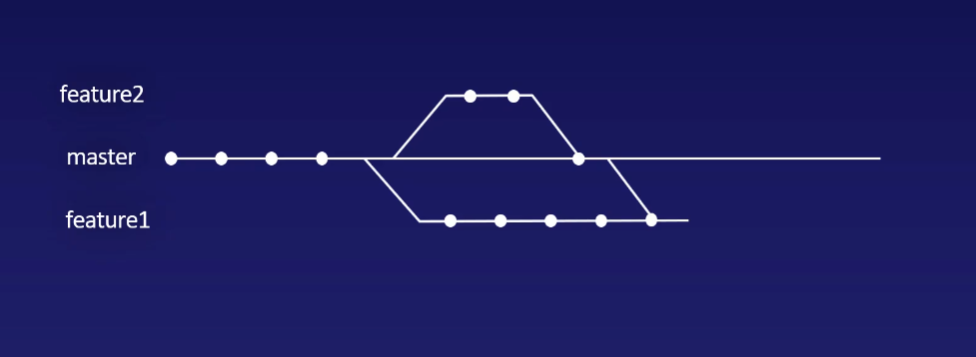


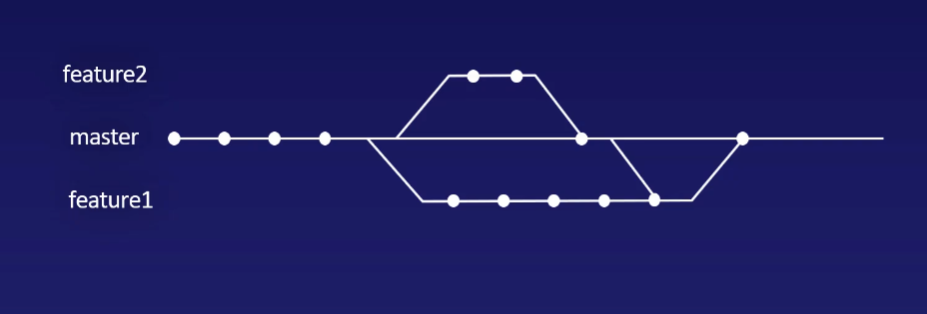


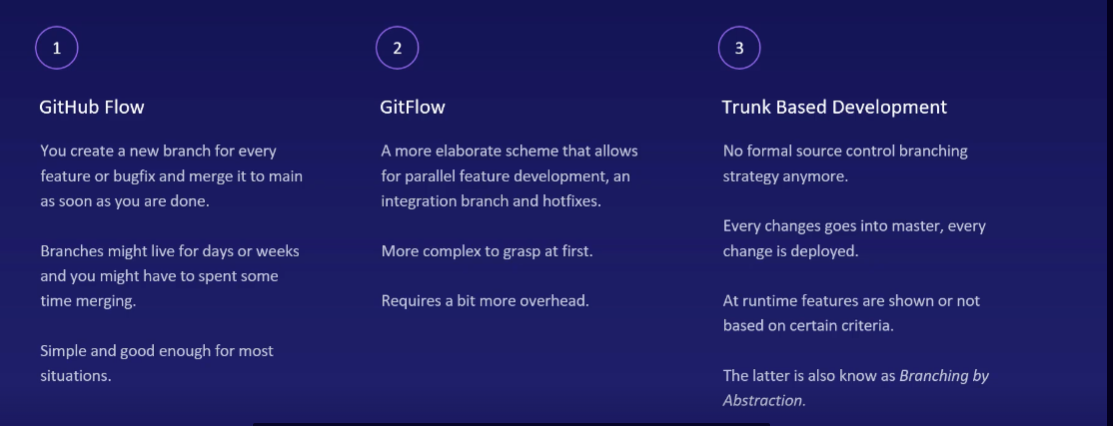
Branching and merging::











GitHub Flow:

GitHub Flow is a simplified and lightweight workflow designed for teams that use GitHub as their version control platform. It is often used for continuous integration and continuous delivery (CI/CD) pipelines. Here are its main characteristics:

**Main Branch:** GitHub Flow typically uses a single main branch (often named "main" or "master") as the primary development branch.

**Feature Branches:** Developers create feature branches for new features or bug fixes. These branches are short-lived and based on the main branch.

**Pull Requests:** Once a feature is complete, a pull request (PR) is created to merge the changes back into the main branch. This allows for code review and automated testing before merging.

**Continuous Integration:** Automated tests and CI/CD pipelines are often used to ensure that code in feature branches is stable before merging.

**Deployment:** Changes are frequently deployed to production, and this workflow promotes a high release frequency.

Git Flow:

Git Flow is a branching model created by Vincent Driessen. It's a more complex and structured workflow, designed for larger projects with scheduled releases. Key points include:

**Main Branches:** Git Flow uses two primary branches: "master" for production-ready code and "develop" for ongoing development.

**Feature Branches:** Developers create feature branches for individual features or bug fixes. These branches are based on "develop."

**Release Branches:** Before a release, a "release branch" is created from "develop" for final testing and bug fixes. Once the release is stable, it's merged into both "master" and "develop."

**Hotfix Branches:** If critical issues are discovered in the production code, a "hotfix branch" is created from "master" to address the problem. The changes are then merged into "master" and "develop."

**Versioned Releases:** Git Flow is well-suited for projects that have versioned releases and need to maintain different versions simultaneously.

Trunk Based Development:

Trunk Based Development is a simpler and more continuous integration-focused approach to development. It encourages frequent integration of code into the main branch (also known as the trunk). Key aspects include:

**Single Main Branch:** Trunk Based Development typically uses a single main branch as the source of truth. Feature branches are discouraged or used very briefly.

**Frequent Commits:** Developers are encouraged to commit and merge their code to the main branch multiple times a day, ensuring that changes are continually integrated and tested.

**Automated Testing:** Extensive automated testing and CI/CD pipelines are crucial for quickly identifying and addressing issues.

**Continuous Deployment:** This workflow is suitable for teams that aim to deploy changes continuously to production.

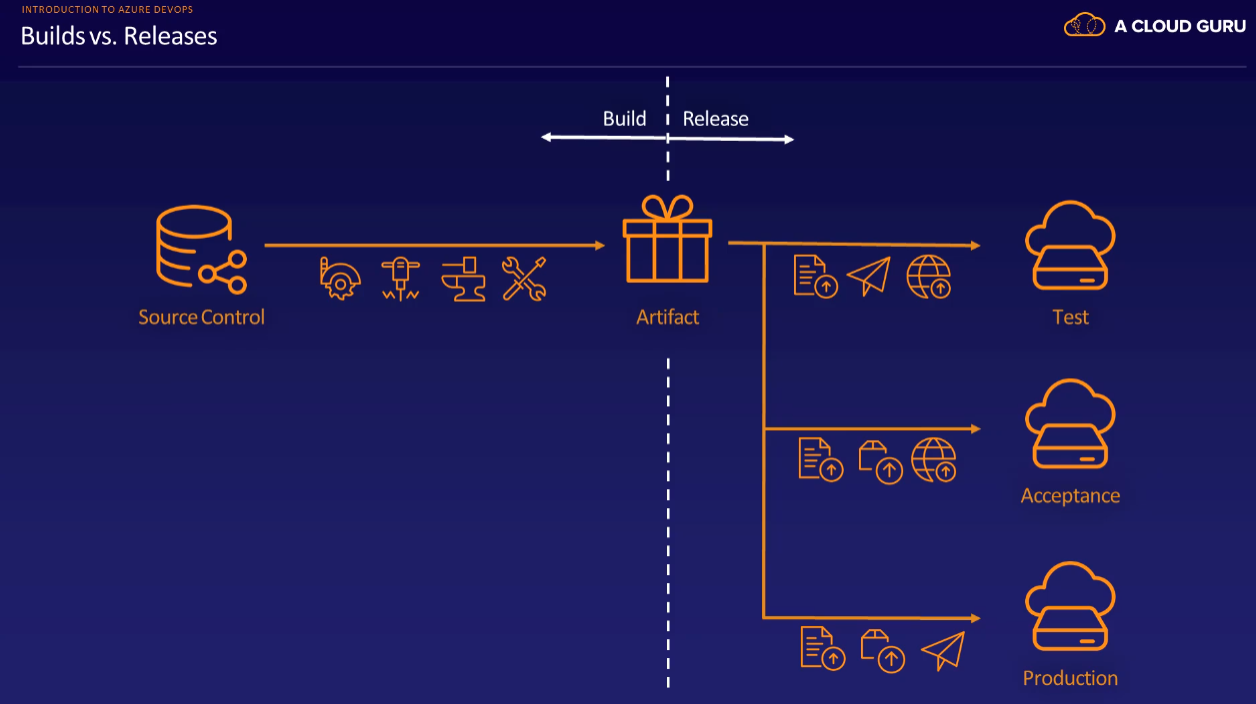
In summary, GitHub Flow is simple and suitable for smaller projects or teams, Git Flow is more structured and suitable for projects with versioned releases, and Trunk Based Development is focused on continuous integration and deployment with a single main branch.

AZURE PIPELINES

Build & Release

Building and Releasing your software both are done using pipelines consisting of a number of automated steps.

The purpose of this is to create an automated repeatable delivery of your software from source control to your environments.

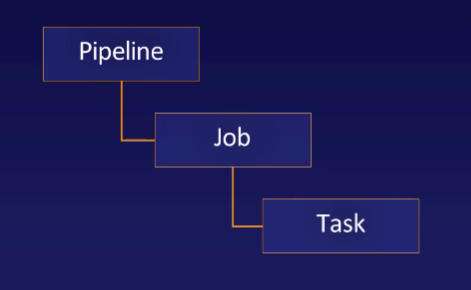


* First we have source control repositories.
* Whenever a new check-in occurs or a scheduled interval, the contents of repositories go through a pipeline of operations called steps, to create what we call an artifact.
* Typical operations includes compiling, testing and packaging an application package and linting and packaging configuration.
* The resulting packages should contain everything that is needed for the deployment of your application.
* Artifact can be reused over time.
* We can now use this package to deploy it to our test environment. Again, we do that using a pipeline consisting of multiple steps.
* We are now focusing on delivery, also called releasing.
* After a successful delivery to our first environments, you can now also deploy to all our environments, like acceptance and production.

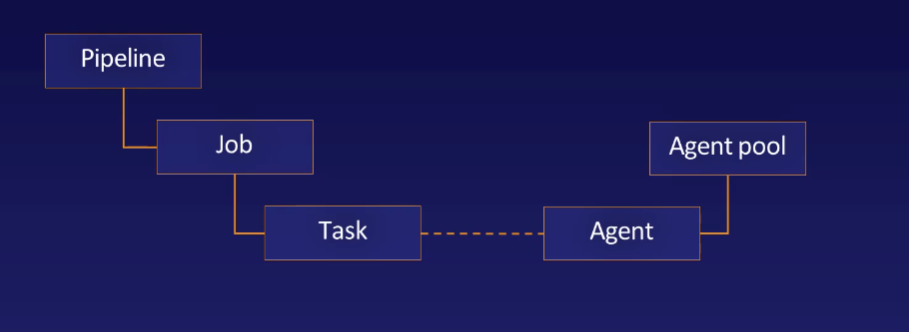
This means that you only deploy to acceptance after deploying to test and only to production after deploying to acceptance.

Agent & Agent pool

* You learned that both build and release pipelines are made up of one or more tasks.
* Tasks can be grouped in one or more jobs.
* This means that technically a pipeline has one or more jobs and a job has one or more tasks.

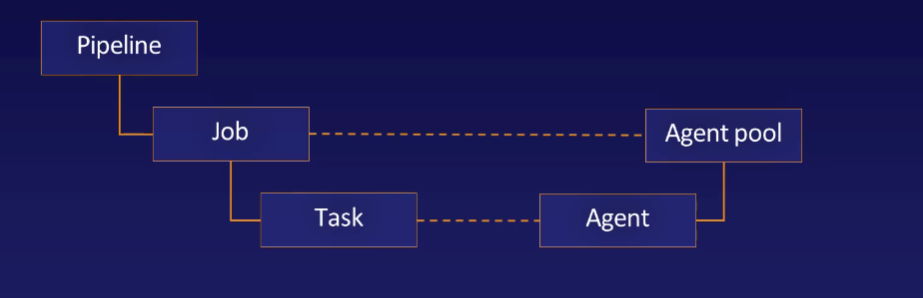


* Now these tasks do not run within an Azure DevOps project, or organization itself.
* They are actually run on what are called agents.
* Agents are run times often on a virtual machine, or any container, that are responsible for actually executing the tasks that you specified.
* "agents" refer to the computing resources or nodes that execute the tasks and jobs within the pipeline.
* These agents do not stand on their own, but are grouped in what are called agent pools.
* An agent pool is nothing more than a grouping with a name and a list of agents in it.
* Every agent is always in one agent pool, and agents in a pool can be temporarily taken offline for maintenance.

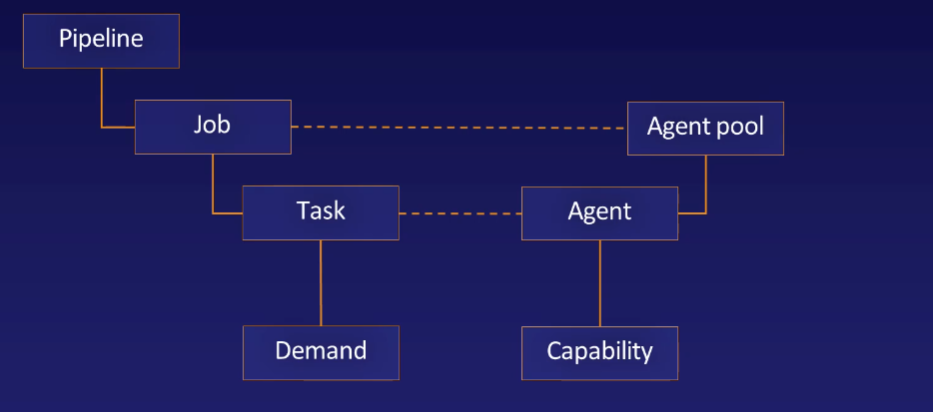


Which agent pool is used for a task is decided in 2ways::

* You can select an agent pool from which an agent should be obtained, for every job in your pipeline. This ensures that when a single agent is offline, and the pool has more than one agent, your job can still be executed.



* Demand and capabilities
* Every task that you can use in a pipeline specifies one or more demands that it has. A demand is a key value pair.
* example : MS Build equals 15.0
* This means that the task can only run on an agent that has a corresponding capability that specifies MS Build equals 15.0.
* To determine which agent to use, all demands of all the tasks in a job are grouped



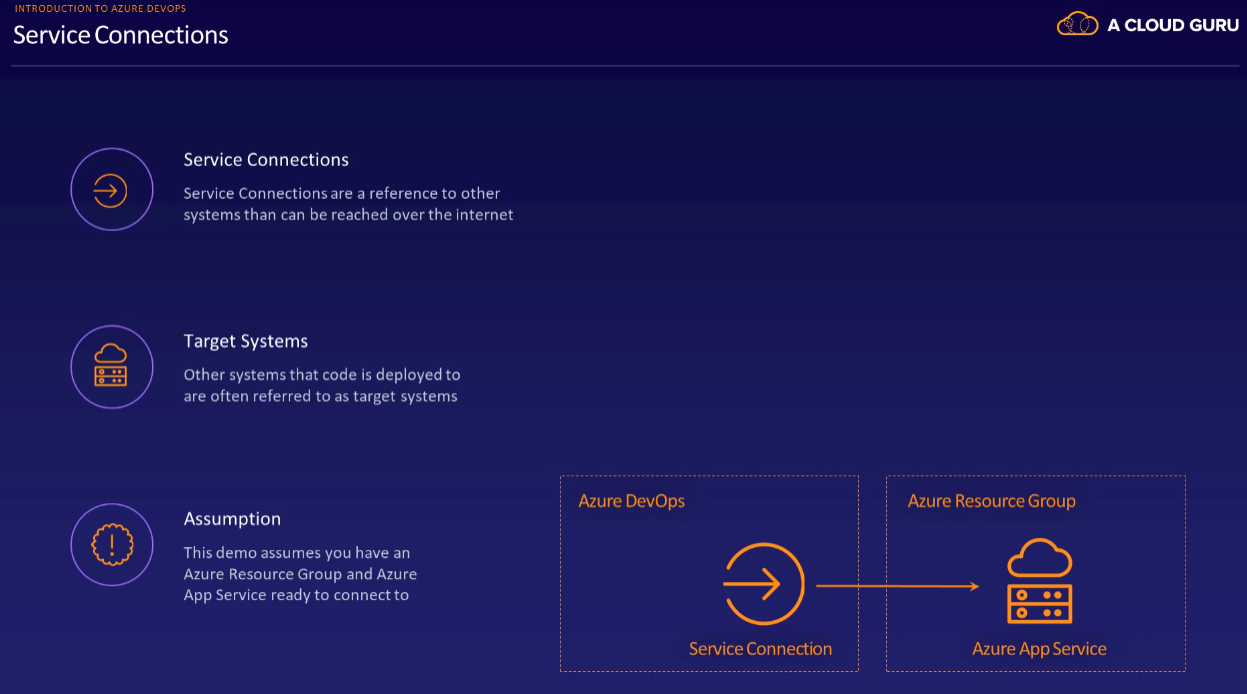
* We should know that agent pools are created at the level of the Azure DevOps organization, from here on they are to individual projects.



Agent pools are 2 Types::

1. Hosted pools - managed for you by Microsoft.
2. Self hosted pools- you create and manage yourself.

A service connection in Azure DevOps is a defined connection that allows Azure DevOps to communicate with an external service.



Library

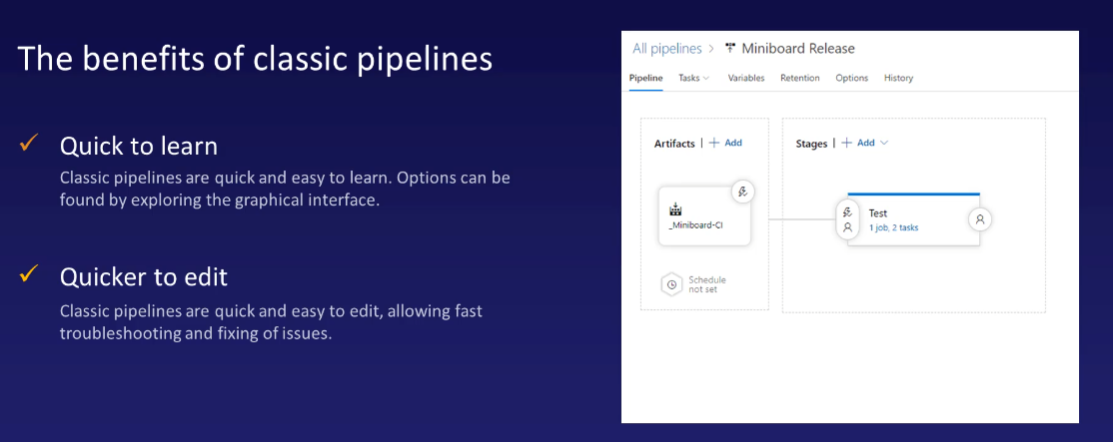
You can create groups of variables that you want to use in more than one pipeline without duplicating them. You can also store files here securely also for use in pipelines. Azure DevOps will then ensure that they are not lingering about, after your pipeline completes.

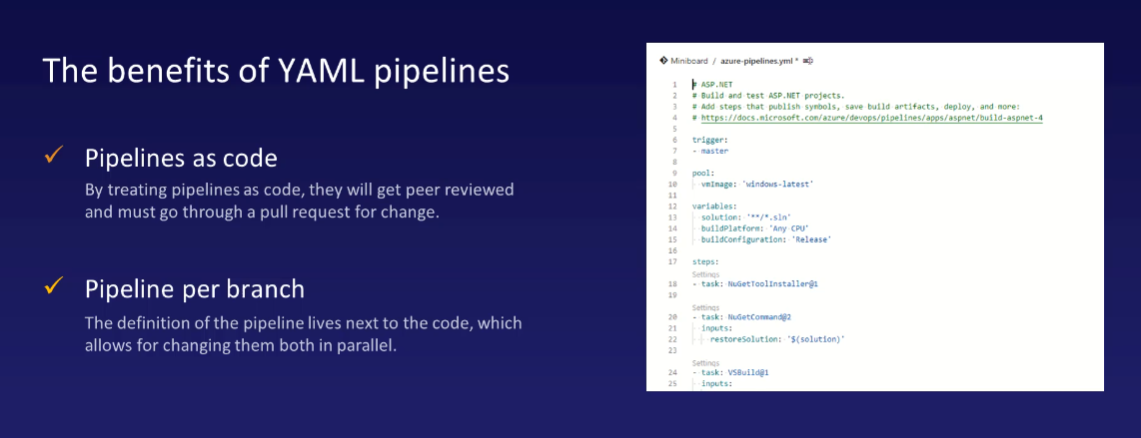
Deployment Groups

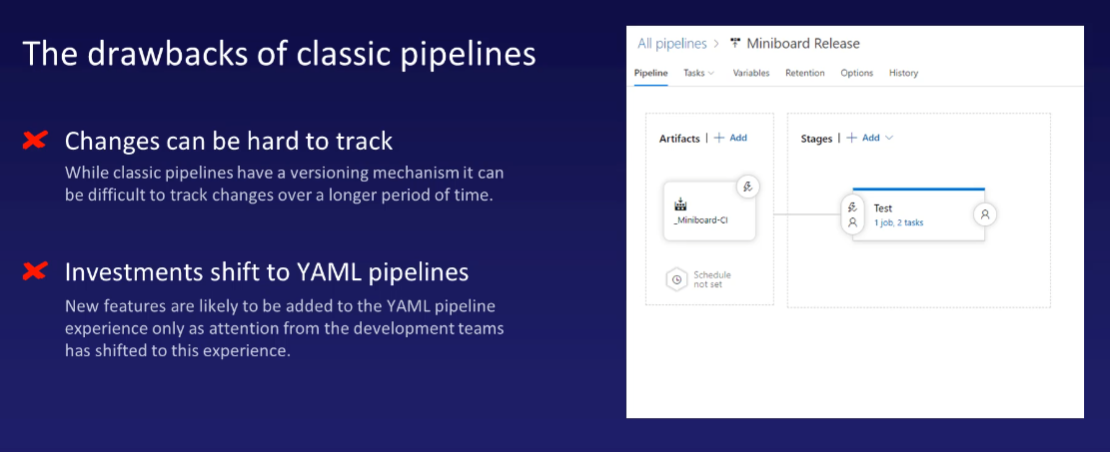
Regular agent job will run on one of the agents in the agent pool, while a task in a deployment group job will run on all the agents in the deployment group. Deployment groups are thus often used when you want to deploy your code to a number of target machines in parallel or to machines that are not accessible over the internet.

An example is when you have a number of virtual machines on premises that you should deploy to.





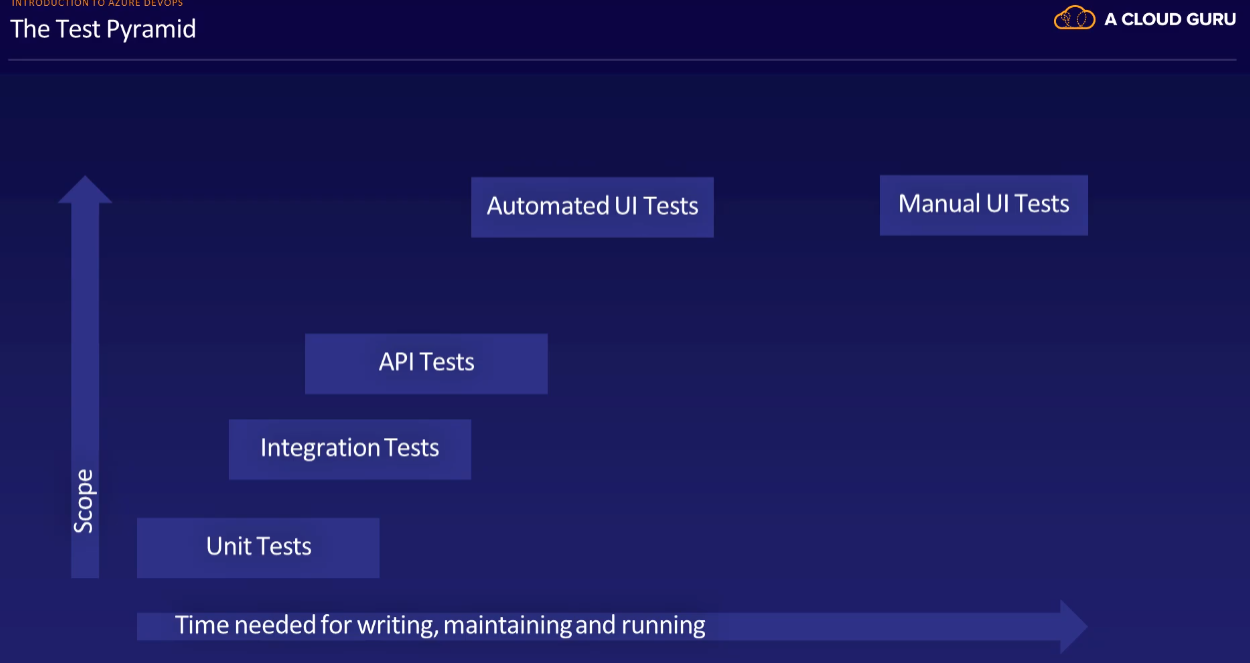




|  |  |
| --- | --- |
| Classic Editor | YMAL file |
| Quicker to edit classic pipelines through to user interface | Editing a YAML pipeline, always enforced editing, text and creating and completing a pull request. |
| During pipeline running also we can edit the pipeline | YAML pipelines can’t be edited, once started  You will always have to restart the whole pipeline from scratch. |
| This doesn’t have that option | YAML files we have source control option (we can see that changes done to YAML file) |
|  |  |

AZURE TEST PLANS







This will take more time

This is process is good

You should try to avoid to creating an ice cream cone of tests.

If you have many, many Manual and Automated UI tests in your project, and just a few Unit and Integration tests, you will have to spend a lot of time on writing and updating these tests.

Also, they will take a long time to run, giving your feedback on the quality of your work only a few hours after you're done preparing to change.

Instead, you should try to create a pyramid of tests having many many unit tests that can quickly provide you with feedback on the quality of your work.

Often within minutes. Of course, they will not catch everything. But well over 90% of all issues you should be able to catch with Unit Tests, then layer other types of tests on top of that only for the things that you cannot cover with Unit Tests. This way, the number of API or Unit Tests you need, should be fairly low.

AZURE ARTIFACTS

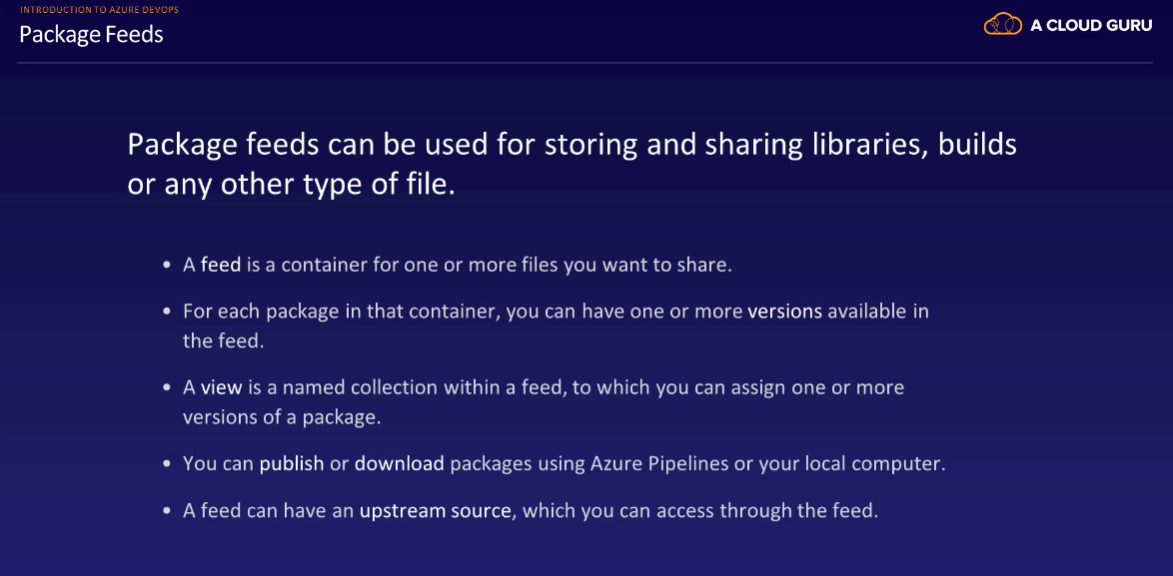
Azure artifacts are the location where you can store your reusable packages, libraries, or build artifacts that you want to consume from other systems than Azure pipelines.

Package Feeds::

Package feeds can be used for storing and sharing libraries, builds or any other type of file.

* Feed is a container for one or more files, you might want to share with other developers, teams or applications.
* Files within a feed are referred to as a package.
* You can have multiple versions of the same package available in your feed.
* A view is a named collection within a single feed.
* You can assign package versions to this view. Or in other words, you can tag a package version as belonging to a view.
* A single package version can be in more than one view.
* With views you can create subsets of packages and grant access to them separately as well.
* You can publish or download packages using Azure Pipelines or your local computer.
* Feeds can have an upstream source.
* This means that you can access other sources of packages through your own Azure feed.

An example is that you can add NuGet.org as an upstream source to a new Git feed. If you do this, your feed will buffer al the packages that you use from NuGet. If now ever somebody removes a package, your application build will not be impacted, since you are buffering this package.



A common type of artifact feed is the NuGet feed.

When developing applications in C-Sharp, this is most likely the main source of libraries for you.



