**Azure DevOps with CI/CD pipelines**

Before understanding about Azure DevOps lets we distinguish between the Azure (azure portal) and Azure DevOps.

**Azure:**

* This is Microsoft's cloud computing platform.
* It provides a wide range of cloud services, including virtual machines, storage, databases, and many other infrastructure and platform services.
* The Azure portal is the web-based interface for managing these Azure cloud resources.

**Azure DevOps:**

* This is a suite of development services that support the software development lifecycle (SDLC).
* It includes tools for:
  + Version control (Azure Repos)
  + Work tracking (Azure Boards)
  + Continuous integration and continuous delivery (CI/CD) (Azure Pipelines)
  + Testing (Azure Test Plans)
  + Package management (Azure Artifacts)
* While Azure DevOps interacts with Azure resources, it is primarily focused on the development and deployment process

**Therefore:**

* They are not the same. Azure is a cloud platform, and Azure DevOps is a set of development tools.
* While they are separate, they often work together. For example, Azure DevOps pipelines can be used to deploy applications to Azure resources.
* It is possible to access and manage aspects of azure DevOps through the azure portal, but azure DevOps has its own web interfaces as well.

The following description outlines how to connect to and work with Azure DevOps:

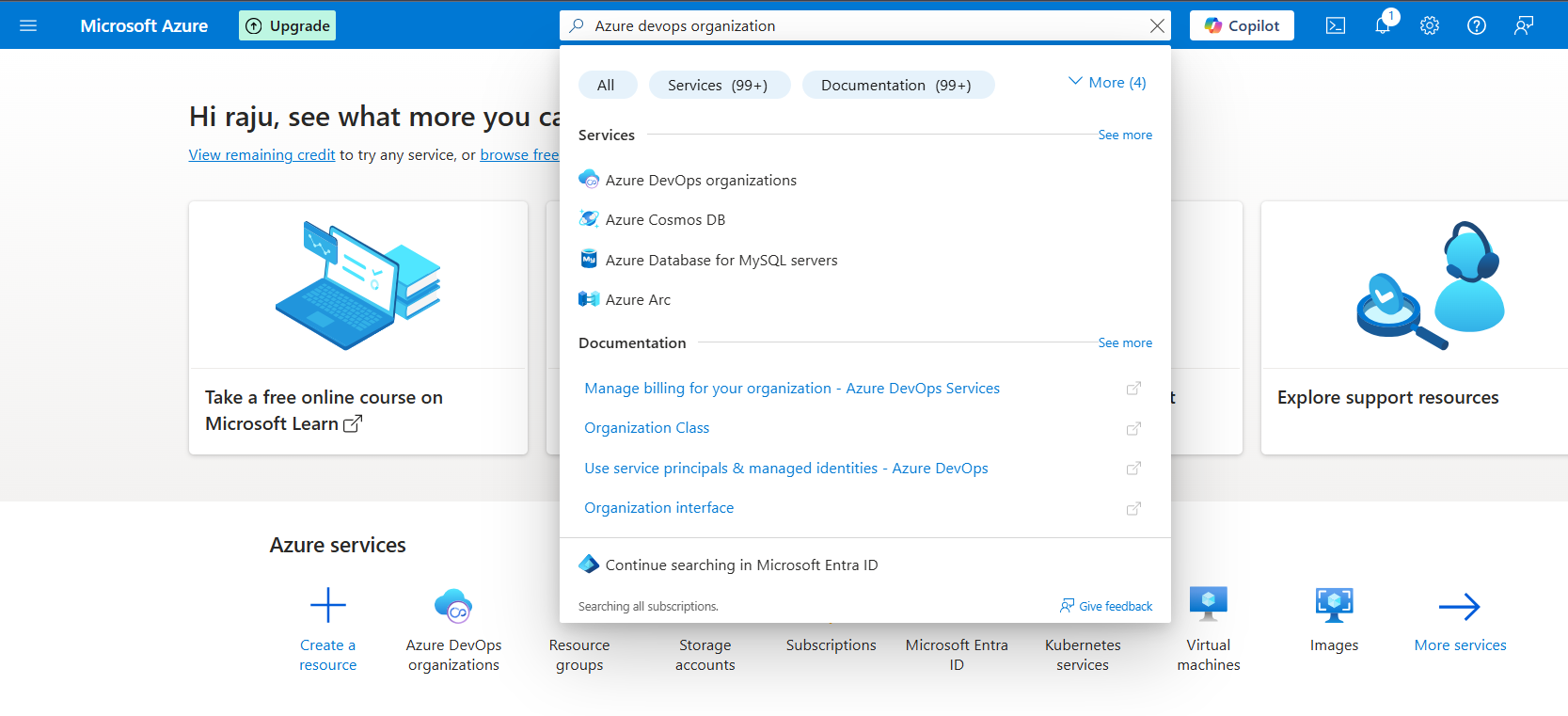


Fig: Azure portal search for the Azure DevOps.

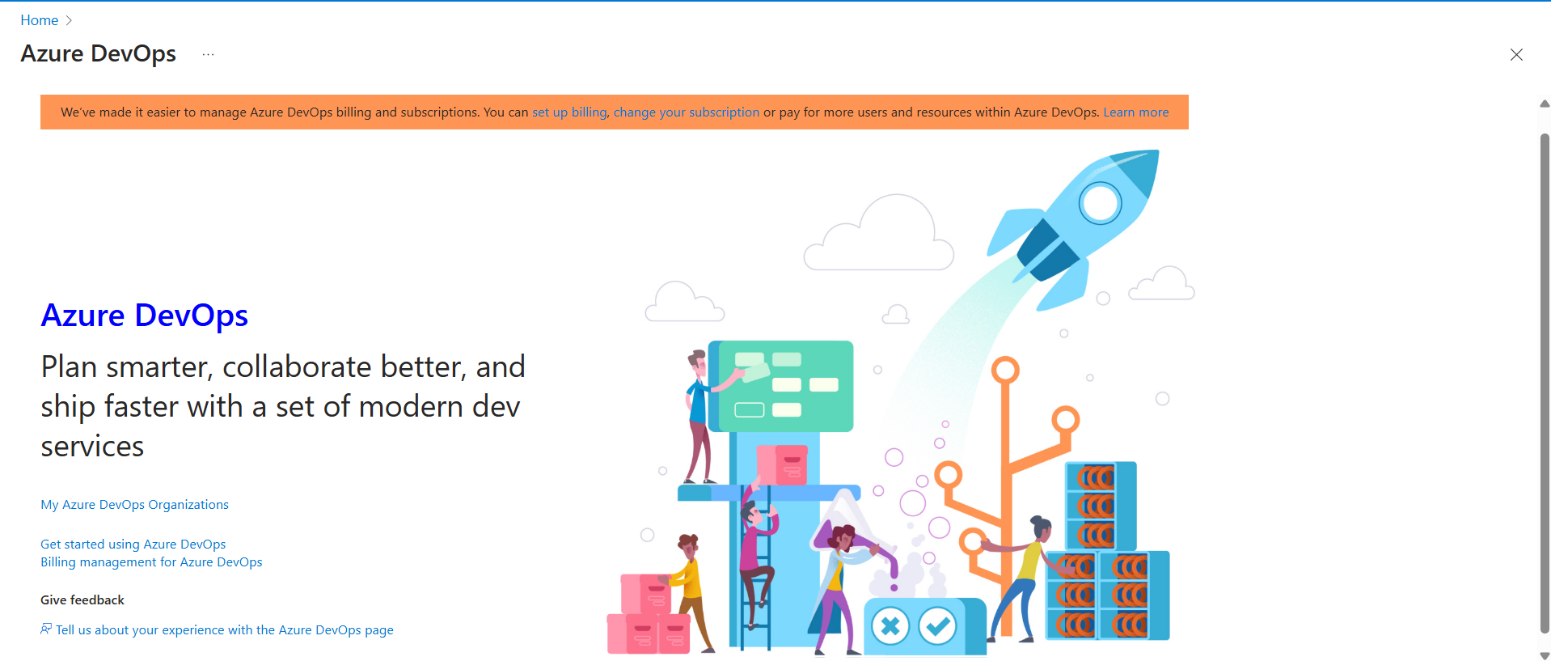


Fig: Azure DevOps GUI

In order to work with the Azure DevOps we have to create an organization and within the organization we can create multiple projects.

**Note:** Azure DevOps is free of cost for 5 users only, if we add more than 5 user then the cost will increase.

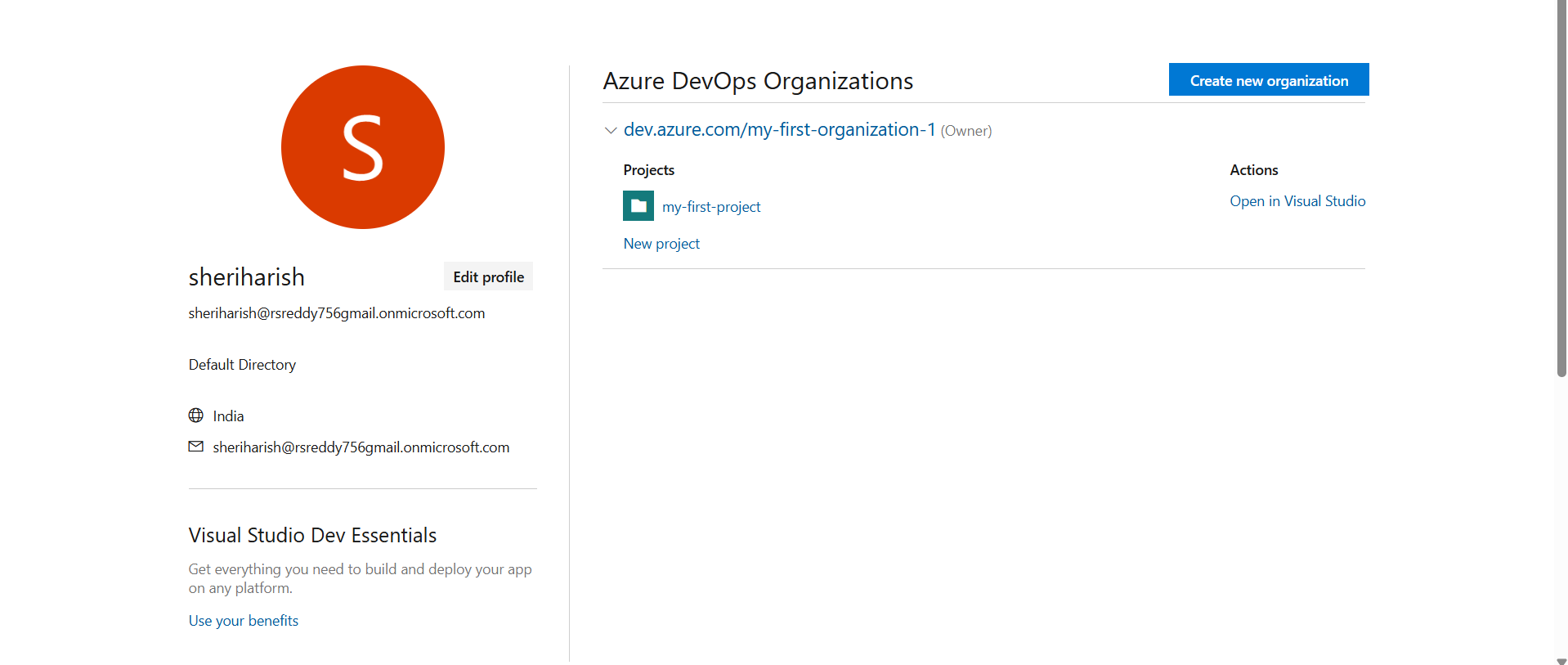


Fig: Creation of Organization.

In above figure the “**dev.azure.com/my-first-organization-1**” is an organizations which are created earlier. And “**my-first-project**” is a project within the organization “**my-first-organization-1**”

If you click on the “create new organization” it will create another new one.

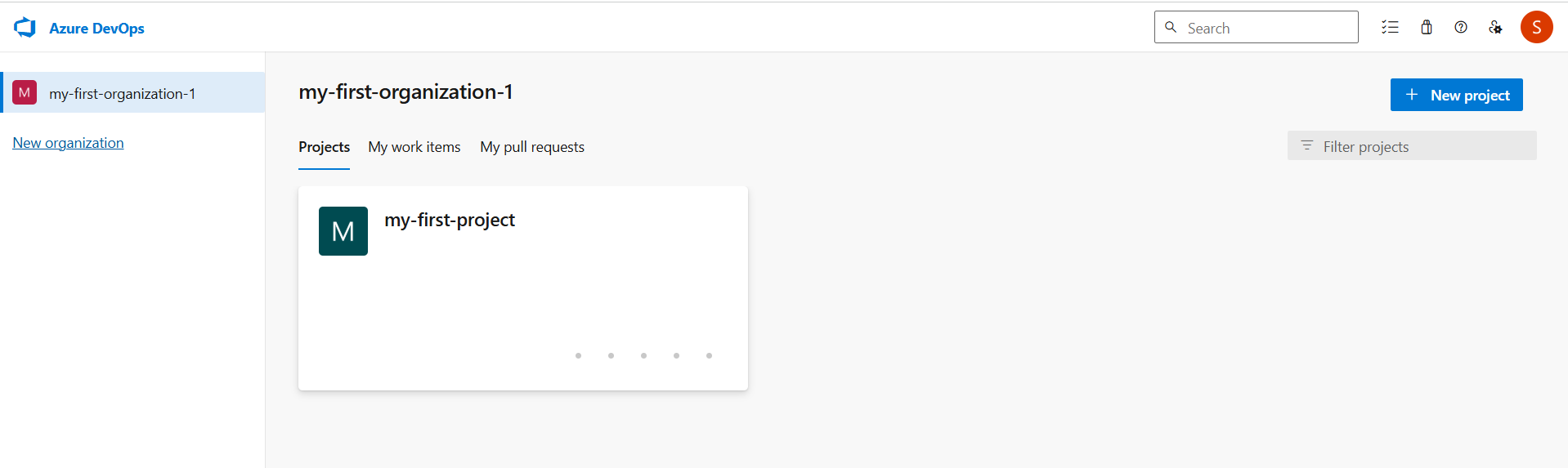


Fig: Creation of New Project within the Organization (my-first-organization).

Before working with the Azure DevOps pipelines we have to rise a **parallelism request**.

A **Parallelism Request Form** is a formal process in Azure DevOps (specifically for **Azure Pipelines**) to request additional **parallel jobs** (also called **parallelism**) for running multiple pipeline jobs or stages simultaneously. This is essential for speeding up CI/CD workflows, especially in large teams or complex projects.

By default, Azure DevOps provides a limited number of **free parallel jobs** (e.g., 1 free job for Microsoft-hosted agents). To scale CI/CD pipelines, teams request additional parallelism to:

* **Speed up Builds & Deployments:** Run multiple jobs/stages **concurrently** (e.g., test suites, deployments to different environments).
* **Optimize Resource Usage:** Avoid bottlenecks where jobs queue up due to limited parallelism.

#### ****Support Large Teams:**** Multiple developers triggering pipelines simultaneously won’t block each other

#### **For Parallelism request form search in any web-**browser as “Parallelism request form”.

#### Link: https://forms.office.com/pages/responsepage.aspx?id=v4j5cvGGr0GRqy180BHbR5zsR558741CrNi6q8iTpANURUhKMVA3WE4wMFhHRExTVlpET1BEMlZSTCQlQCN0PWcu&route=shorturl

#### 

Fig: Parallelism request form.

**Note1:** We can rise the parallelism request for each organization not for entire Azure DevOps.

**Note2:** It takes 4-5 business days to process this parallelism request form.

**Note3:** This parallelism request is raised only when we work with the Microsoft agent machine only. It does not require when we work with the Self-hosted agent machine.

**Note4:**

* In real world we only work with the self-hosted agent machine, but not with Microsoft hosted agent machine while working with the Languages like JAVA, Python and .net so on. So that we can easily take care about all the dependences.
* Microsoft hosted agent machine can be used when we work with the IAC code (terraform) because we no need to take care of dependences, Build engine tools and sop on.

In order work with the Azure DevOps first we have to know about some concepts like CI/CD, Pipelines and Azure repository

**CI/CD**

CI/CD stands for **Continuous Integration** and **Continuous Delivery (or Deployment)**.Is a core practice within DevOps (Azure DevOps) that automates the software development lifecycle (software development, testing, debugging and delivery), aiming to deliver code changes more frequently and reliably.

**CI (continuous Integration):**

* **Definition:** Developers frequently merge code changes into a shared repository (e.g., GitHub, GitLab).
* **Automation:** Each merge triggers an automated **build and test** process to catch bugs early.
* **Goal:** Ensure code changes integrate smoothly without breaking the application.
* **Example:** You push the code to GitHub a CI tool like Jenkins/Azure DevOps actions automatically runs the unit tests, if tests pass, your code is approved for the next steps.

**CD (Continuous Delivery):**

* **Definition:** Automates the **release process** so that code can be deployed to **staging or production** environments at any time.
* Usually requires a manual approval to go live (Delivery), but everything is ready.
* **Goal:** Ensure software is always in a deployable state.
* **Example:** After passing tests, your app is automatically deployed to a staging environment.

**Continuous Deployment:**

* **Definition:** Automatically deploys every validated change to **production** (no manual approval needed).
* **No human (manual) intervention** required after the CI pipeline finishes

**Or**

* Every change that passes the pipeline is **automatically deployed** to production—no human intervention.
* **Automation:** Full pipeline from commit → production.
* **Goal:** Deliver updates to users instantly.

| **Feature** | **Continuous Delivery** | **Continuous Deployment** |
| --- | --- | --- |
| **Definition:** | Code changes are automatically tested and prepared for release. | Every change that passes all tests is automatically deployed to production. |
| **Deploy to production?** | Manual approval needed (**Manual approval** is required before pushing to **production)** | Automatic after tests (**No human intervention** required after the CI pipeline finishes) |
| **Risk of errors?** | Lower (more control) | Slightly higher (if not tested well) |
| **Speed of updates?** | Fast, but controlled | Fastest possible |
| **Human intervention?** | Yes | No |
| **Use case** | Most companies (more control) | High-automation teams, startups |

**Block Diagram:**

Packaging

.war .jar .exe

Unit tests &

Integration tests

Code Build &

Code Compile

Continuous integration (CI)

Azure Repo

(Version control)

Testing/Monitored

Dev-env

Pro-env

UAT-env

Infrastructure setup

Dev-env

Pro-env

UAT-env

Deploying to Production

Developer

Continuous delivery/deploy (CD)

From above block diagram we can say the, Firstly the Developer pushes all the source code files and dependences to the azure repository, and then these sources code is converted into single file like (.war, .jar, .war and so on) using the build engine tools like maven (java), Gradle (multiple languages), MSbuild (.NET) at **code Build** stage, and then testing of the code and integration test is done at “**testing**” stage. These all the process is comes under continuous integration (CI).

In continuous delivery/development the infrastructure like VMs app service, storage, and container are build and assigned with the application. These infrastructure is tested and monitored before deploying it into the production environment. So that our application get live.

**Note:** What are the functions (like building, testing unit test, and packaging) in the CI are happed/performed in an Agent Machine, this agent machine may be Microsoft hosted or self hosted machine. If it is a Microsoft hosted machine then this machine will be automatically runs & deletes after completing the tasks like build and testing.

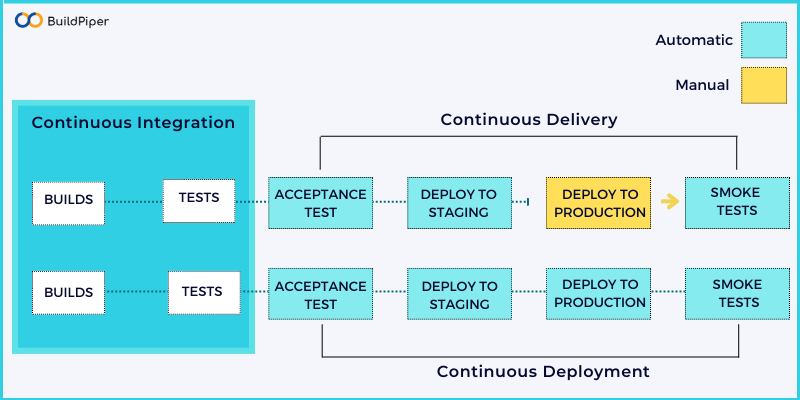


Fig: CI/CD

**Agent Machines:**

While working with the Azure DevOps CI/CD pipelines we require an Agent machine,

An **agent machine** is the **worker** that actually **executes your CI/CD pipeline** jobs.

When you create a pipeline in Azure DevOps (like to build your code, run tests, or deploy to production), the pipeline itself is just a set of instructions. But those instructions need to **run on a machine**. That machine is called an **agent**.

There are two main types of Agent machines:

### 1. **Microsoft-Hosted Agent** (Cloud Agent)

* **Provided by Azure DevOps**.
* **Runs in the cloud.**
* Comes with **pre-installed tools** like .NET, Node.js, Java, etc.
* You **don’t have to maintain it**—Azure takes care of everything.
* You get a **new, clean VM** for each run.

#### 🔹 Use Case:

Perfect for general CI/CD jobs when you don't need anything custom.

### 🔸 2. **Self-Hosted Agent**

* **You provide the machine** (can be on-prem or cloud-based).
* You install the Azure DevOps agent software on it.
* You have full control over the environment:
  + Install custom software.
  + Control security.
  + Avoid pipeline time limits on free-tier cloud agents.

#### 🔹 Use Case:

* When you need specific tools or dependencies not available on Microsoft agents.
* For **faster builds** (no need to spin up a new VM each time).
* To avoid **billing costs** if you're running lots of pipelines (especially on large projects).

**Note:** In **Azure DevOps,** you can absolutely use a **single agent** for both **CI (Continuous Integration)** and **CD (Continuous Deployment/Delivery)** within a single project. Let’s break it down so it’s crystal clear:

## **How It Works:**

* Azure Pipelines use **agents** to run jobs.
* An **agent** can be either:
  + **Microsoft-hosted** (shared, managed by Azure)
  + **Self-hosted** (your own VM or on-prem machine registered with Azure DevOps)
* You can configure your **CI pipeline** and **CD pipeline** to both target the **same agent pool**.

**Pipelines**

A **pipeline** is a **series of automated steps** that take your code from development to production. It's like an assembly line for your software.

Every step—like building, testing, and deploying—is **automated**, and it happens in a specific order

### Typical Stages in a DevOps Pipeline:

1. **Source**
   * Code is pushed to a repository (e.g., GitHub, GitLab).
2. **Build (build engine tools)**
   * Code is compiled or packaged (e.g., Java →.war, .jar, JS → bundle).
3. **Test**
   * Automated tests run (unit tests, integration tests, etc.).
4. **Deploy**
   * Code is deployed to a staging or production environment.
5. **Monitor (optional)**
   * Tools monitor the app in production for errors or performance issues.

### Example Tools Used in Pipelines:

| **Stage** | **Tools Used** |
| --- | --- |
| Source | Git, GitHub, GitLab |
| Build | Jenkins, Azure DevOps, Maven |
| Test | JUnit, Selenium, Postman |
| Deploy | Docker, Kubernetes, Ansible |
| Monitor | Prometheus, Grafana, ELK Stack |

**Relation between the CI/CD and pipelines in DevOps:**

CI/CD and Pipelines are work together to automate the software development and deployment.

**CI/CD🡺**is a concept.

**Pipelines🡺are the** implementation*.*

### CI/CD Defines **What Happens**:

* **CI (Continuous Integration)**: Automatically build and test code when it's committed.
* **CD (Continuous Delivery/Deployment)**: Automatically release or deploy the code after it passes testing.

### **Pipelines Define How It Happens:**

* A pipeline is a **scripted workflow** or set of steps that executes the CI/CD process.
* It's the actual automation tool that performs the CI/CD actions

**Azure Repository:**

It is also a version control system similar as GitHub, here we can maintain the source code in version control so that we can easily track the changes, maintain the source code and collaborate with other developers.

**Azure Repos** is a service within **Azure DevOps** that provides **Git repositories** or **Team Foundation Version Control (TFVC)** for source control of your code.

Azure Repos is essentially a **version control system** hosted on the cloud. It helps you manage your codebase, track changes, and collaborate with other developers.

Azure Repos allows you to track changes to your code over time. This means you can:

* See the history of every file.
* Identify when and why changes were made.
* Revert to previous versions if needed.
* Compare different versions of your code

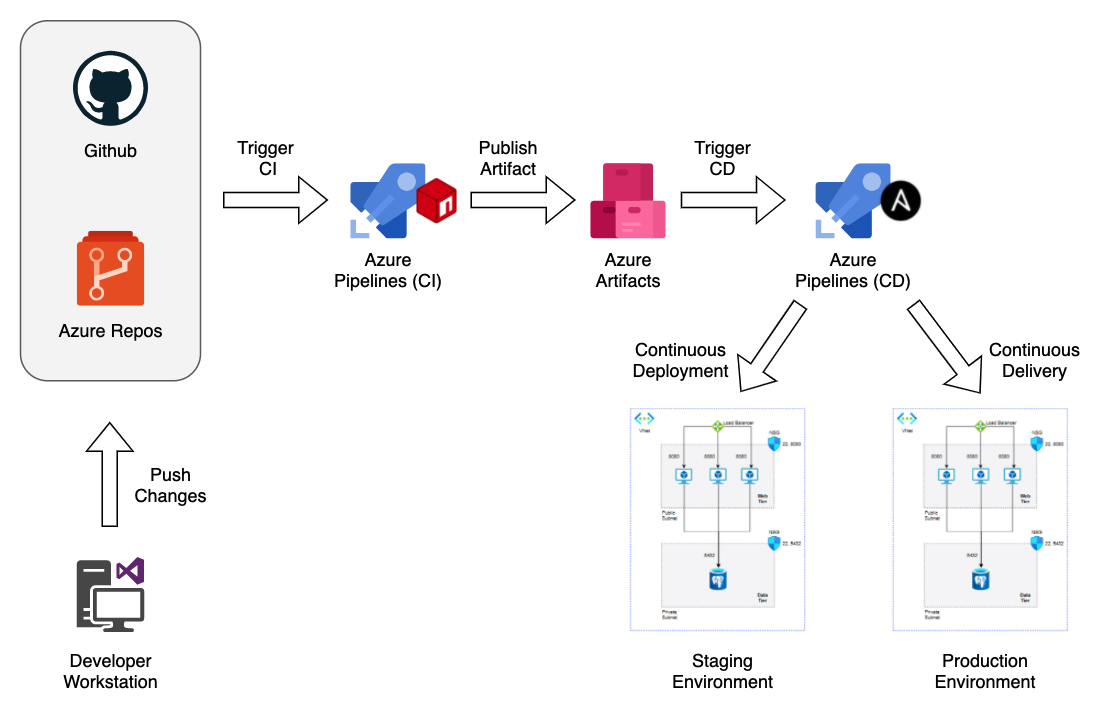


Fig: Visualizing the seamless journey of code to deployment with Azure DevOps!

**Hierarchy or work flow of CI/CD pipeline of Azure DevOps:**

The below figure illustrates the internal work flow at CI and CD of the Azure DevOps.

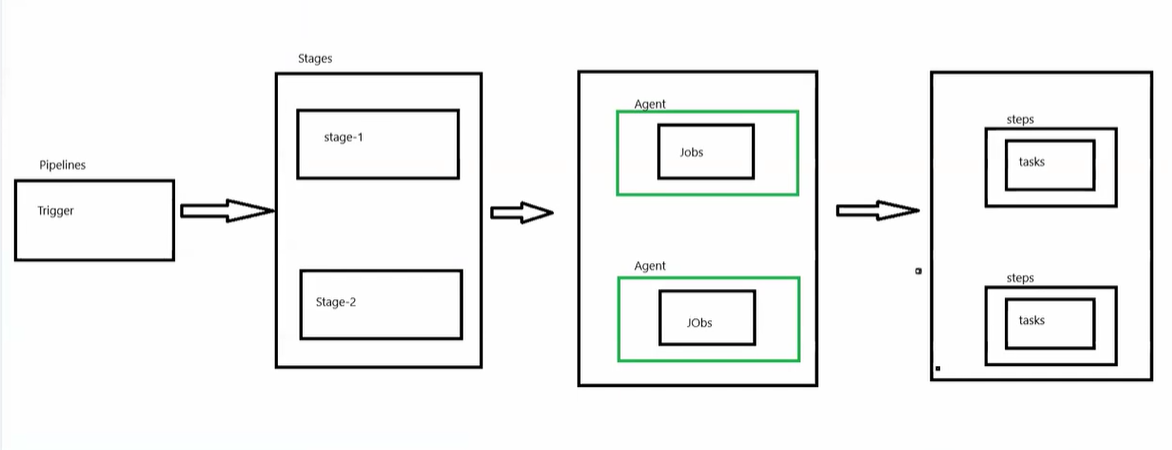


Fig: Azure pipeline work flow.

The diagram you've shared outlines the **hierarchical structure of a CI/CD pipeline**, such as in **Azure DevOps** or similar automation tools. Here's a breakdown of each term from top to bottom:

### 1. **Trigger**

* **What it is:** The event that starts the pipeline.
* **Examples:**
  + A new commit pushed to a repository.
  + A pull request created or updated.
  + A scheduled time (cron jobs).
  + Manual trigger by a user.

### 2. **Stages**

* **What it is:** A logical boundary that groups jobs. Stages often represent **phases** of a pipeline (like Build, Test, Deploy).
* **Examples:**
  + Build stage: Compile code, install dependencies.
  + Test stage: Run unit/integration tests.
  + Deploy stage: Deploy application to an environment.

### 3. **Job**

* **What it is:** A collection of steps that run together on the **same agent** (worker).
* **Features:**
  + Jobs can run sequentially or in parallel.
  + Jobs have their own execution environments.

### 4. **Steps**

* **What it is:** Individual tasks or scripts that run in a job.
* **Types:**
  + Script steps: Shell/PowerShell commands.
  + Task steps: Predefined tasks (like publishing artifacts).

### **5**. **Tasks**

* **What it is:** The smallest unit of work — a specific operation like building code, copying files, or deploying.
* **Examples:**
  + DotNetCoreCLI@2: .NET build task.
  + CopyFiles@2: Copies files to staging.
  + AzureWebApp@1: Deploys to Azure Web App.
  + terraform init, plane, valiadate and apply

**Quick Analogy:**   
Think of it like organizing a project:

* **Trigger** is the event that kicks off the project.
* **Stages** are project phases (planning, execution).
* **Jobs** are assigned to teams. (agent job)
* **Steps** are the team’s to-do list.
* **Tasks** are the individual actions to complete the to-do.

## **Note:**

## **Visual Relationship**

* A **task is a type of step**.
* All tasks are steps.
* Not all steps are tasks.