

# Azure Devops 4.0

Wednesday, February 4, 2026 4:12 PM

## Project with it's own pipeline vs microservice

Many teams misunderstand "*a project with its own pipeline*" as equal to "*a microservice*". But these are **not the same**.

Great question, Arka — and an important one.

Many teams misunderstand "*a project with its own pipeline*" as equal to "*a microservice*". But these are **not the same**.

Below is a clear, practical breakdown of **how modules, shared libraries, front-end apps, backend APIs, utility projects, infrastructure repos, and true microservices differ** — in architecture, deployment, ownership, and purpose.

## How They Differ from Microservices

Here's a concise comparison:

### 1. Modules within a Monolith

#### Not microservices because:

- Cannot be deployed independently
- Share the same runtime (same process, memory, codebase)
- Changes require redeploying the entire application
- Not independently scalable

#### Example

A "Payments" folder inside a big .NET application.

Even if it has its own pipeline for tests, it's still part of a single deployable unit.

### 2. Shared Libraries

#### Not microservices because:

- They are **dependencies**, not services.

- They don't run on their own → no server, no container, no API endpoint.
- They're versioned and published, not deployed.

## Example

A project like Common.Utilities that contains:

- Logging helpers
- Validation helpers
- Encryption utilities

These are packaged (like NuGet/JARs), but **not deployed**.

## 3. Front-end Applications

### ✗ Not microservices because:

- They are user interfaces, not backend services.
- They don't encapsulate domain logic; they present it.
- They don't own data or business capabilities.

## Example

PortalUI from your screenshot is likely:

- Angular/React front-end
- Built and deployed as a static site or SPA

A front-end app ≠ microservice.

## 4. Backend APIs (But Not Microservices Yet)

A backend API *can* be a microservice but is not automatically one.

### ✗ Not necessarily a microservice when:

- It serves multiple business domains
- It's tightly coupled to other services
- It shares a database
- It cannot be deployed independently

## Example

A large ASP.NET Web API with controllers for:

- Orders
- Payments
- Notifications
- User management

This is a **modular monolith**, not microservices.

## 5. Utility Projects

### ✗ Not microservices because:

- They perform offline or background jobs.
- Not accessed by other services via HTTP/GRPC.
- They aren't independent business domains.

## Example

A pipeline for:

- Database migration tasks
- Cron jobs
- PDF generation tool

These are *jobs*, not services.

## 6. Infrastructure Repos

### ✗ Not microservices because:

- They define infrastructure, not business capabilities.
- They produce no running service on their own.

### Example

A Terraform or Azure Bicep pipeline for:

- VMs
- Networks
- Storage
- Kubernetes clusters

Deploys infrastructure, not microservices.

## ⌚ So What *Is* a Microservice Then?

A microservice is specifically:

### ✓ A business-capability-oriented service that:

- Runs independently
- Has its own *database* or persistence
- Is deployable independently
- Can be scaled independently
- Has a clear API boundary
- Is owned by a small team
- Is replaceable/rewriteable without breaking the system

### ✓ Example:

A standalone **Payments Service** that:

- Runs in its own container
- Owns the “payments” domain
- Has its own schema
- Has a REST/GRPC API
- Can be deployed separately
- Can scale differently (e.g., 5 replicas vs 1)

That is a true microservice.

## ⌚ Summary Table

Type	Deployable Alone?	Owns Data?	Independent Scaling?	Independent Lifecycle?	Microservice?
Module in monolith	✗	✗ Shared	✗	✗	✗
Shared Library	✗	✗ None	✗	✓ (version only)	✗
Front-end App	✓	✗	✓	✓	✗
Backend API (monolithic)	✓	✗ Shared	✗	✗	✗
Utility Project	✗	✗	✗	✗	✗

Infra Repo	X	X	X	✓	X
Microservice	✓✓	✓ Own	✓	✓	✓✓

## 👉 Wrap-up

So, having a pipeline simply means a component has CI/CD automation.

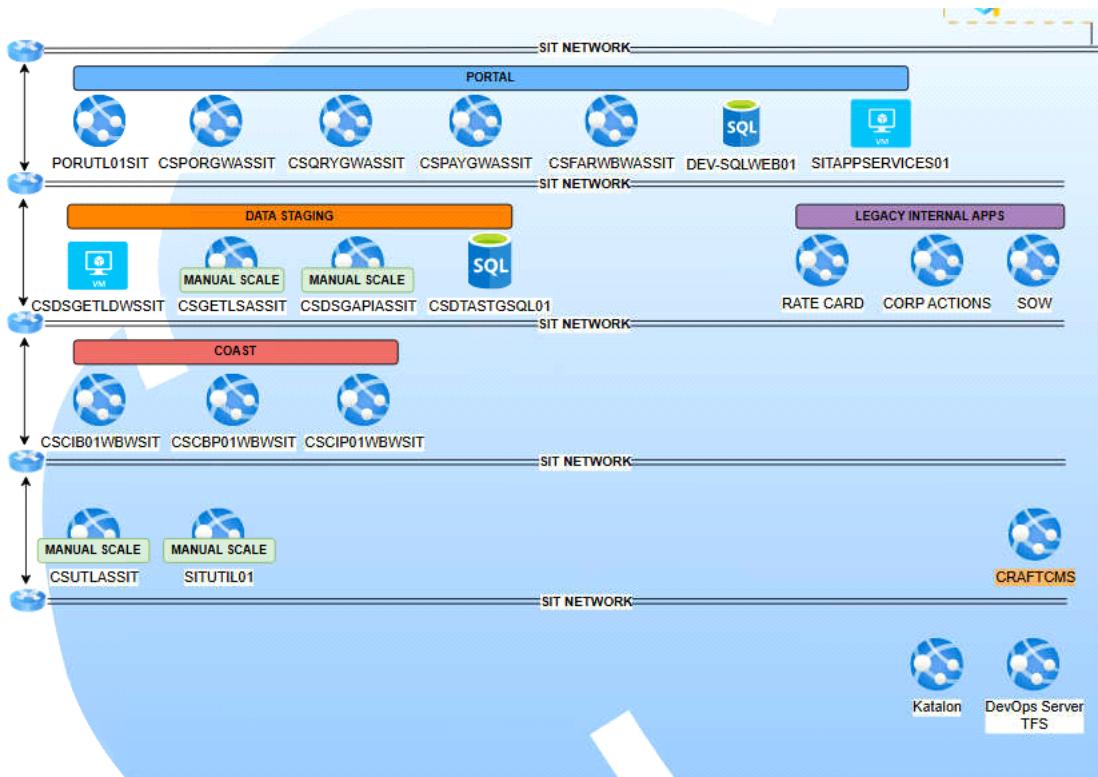
It **does not** automatically mean that component is a microservice.

If you want, you can share:

- the repo structure
- deployment model
- how these services run
- whether they have independent databases

And I can help you identify which ones are true microservices.

Would you like me to evaluate the microservice-ness of each pipeline in your screenshot?



Here the top row are microservices.

Are Microservices server? Lets find out.

Great question — and **this is exactly the point that confuses many people** when they first see cloud architecture diagrams.

Let's clarify it completely and cleanly:

## Is a microservice a server?

**No — a microservice is NOT a server.**

A microservice is just **software**:

a small standalone application, usually exposing REST APIs.

But...

**A microservice must run on something — and that “something” is a compute platform provided by Azure.**

So while it's not a physical server, it *runs on* a compute resource.

## So where do these microservices run in your diagram?

In Azure, microservices can run on **four possible platforms**:

### 1. Azure App Service (most common)

- ✓ A PaaS (Platform-as-a-Service) hosted environment
- ✓ Azure manages the servers internally

✓ You deploy your microservices as "Web Apps"

So:

- No physical server you manage
- No OS patching
- No hardware
- No IIS configuration

Azure handles all of that.

☞ **Most blue circular icons in your diagram = App Services (Web Apps).**

## 2. Azure Kubernetes Service (AKS)

Some enterprises deploy microservices as Docker containers inside Kubernetes clusters.

In that case:

- You have worker nodes (servers), *but*
- They're **Azure-managed VMs**

In your image, I didn't see explicit AKS icons, so SIT likely uses App Services instead.

## 3. Azure Virtual Machines (VMs)

Some workloads (especially old or legacy apps) run on real VMs.

These **are** actual servers, but virtualized and cloud-hosted.

In your diagram:

- The square blue monitor icons (e.g., **SITAPPSERVICES01, DEV-SQLWEB01**) = **VMs**

These are true servers.

## 4. Azure Functions (Serverless)

Some integration logic runs inside **Logic Apps + Functions**

These do not run on fixed servers either — Azure spins up compute automatically as needed.

## ⌚ So what's the reality in your diagram?

### ✓ Blue circular icons

= **Microservices deployed on Azure App Services or containers**

→ They are **not servers**

→ They *run on Azure-managed compute* (you never see the server)

### ✓ Cylinder icons

= **Azure SQL databases**

→ Not servers, fully managed database service

### ✓ Yellow "MANUAL SCALE" icons

= App Services with manually controlled instance count

→ Still not servers; Azure runs them on its infrastructure

### ✓ Blue monitor icon (VM)

= **Actual virtual machine**

→ This *is* a server (virtualized)

## ■ Analogy to make it clear

Thing	Is it a server?	What it actually is
-------	-----------------	---------------------

Microservice	<input checked="" type="checkbox"/> No	A small application
App Service	<input checked="" type="checkbox"/> Not exactly	A platform that <i>runs</i> your app on hidden servers
AKS Pod	<input checked="" type="checkbox"/> No	A container running inside Kubernetes
VM	<input checked="" type="checkbox"/> Yes	An actual server (virtualized)
SQL Database	<input checked="" type="checkbox"/> No	Managed database service

## ★ So the answer to your question:

A microservice is NOT itself a server, but it ALWAYS runs on a server-like compute platform — in Azure's case, usually an App Service or a container host.

You don't see the server, but Azure is running it underneath.

## If you want

I can give you a simple diagram showing how microservices sit on App Services so the idea becomes crystal clear.

Would you like that?

## ✓ Why I said “microservices are not servers”

Because a **microservice is software** — not hardware.

Just like:

- A mobile app isn't a phone
- A website isn't a laptop
- A game isn't the PlayStation

A **microservice isn't a server** — but it *needs* a server to run on.

So the first part is:

**A microservice = NOT a server.**

That is correct.

## ✓ But do microservices *need* a server to run?

**YES. ALWAYS.**

A microservice must run inside *some compute environment*, such as:

- a **VM** (which *is* a server)
- an **AKS node** (a VM — also a server)
- an **Azure App Service** (Azure hides the server from you but it *still runs on servers*)
- an **Azure Container App environment** (again, backed by servers)

So the second part is:

**A microservice ALWAYS runs on a server — physical or virtual — even if you don't see it.**

## 💧 So what actually happened in our earlier messages?

You asked:

*"So this must be a server?"*

I said:

**The microservice itself is not a server.**

It runs *on top of* a server/platform like App Service or AKS.

This is exactly correct.

Then later you asked:

*"AKS must be a server, right?"*

And I clarified:

**AKS is not a server; AKS uses servers (VMs) called nodes.**

So both statements are correct, but describing **different layers**.

Let me show you visually.

## **Layers of Reality (This will make everything clear)**

### **Layer 1 — YOUR CODE**

- Microservice (Java/Spring, .NET, Node.js, etc.)
- This is just an APPLICATION.

### **Layer 2 — PLATFORM that runs your code**

- Azure App Service
- Kubernetes (AKS)
- Azure Functions

These are PLATFORM SERVICES — not servers.

### **Layer 3 — SERVERS that execute everything**

- Azure VMs
- AKS nodes
- App Service underlying compute
- Bare-metal hardware in Microsoft datacenters

These **ARE servers**.

So:

MICROSERVICE (your code)



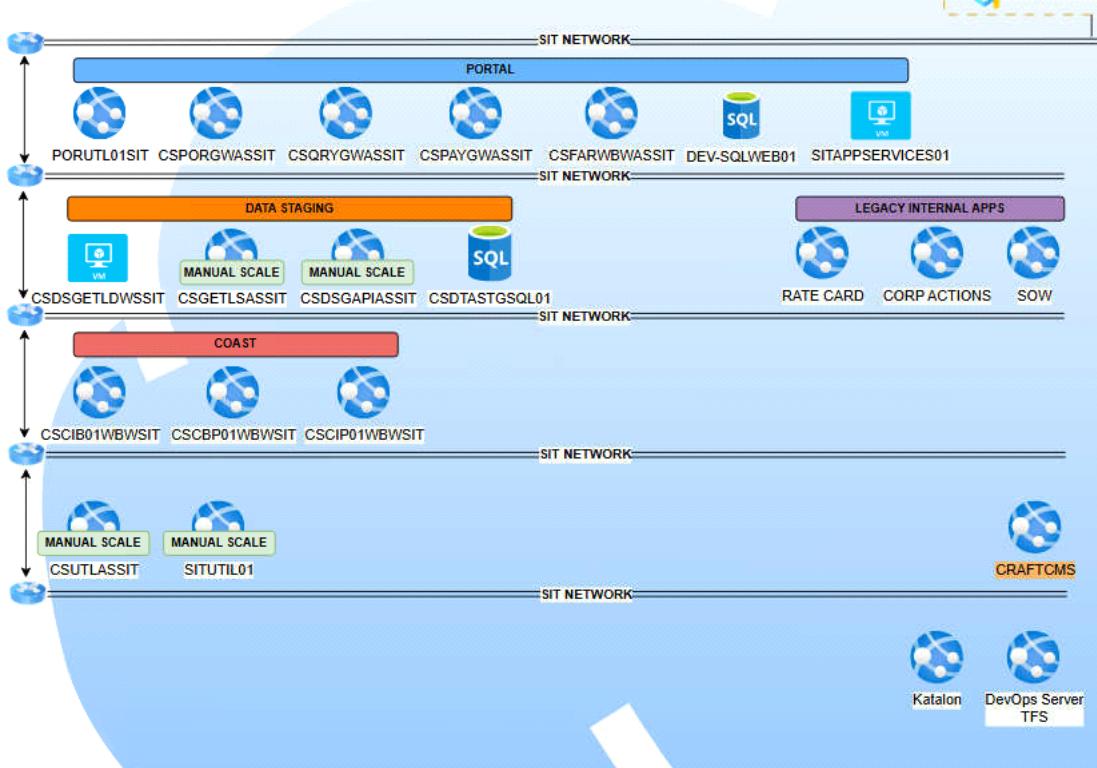
PLATFORM (AKS / App Service)



SERVERS (VMs / physical hardware)

## **★ Final Answer (Simple, Clear, Correct)**

- ✓ Microservices themselves are **NOT servers**.
- ✓ But they **ALWAYS run on servers**.
- ✓ AKS is **NOT a server**.
- ✓ AKS **USES servers (VM nodes) to run your microservices**.
- ✓ Azure App Service is **NOT a server — but internally Azure uses servers to run it**.



Got you, Arka — I'll explain **every single component** in this cropped image, one by one, in a clean and simple way.

Here is your image, re-explained **element-by-element**:

## ■ Legend — What The Icons Mean

Before listing the components:

### ✓ Blue circle with white arrows

→ A microservice / application (App Service or container)

### ✓ Cylinder icon

→ SQL database

### ✓ Computer monitor icon

→ A Virtual Machine (VM)

### ✓ Yellow label "MANUAL SCALE"

→ Manually scaled App Service instance (not auto-scaling)

### ✓ Boxes labelled "Portal", "Data Staging", "Coast"

→ Functional layers grouping related services

## ■ 1. PORTAL LAYER (Top Row)

These are the **frontend and UI-facing applications**.

**These are the items inside the PORTAL bar:**

Component	What It Is	What It Does
PORUTL01SIT	Portal UI service	Customer-facing website / login portal

<b>CSPORGWASSIT</b>	Org-level portal service	Manages organization-level portal features
<b>CSORGYGWASSIT</b>	Org gateway service	API gateway for portal-related calls
<b>CSPAYGWASSIT</b>	Payment gateway UI service	Payment UI/redirection
<b>CSFARMWBASSIT</b>	Farm web service	Likely part of content/website farm
<b>SQL (database)</b>	Azure SQL	Stores portal data
<b>DEV-SQLWEB01</b>	SQL Web server VM	Development/test SQL server
<b>SITAPPSERVICES01</b>	VM hosting app services	Dev/test application hosting VM

☞ All of these are **applications**, not servers (except the VMs shown explicitly).

## 2. DATA STAGING LAYER

This is a middle layer used for **staging, ETL, and intermediate data processing**.

Component	Type	Purpose
<b>CSDSGETLDWSSIT</b>	Microservice	Extract/load data into staging
<b>CSGETLSASSIT</b>	Microservice	Data ingestion/ETL service
<b>CSDGAPIAASSIT</b>	Microservice	API for data staging processes
<b>CSDTSGSQL01</b>	SQL Database	Stores staging/intermediate data
<b>VM icon</b>	Windows/Linux VM	Probably hosting ad-hoc ETL tasks
<b>MANUAL SCALE</b>	App service scaling	Services here need manual instance scaling

☞ These are *applications + one SQL DB + one VM*.

## 3. LEGACY INTERNAL APPS

These purple circular icons represent **legacy enterprise applications** integrated via APIs.

Component	What It Represents
<b>RATE CARD</b>	Legacy rate calculation system
<b>CORP ACTIONS</b>	Corporate actions processing
<b>SOW</b>	Statement of Work / internal workflow system

☞ These are older internal apps, likely hosted on-prem.

## 4. COAST LAYER (Middle Row)

"COAST" is a functional grouping — looks like internal backend services.

Component	What It Is	Purpose
<b>CSCIB01WBWSIT</b>	Backend service	COAST interface service
<b>CSCIP01WBWSIT</b>	Backend service	Policy/COAST processing
<b>CSCIP01WBWSIT</b>	Backend service	Claims/COAST processing

All blue circular icons = **microservices**.

## 5. BOTTOM LAYER (Misc Services)

Component	Type	Purpose
<b>CSUTLASSSIT</b>	Microservice	Likely an internal utility / assist service

<b>SITUTIL01</b>	Microservice	SIT environment utility service
<b>CRAFTCMS</b>	Microservice/App	CMS (Content Management System)
<b>Katalon</b>	QA tool	Automated UI test executor
<b>DevOps Server (TFS)</b>	Server	CI/CD build server for SIT

The only **actual servers here** are:

- The VM icons (blue monitor)
- TFS server

Everything else is a **cloud microservice**.

## ✳️ Putting it together (Simple Explanation)

**This diagram shows five rows:**

1. **Portal row** → all frontend-facing apps
2. **Data staging row** → ETL, data loading, backend utilities
3. **Legacy row** → internal enterprise systems
4. **COAST row** → backend business logic services
5. **Utility row** → miscellaneous internal services + testing tools

They all run on the **SIT (System Integration Testing) network**.

## ⌚ If you want, I can also create:

- ✓ A cleaner version of the diagram (simplified)
  - ✓ A table listing *every* component and its role
  - ✓ A full explanation of how data flows between these components
- Just tell me what you'd like!

**Is “Azure SQL Database” a server?**

Great question, Arka — and you're not wrong to think this through.  
Let's clear this up **once and for all**, in the simplest and most accurate way.

## **Is “Azure SQL Database” a server?**

 **NO — Azure SQL Database is *not* a server.**

 **YES — Azure SQL Database *runs on servers*, but you do NOT manage or see them.**

This is the key distinction.

## **1. Azure SQL Database itself is *not* a server**

Azure SQL Database is a **PaaS (Platform-as-a-Service)** offering.

That means:

- There is **no visible VM**
- You **cannot RDP** into it
- You **cannot install software** on it
- You **cannot patch the OS**
- You **don't manage storage, CPU, RAM, OS, backups**

Microsoft fully manages all underlying servers.

This is why it is **NOT a server**.

You only get:

- A logical database endpoint
- Tables, schemas, queries
- A connection string

Not a machine.

## **2. But Azure SQL runs on *actual servers* behind the scenes**

Every cloud service ultimately runs on physical servers inside Microsoft datacenters.

But Azure hides these from you.

So the accurate statement is:

**Azure SQL Database is not a server, but it runs on servers you do not control.**

This is exactly like:

- Gmail is not "your email server," but it runs on Google's servers
- Netflix is not a "video server," but it runs on servers
- Azure App Service is not a "web server," but it runs on servers

## **3. Compare with SQL Server on VM**

To make it crystal clear:

**SQL Server on an Azure VM**

- ✓ You see the server
- ✓ You maintain the OS
- ✓ You install SQL Server software
- ✓ You patch the machine
- ✓ Fully under your control

## Azure SQL Database (the one in your diagram)

- ✗ No server access
- ✗ No OS
- ✗ No software installs
- ✗ No RDP
- ✓ Microsoft manages everything behind the scenes

This is why Azure SQL is **not** a “server”—it is a **database service**.

## SUPER SIMPLE EXPLANATION

Here's the best way to remember it:

Thing	Is it a server?	Why?
Azure SQL Database	✗ No	It's a <i>service</i> , not a VM; you don't manage hardware/OS
SQL Server on VM	✓ Yes	You manage Windows/Linux + SQL software
AKS Nodes	✓ Yes	They are VMs (servers)
Microservice	✗ No	It's code that runs <i>on servers</i>
Azure App Service	✗ No	Platform-as-a-service; hidden servers underneath

## Analogy (This will click instantly)

- **Azure SQL** = Uber
  - You get the ride
  - You don't own the car
  - You don't maintain the car
  - The car exists but you don't manage it
- **SQL on VM** = Your own car
  - You own it
  - You maintain it
  - You drive it
  - You fix it

Azure SQL is “Uber”, not “your car”.

## Final Verdict

- ✓ Microservices are **NOT servers**
- ✓ AKS is **NOT a server**
- ✓ Azure SQL is **NOT a server**

BUT

- ✓ All of them run *on servers behind the scenes*—you just don't manage those servers.

If you want, I can also give you:

- ◊ A diagram showing **Microservice → Platform → Server**
  - ◊ A simple comparison of **PaaS vs IaaS vs Serverless**
  - ◊ A full breakdown of every icon in your architecture and what is/is not a server
- Just tell me!

DEV-SQLWEB01

- SQL Web server VM
- Development/test SQL server

So this one is a server?

How's this DEV-SQLWEB01 different from CSDTSGSQL01 or Azure SQL?

Great, Arka — now we're getting into the **core difference between three similar-looking things**:

- **DEV-SQLWEB01**
- **CSDTSGSQL01**
- **Azure SQL (PaaS SQL Database)**

Let me explain each one **clearly and simply**, and show how they differ.

## **1. DEV-SQLWEB01 — YES, THIS IS A SERVER**

- ✓ This is a VM (Virtual Machine) in Azure**
- ✓ It is a server — just a virtual one**
- ✓ You manage OS + SQL Server software**

This means:

- It has an operating system (Windows Server)
- SQL Server is installed manually
- You can RDP into it
- You manage patches, updates, firewall, backups, antivirus
- It behaves like a full physical server

Think of it as your own SQL Server machine, but hosted in the cloud.

## 2. CSDTSGSQL01 — This is a SQL DATABASE, NOT a server

This one is a **resource inside Azure SQL** (PaaS) — a database instance.

Not a VM

Not a server

It is a *managed Azure SQL database*

You cannot:

- RDP into it
- See its OS
- Install software
- Manage Windows updates

Azure manages everything underneath.

CSDTSGSQL01 is a **database**, not the machine that hosts it.

## 3. Azure SQL (PaaS) — Also NOT a server

Azure SQL is Microsoft's **fully managed database service**.

Not a server

No OS

Databases run on hidden servers Azure maintains

You only see the SQL endpoint (connection string)

You get:

- A database
- A connection string
- Automatic backups
- Automatic patching
- Auto-failover

But you **never see the VM** or OS.

## SO WHAT'S THE DIFFERENCE BETWEEN THE THREE?

Here is the clearest comparison:

### A. DEV-SQLWEB01 → “SQL on a VM” (IaaS)

Attribute	DEV-SQLWEB01
Is it a server?	<input checked="" type="checkbox"/> YES
Underlying type	Azure VM
You manage OS?	<input checked="" type="checkbox"/> Yes
You install SQL Server?	<input checked="" type="checkbox"/> Yes

You apply patches?	✓ Yes
RDP allowed?	✓ Yes
Flexibility	Highest
Maintenance	Highest

This is like having **your own physical laptop**, but in Azure.

## ▀ B. CSDTSGSQL01 → “Azure SQL Database” (PaaS)

Attribute	CSDTSGSQL01
Is it a server?	✗ NO
Underlying type	Database <i>inside</i> Azure SQL
You manage OS?	✗ No
You install SQL Server?	✗ No
Patches handled by?	✓ Azure
RDP allowed?	✗ No
Flexibility	Medium
Maintenance	Low

This is a **database**, not the whole machine.

## ▀ C. Azure SQL (Service) → Platform as a Service

Attribute	Azure SQL
Is it a server?	✗ NO
Underlying type	Managed service
You manage OS?	✗ No
You install SQL Server?	✗ No
Scalability	Automatic
Maintenance	Zero (Azure handles)

This is like **using Gmail instead of hosting your own mail server**.

## ★ SUPER SIMPLE SUMMARY

Name	What It Is	Server?
DEV-SQLWEB01	Full VM running SQL Server	✓ YES, a real server
CSDTSGSQL01	A single SQL database	✗ NO
Azure SQL	Platform providing SQL databases	✗ NO

## 📌 Ultimate Answer

**DEV-SQLWEB01 is a real server. CSDTSGSQL01 and Azure SQL are NOT servers — they are database services running on Microsoft's servers you don't control.**

If you want, I can also explain:

- Why companies use both VMs and Azure SQL
- When to choose VM SQL vs Azure SQL
- Cost differences
- Performance differences

Just tell me!

Ok, why are we calling it a sql server in the first place? i mean i could also have a minecraft game in that server ?

Great question, Arka — and **YES, you absolutely could install Minecraft on that server** if you wanted.

This is exactly the key difference you're noticing:

- Some things we call "SQL Server"
- But in reality, they are **general-purpose machines** that *can run anything*, including Minecraft.

Let's break down WHY we call it a **SQL Server**, even though it's still "just a server."



## 1. "Server" = a machine

A **server** is just a computer—usually a VM or physical box.

It can run:

- SQL Server
- Minecraft
- IIS
- Node.js
- Python scripts
- Docker containers
- Web apps
- Anything else

So yes — technically:

- ✓ **DEV-SQLWEB01 could run Minecraft**
- ✓ **DEV-SQLWEB01 could run games, web servers, etc.**

## 2. Then why do we call it “SQL Server”?

Because we give servers **names based on their primary role**, not based on their technical capabilities.

☞ If the server's main job is to run SQL Server,  
we *name it* “SQL Server.”

But that doesn't mean it *can't* do anything else.

### **Example in real life:**

You have a laptop named “Work Laptop.”

- You *could* install games
- You *could* watch movies
- You *could* edit videos

But the **name reflects its primary purpose**, not its limitations.

Same idea for servers.

## 3. Naming conventions, not actual restrictions

Let's break down the exact example:

### **DEV-SQLWEB01**

- DEV → Development environment
- SQL → This server is used to host SQL Server
- WEB → Might also be running IIS or web tools
- 01 → This is the first instance

BUT technically:

- ✓ **It is just a Windows Server VM**
- ✓ **You have full OS access**
- ✓ **You can install anything**

## 4. Why Azure SQL (PaaS) is NOT called a “server”

Because you *cannot*:

- install software
- log in to the machine
- run Minecraft
- access the OS

Azure SQL is a **service**, not a general-purpose machine.

So we call it:

- “Azure SQL Database”
- “PaaS SQL”
- “Managed SQL”

NOT a “server.”

## ★ 5. Summary of your question

### ? Why do we call it SQL Server?

→ Because its **intended role** is to run SQL Server software.

### ? Could it run a Minecraft server instead?

→ **YES.**

Because it's just a VM / computer.

### ? How's it different from Azure SQL?

→ Azure SQL is *not a computer*; it's a *service* running on Microsoft's infrastructure.

## ⌚ Super simple analogy

### SQL Server VM = Kitchen you own

You can cook SQL.

You can also cook pizza, pasta, or bake a cake.

You control everything.

### Azure SQL = Restaurant

You can **order food**, but:

- you don't see the kitchen
- you don't cook
- you can't install a fryer
- you can't bake your own pizza

## If you'd like, Arka, I can also explain:

- ◊ When to choose SQL Server on VM vs Azure SQL
- ◊ Costs between them
- ◊ Performance differences
- ◊ Why enterprises still use VMs in 2026

Just tell me!

Since Azure SQL gives you endpoints for your database, why do we need a SQL server by the first place?

Great question, Arka — you've reached the **heart of why BOTH things exist**:

- **Azure SQL (PaaS)**
- **SQL Server on a VM (IaaS)**

If Azure SQL already gives you a database endpoint, **why do we still need SQL Server VMs at all?**

This is one of the most important architectural questions in cloud design.

Let me explain it fully, clearly, and with real-world reasoning.

## ★ Short Answer

We still need SQL Server (on VMs) because:

- ✓ **Azure SQL cannot do everything**
- ✓ **Some applications require full SQL Server features**
- ✓ **Legacy systems cannot run on Azure SQL**
- ✓ **Some workloads require OS-level control**
- ✓ **Some architectures require hosting *multiple components* together**

So PaaS (Azure SQL) is great — but not universal.

Let's go step-by-step.

## ■ 1. Azure SQL is NOT the same as SQL Server

Azure SQL (PaaS):

- Gives you **only a database**
- Hides OS, hardware, SQL Agent jobs, file system
- Does NOT allow you to install additional tools
- Does NOT give you instance-level control
- Does NOT support every SQL Server feature

So Azure SQL is "SQL-like", but **not full SQL Server**.

You get:

- A database
- On a managed service
- With restrictions

You do **not** get:

- The machine
- Full SQL instance
- File system
- Windows-level access
- Full compatibility

## ■ 2. Many enterprise features are NOT

## **supported in Azure SQL**

There are MANY SQL Server features still unsupported or limited in PaaS:

- ✗ SQL Server Agent advanced capabilities**
- ✗ CLR assemblies**
- ✗ Cross-database transactions**
- ✗ Linked servers**
- ✗ Certain types of replication**
- ✗ SSIS/SSRS/SSAS (unless separate services used)**
- ✗ Custom file groups or file system access**
- ✗ SQL Server configurations**
- ✗ Extended stored procedures**
- ✗ xp\_cmdshell**
- ✗ Full compatibility with old T-SQL syntax**

These are essential for many enterprise workloads.

If your application depends on ANY of these →

**You cannot use Azure SQL PaaS.**

## **3. Legacy or vendor applications REQUIRE SQL Server on a machine**

Many older applications:

- expect full SQL Server
- need Agent jobs
- use file-level backups
- rely on special SQL DLLs
- need Windows authentication linked to AD
- use .bak file restores
- need custom tempdb size
- depend on server-level configurations

These apps **simply won't run** on Azure SQL.

So you must use:

→ **SQL Server on a VM (IaaS)**

## **4. You may need OS-level access (which Azure SQL does NOT allow)**

With SQL Server on a VM, you can:

- RDP into the server
- Install custom tools
- Configure max server memory
- Change SQL collation
- Tune disk I/O manually

- Install monitoring agents
- Use Windows domain authentication
- Access file system for ETL

Azure SQL does **not** permit any of these.

## 5. Some environments use SQL Server VM because they run OTHER software too

Example:

Your DEV-SQLWEB01 may host:

- SQL Server
- IIS websites
- SQL management scripts
- Internal utilities
- Scheduled tasks
- Developer test tools
- Integration runtimes

A VM offers the flexibility to be a **multi-purpose machine**.

Azure SQL cannot host anything except databases.

## 6. Cost, control, and testing realities

For **development and SIT environments**, companies often prefer:

- A single SQL VM that hosts multiple dev/test databases
- full control
- lower cost for multiple DBs
- easier restore from .bak files

Azure SQL, when you have many separate databases, can be **more expensive**.

## 7. Some data must stay on-prem or hybrid

Even in cloud-first companies:

- Some databases are still on-prem
- Some data cannot leave the datacenter (regulations)
- Some workloads require hybrid ETL flows

A VM-based SQL Server makes hybrid connectivity easier.

## ★ Final Summary (Simple Version)

Option	What It Is	Pros	Cons
Azure SQL	Managed database service	No maintenance, auto-scale, secure	Limited features, no OS access
SQL Server on Azure VM	Full SQL Server installed on a VM	Total control, all features, flexibility	Must manage patching, backups, OS

## So why do we need SQL Server VMs when Azure SQL exists?

Because **Azure SQL is not a full SQL Server**, and many enterprise apps require things Azure SQL cannot provide.

## If you want, I can also explain:

- ◊ When to choose Azure SQL vs SQL Server VM
- ◊ Performance comparison
- ◊ Cost comparison
- ◊ Why modern architectures prefer PaaS (when possible)

Just tell me and I'll continue!

## ★ So why not use Azure SQL instead of a Test SQL VM?

Because in TEST environments, teams often need:

- OS access (to run tools/scripts)
- File system access
- Custom SQL Server setup
- SQL Agent jobs
- Debugging tools
- Import/export of .bak files
- Running other services alongside SQL

Azure SQL (PaaS) DOES NOT ALLOW any of these.

This is why test environments commonly use SQL Server on **a VM**.

No worries, Arka — this is a **core concept** in backend systems, and once you get it, everything else starts to make sense.

Let me explain **ETL**, how it works, and how a **Test SQL Server** is used to validate **ETL, APIs, and**

**microservices** in a simple, beginner-friendly way.

## ★ What is ETL? (Super Simple Explanation)

**ETL = Extract → Transform → Load**

It is a **data pipeline process** used to move data from one place to another.

✓ **Extract = Take data FROM somewhere**

✓ **Transform = Clean, modify, merge, compute**

✓ **Load = Put data INTO the target database**

**Think of ETL like:**

Taking raw vegetables → cutting/cooking → serving as a dish.

## ⌚ Real-world example of ETL

Imagine your app needs customer data:

1. **Extract:** Read records from a CSV file or an on-prem system
2. **Transform:** Clean bad data, remove duplicates, correct formats
3. **Load:** Insert/update rows into SQL tables in the Test SQL Server

That whole pipeline = **ETL**.

## ▀ Where do ETL processes run?

They can run on:

- Azure Data Factory (ADF)
- SQL scripts
- Logic Apps
- Python scripts
- SSIS packages
- Batch jobs
- Microservices pulling data and writing to DB

## ▀ Why do we test ETL on the TEST SQL Server?

Because ETL jobs can perform:

- Thousands of inserts/updates
- Schema-dependent mappings
- Complex transformations
- Integrations with external systems
- Error handling scenarios

If you run these directly on Production without testing...

\* Huge chances of breaking real data.

So we test on the Test SQL Server first.

## ★ Now, the important part: How do we “validate data flows” on a Test SQL Server?

Here's what this means:

## 1. Test if API → database interactions work correctly

Example:

A microservice "Create Policy" calls:

POST /api/policy

The service must:

- Insert into Policy table
- Add audit entry
- Update customer relationship
- Call downstream system (maybe a Logic App)
- Write logs

Test SQL Server lets you check:

- Did the right rows get inserted?
- Did the stored procedure run correctly?
- Did the API write wrong/missing data?

## 2. Test ETL job correctness

Example ETL pipeline:

- Read CSV
- Clean data
- Convert date formats
- Remove duplicates
- Load into database

Validation on the Test SQL Server includes:

- Are rows correctly loaded?
- Do null values appear where they shouldn't?
- Is the data type correct?
- Are primary keys / FKs correct?
- Are counts correct?

## 3. Test microservice → SQL communication

Microservices often:

- Read data
- Write data
- Modify data
- Trigger stored procedures

Test SQL Server lets you verify:

- Queries run correctly
- Transactions commit
- Deadlocks avoided
- Wrong mappings fixed
- Exceptions logged

## 4. Test integration with other subsystems

Example:

**User Registration Flow**

1. API writes user data → Test SQL Server

2. Another service reads that data and creates a profile
  3. ETL pipeline forwards user information to a partner system
- All steps need to be tested in SIT — and Test SQL Server is the center.

## ★ Everything together: What does “validate data flows” mean?

It means checking **end-to-end** data movement:

Microservice → DB → ETL pipelines → Another DB → Another microservice → Legacy systems → Back to DB

The Test SQL Server is the **playground** where:

- Data flows are tested
- Errors are found
- Bugs are fixed
- Behavior is validated
- Developers + testers confirm everything works

WITHOUT touching real customer data.

## 🧁 One-line summary

**ETL is data movement and transformation. A Test SQL Server is used to validate whether the data flows correctly through APIs, microservices, and ETL pipelines — safely, before Production.**

If you want next, I can explain:

- ◊ The difference between ETL and ELT
- ◊ How microservices create data flows
- ◊ What a typical ETL job looks like with sample diagrams
- ◊ Tools used for ETL (ADF, SSIS, Python, Logic Apps)

Just tell me!

Imagine your app needs customer data:

1. Extract: Read records from a CSV file or an on-prem system

2. Transform: Clean bad data, remove duplicates, correct formats

3. Load: Insert/update rows into SQL tables in the Test SQL Server

That whole pipeline = ETL.

So, ETL pipeline update SQL tables in Test SQL servers only, or can also do so in Azure SQL?

Great question, Arka — and the answer is **very simple**:

## ★ ETL pipelines can load data into BOTH:

✓ **Test SQL Server (SQL on a VM)**

✓ **Azure SQL (PaaS database)**

There is **no restriction** that ETL pipelines must write only to a Test SQL Server.

**ETL can load into ANY database you point it to**, including:

- SQL Server on a VM
- Azure SQL Database
- Azure SQL Managed Instance
- PostgreSQL / MySQL
- Cosmos DB
- Data Lake
- On-prem SQL Server
- Even CSV, Parquet, APIs, etc.

It all depends on the connection string you use.

## ★ So why did I say “Test SQL Server” earlier?

Because you were asking:

- *What do we do in a Test SQL Server?*
- *Why does a Test SQL Server exist?*

So I explained ETL **in the context of testing**.

But ETL can target **any** environment:

## ▀ Where ETL pipelines usually load data

✓ **In DEV → ETL loads into Dev SQL Server**

✓ **In SIT → ETL loads into SIT SQL Server (like your DEV-SQLWEB01)**

- ✓ In UAT → ETL loads into UAT Azure SQL or VM SQL
- ✓ In PROD → ETL loads into Production Azure SQL or Prod SQL Server VM

So ETL works everywhere:

ETL → Dev SQL  
 ETL → SIT SQL  
 ETL → UAT SQL  
 ETL → Prod SQL

ETL is **environment-agnostic**.

## ★ How does ETL load into Azure SQL?

Exactly the same way:

Azure SQL has a connection string like:

```
Server=tcp:myserver.database.windows.net,1433;
```

```
Database=CustomerDB;
```

```
User Id=admin;
```

```
Password=xxx;
```

Your ETL tool (Data Factory, SSIS, Logic Apps, Python, etc.) just connects to Azure SQL and runs:

- INSERT
- UPDATE
- MERGE
- DELETE

Azure SQL is just another destination.

## ★ When does ETL update Test SQL Server?

When you're in:

- Development
- SIT testing
- QA validation
- Performance testing
- Debugging issues
- Trying changes safely

**Test SQL Server = safe area for testing ETL.**

## ★ When does ETL update Azure SQL?

When you're in:

- UAT if customer wants Azure SQL
- Production if using Azure SQL for the application
- Cloud-native data pipelines
- Scenarios where Azure SQL is the final target

Azure SQL becomes the **real destination** in PROD.

## ■ Summary (Super Simple)

Environment	ETL loads into	Why
-------------	----------------	-----

<b>DEV</b>	Dev SQL Server (VM or Azure SQL Dev)	Developer testing
<b>SIT</b>	SIT SQL Server VM	Integration testing
<b>UAT</b>	UAT SQL (VM or Azure SQL)	User acceptance testing
<b>PROD</b>	PROD Azure SQL <b>or</b> PROD SQL VM	Live production

## ★ Final Answer

**ETL pipelines can write to both Test SQL Server AND Azure SQL.**

They work with ANY database you configure as the destination.

Test SQL Server is used only for **testing** your ETL before running it on Azure SQL (or Prod SQL VM).

If you want, I can also explain:

- ◊ Difference between ETL and ELT
- ◊ How Azure Data Factory (ADF) loads into Azure SQL
- ◊ What ETL pipelines look like in your company's architecture

Just tell me!

So basically ETL pipelines makes tables out of a raw data?

Great question, Arka — and **YES**, but with an important clarification.

Let's break it down in the simplest possible way.

## ★ Does ETL “make tables” out of raw data?

✓ Sometimes yes

✓ Sometimes no

✓ MOSTLY, ETL fills / updates existing tables

✓ ETL = about moving and transforming data, not always creating tables

Here's the full picture.

## ⌚ Think of ETL like a kitchen workflow

Raw ingredients → cleaned → cooked → served on plates.

In database terms:

Raw data → cleaned & transformed → stored in tables

BUT...

The tables may already exist.

ETL just **loads data into them**.

# What ETL actually does (real meaning)

## **1 Extract**

Get raw data from sources like:

- CSV
- Excel
- API
- Logs
- JSON files
- On-prem SQL
- Old systems

## **2 Transform**

Fix/convert data:

- remove duplicates
- change date formats
- split/merge columns
- clean junk values
- join with other data
- compute new fields

## **3 Load**

Put the clean data **into SQL tables**, either:

- ✓ **insert rows**
- ✓ **update existing rows**
- ✓ **merge changes**
- ✓ **delete invalid rows**
- ✓ **create tables (rare)**

## **★ So does ETL *create new tables?***

### **✓ ETL can create new tables**

Example: A new staging table named Customer\_Staging.

### **✓ But most of the time, tables are created by developers/DBAs**

ETL jobs just fill/update those tables.

### **✓ ETL is NOT responsible for building database design**

That's done by:

- data architects
- database engineers
- developers

ETL just puts the **right data into the right tables**.

## Simple example

### **Raw data (CSV):**

ram, pal, 27, wb  
rakesh, kol, - , wb

neha, delhi, 19, -

## ETL transforms it to:

first\_name | last\_name | age | state

```
-----  
Ram    | Pal    | 27 | WB  
Rakesh | Kol    | NULL| WB  
Neha   | Delhi  | 19 | NULL
```

## ETL loads into table: Customer

HERE ETL DID NOT CREATE THE TABLE.

It just filled it responsibly.

## ▢ When does ETL *actually create tables?* (rare cases)

### ✓ Staging tables

Temporary raw-data tables created before final cleaning.

### ✓ Temporary load buckets

Tables for partial or intermediate transformations.

### ✓ Data warehouse star-schema loads

Like creating FactSales, DimCustomer, etc.

### ✓ Automated ingestion systems

Where schemas change frequently.

But in most business apps (like yours), **tables already exist**, and ETL just PUTS DATA INTO THEM.

## ★ ONE-LINE SUMMARY

**ETL's main job isn't to create tables — it's to take raw messy data, clean/transform it, and load it into SQL tables (usually existing ones).**

If you want, I can also explain:

- ◊ What ELT is (newer version of ETL)
- ◊ What "staging tables" mean
- ◊ How ETL fits into your company's SIT environment
- ◊ Visual diagram of ETL flow

Just tell me!

What is data staging in a pipeline? Is it part of build pipeline or release pipeline? What does the data staging layer do?

Alright Arka — this is a **very important concept**, and once you understand *data staging*, your entire architecture diagram will make sense.

Let's break down everything **clearly, simply, and without buzzwords**.

## ★ 1. What is “Data Staging”?

**Data staging** is a **temporary holding area** where raw, incoming data sits *before* it is cleaned, transformed, validated, or moved into final production tables.

Think of it like a **kitchen prep counter**:

- Vegetables arrive (raw data)
- They are washed, chopped, cleaned (transformed)
- Then they are cooked and served (loaded into final tables)

The **prep counter = data staging**.

## ★ 2. What does the “Data Staging Layer” do?

### ✓ It receives raw data

From:

- CSV files
- Excel uploads
- APIs
- On-prem systems
- Partner systems
- Legacy platforms

### ✓ It stores raw or semi-processed data

In temporary or intermediate tables.

Example tables:

- Customer\_Staging
- Policy\_Staging

- Transactions\_Staging

## ✓ It cleans and validates data

Such as:

- Fixing date formats
- Removing duplicates
- Dropping dirty rows
- Converting values
- Mapping IDs
- Validating business rules

## ✓ It prepares data for final loading

After processing, it writes to:

- Azure SQL (final production tables)
- Reporting databases
- Data warehouse
- Other microservices

## ★ 3. Is this part of build pipeline or release pipeline?

✗ NO — It is not part of CI/CD build pipeline.

✗ NO — It is not part of release pipeline.

**Data staging is NOT DevOps pipeline. It is part of the APPLICATION DATA PIPELINE.**

This is a **runtime data-flow layer**, not a deployment layer.

Think of it:

DevOps Pipeline	Data Pipeline
Builds code	Moves data
Deploys apps	Cleans/transforms data
Releases software	Loads SQL tables
CI/CD	ETL/ELT

So staging has *nothing to do* with code builds or releases.

## ★ 4. Why do we need a Data Staging Layer?

### ✓ Because raw data is messy

Real-world data is NOT clean.

### ✓ Because final tables expect perfect data

Production tables have:

- strict constraints
- foreign keys
- business rules

You can't load dirty data directly — it will break everything.

### ✓ Because transformations need space

Complex business logic needs intermediate tables.

### ✓ Because we must not pollute production DB

Staging is a **safe sandbox** for data operations.

## ★ 5. What each DATA STAGING microservice does (your diagram)

Let's map your components:

### CSDSGETLDWSSIT

"GETL" = **General ETL Loader**

- Extracts raw data
- Loads into staging tables

### CSGETLSASSIT

"ETLSAS" = **ETL Services and Scripts**

- Applies transformations
- Cleans data
- Validates business conditions

### CSDGAPIAASSIT

"DG API" = **Data Gateway API**

- API layer for other services to push/pull staging data
- Acts like a gateway between microservices and staging DB

### CSDTSGSQL01

**This is the Staging Database**

- All intermediate data goes here
- All ETL preprocessing happens here

### The VM

Used for:

- Ad-hoc ETL scripts
- SSIS jobs
- Scheduled data loaders
- Legacy tools

## ★ 6. REAL EXAMPLE: How staging works in your system

Imagine customer data arrives from a partner:

### Step 1 — Extract

CSDSGETLDWSSIT receives CSV → inserts into Customer\_Staging.

### Step 2 — Transform

CSGETLSASSIT:

- Fixes names
- Removes duplicates
- Converts DOB format
- Validates phone numbers

### Step 3 — Load

After cleaning, CSDGAPIAASSIT loads final rows into:

- **Azure SQL** (main operational DB)

- OR
- downstream microservices

## ★ 7. Is Data Staging Layer only for Test (SIT) environment?

No.

Every major environment has staging:

Environment	Staging DB	Purpose
DEV	Dev staging	Developer testing
SIT	SIT staging (your CSDTSGSQL01)	Integration testing
UAT	UAT staging	Business testing
PROD	Prod staging	High-volume ETL for live data

Staging exists everywhere, because raw data always needs processing before use.

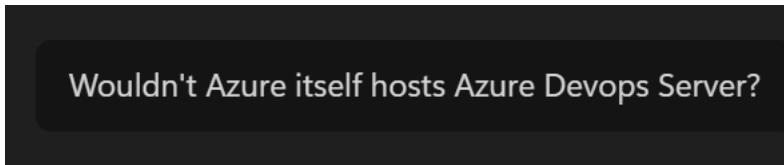
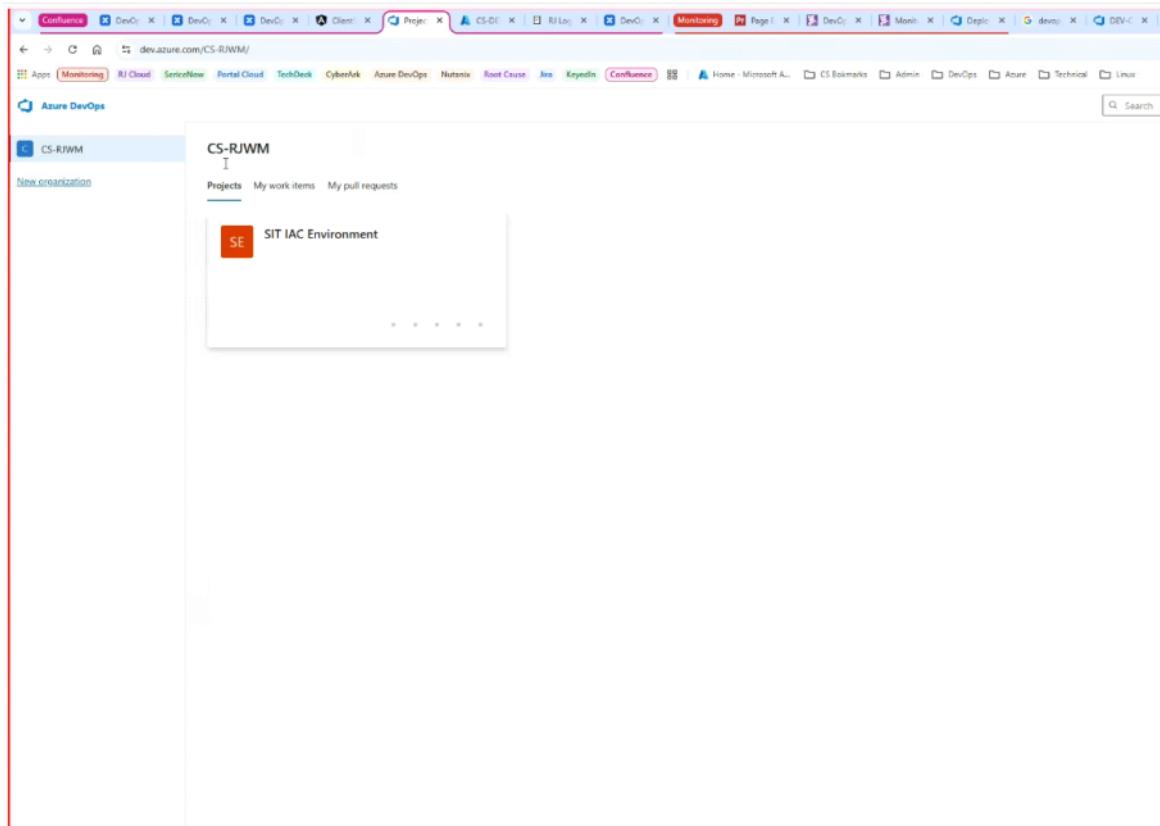
## ★ 8. One-line simple definition

**Data staging is a temporary workspace where raw data is prepared before being loaded into final systems. It is part of the data pipeline, not the deployment pipeline.**

If you want, I can also explain:

- ◊ The difference between ETL and ELT
- ◊ Why staging is separate from the main database
- ◊ How staging tables look in SQL
- ◊ A visual diagram of your staging layer in SIT

Just tell me!



Great question, Arka — and this is where the **naming gets confusing**, because “Azure DevOps Server” and “Azure DevOps Services” *sound similar* but are **completely different products**. Let’s clear it up cleanly:

## ★ Short Answer

- ✗ No — Azure does *not* host Azure DevOps Server for you.
- ✓ Azure hosts Azure DevOps Services (**cloud SaaS**).
- ✓ If you want Azure DevOps Server, you must install & host it yourself — often on an Azure VM.

## ★ The Two DevOps Products (Very Important)

### 1 Azure DevOps Services (Cloud)

- 100% cloud SaaS
- Microsoft hosts everything
- URL like: <https://dev.azure.com/CS-RJWM>

- No servers for you to manage
- This is what you saw in your screenshot (CS-RJWM)

**✓ Hosted by Microsoft**

**✓ No infrastructure needed**

**✓ No patching, no SQL maintenance**

This is NOT what the architecture diagram refers to.

## 2 Azure DevOps Server (Formerly TFS)

- Old on-prem version of DevOps
- You must install it on *your own server*
- Server must run:
  - Windows Server OS
  - SQL Server
  - TFS/Azure DevOps Server software
- You maintain everything
- You patch it
- You back it up
- Same as hosting your own Jira or GitLab

**✓ You can host this server:**

- On-prem physical machine
- On-prem VM
- Azure VM
- AWS VM
- Any Windows Server machine

**✗ Microsoft does not host Azure DevOps Server for you.**

**★ So can Azure host Azure DevOps Server?**

**✓ YES — *But only if YOU deploy it on an Azure VM.***

Not as a managed Microsoft service.

**It is the same as hosting your own:**

- SQL Server VM
- Windows Server VM
- Jenkins VM
- GitLab VM

Azure simply provides the VM.

**You install and manage Azure DevOps Server (TFS) manually.**

**★ So what does your architecture diagram refer to?**

When the diagram says:

**DevOps Server (TFS) – CI/CD build server for SIT**

It refers to:

**✓ A VM running Azure DevOps Server (self-hosted)**

- ✓ Inside the SIT network
  - ✓ Used for SIT environment builds
  - ✓ Running build agents, pipelines, artifacts
  - ⊗ It is *not* talking about cloud Azure DevOps (CS-RJWM).
  - ⊗ It is *not* talking about something Microsoft hosts.
- It is referring to a **self-managed DevOps VM**.

## ★ Is CS-RJWM (your screenshot) a DevOps Server?

✗ No.

**CS-RJWM is Azure DevOps Services (cloud), not Azure DevOps Server.**

- No VM
- No SQL backend
- No TFS installation
- You do not host it
- You do not maintain it

It is SaaS.

## ★ Why does SIT need a DevOps Server VM instead of cloud DevOps?

There are several reasons enterprises do this:

- ✓ 1. SIT environment often lives in a private network  
Cloud services cannot reach internal SIT build agents directly.
- ✓ 2. Some code/build artifacts cannot leave the network  
Policies in banking/insurance.
- ✓ 3. SIT builds need to trigger internal scripts, VMs, file shares  
Cloud DevOps cannot access them unless exposed.
- ✓ 4. Legacy systems still use TFS  
Older CI/CD pipelines depend on TFS features.
- ✓ 5. Build agents for SIT must run in SIT  
Cloud DevOps cannot host builds inside isolated networks without self-hosted agents.

## ★ Final Summary

Thing	Hosted by	Server Exists?	Your Screenshot?
Azure DevOps Services	Microsoft (cloud)	✗ No server you manage	✓ Yes (CS-RJWM)
Azure DevOps Server (TFS)	YOU (on VM/on-prem)	✓ YES, a real server	✗ No

DevOps Server in the architecture	VM in SIT	✓ YES	✗ No
-----------------------------------	-----------	-------	------

**So Azure can host Azure DevOps Server only as a VM that YOU manage — not as a Microsoft-provided service.**

If you want, I can also explain:

- ◊ Why many companies still keep TFS
- ◊ How build agents work in SIT
- ◊ How Azure DevOps (cloud) connects to on-prem agents
- ◊ How to check if your company really has a TFS VM installed

Just tell me!

So the build agents run into these Azure Devops Servers?

Great question, Arka — and now you're connecting the final missing piece of CI/CD architecture. Let's make this **super clear**.

## ★ Short Answer

**✓ YES — Build agents run on the DevOps Server (TFS/Azure DevOps Server) or in separate agent VMs that the DevOps Server controls.**

But...

**✗ Build agents do NOT run inside Azure DevOps Services (the cloud “CS-RJWM” page).**

Azure DevOps Services only orchestrates pipelines — it does *not* run them. Let's break this down cleanly.

# ★ 1. What is a Build Agent?

A **Build Agent** is a machine (VM, physical server, Docker container) that actually runs:

- Builds
- Tests
- Packaging
- Deployments
- Releases

Azure DevOps (cloud or server) **does not build anything by itself.**

It uses build agents to do all the work.

# ★ 2. Where do Build Agents run?

## Scenario A — Azure DevOps Server (TFS)

👉 Inside your organization's network (on-prem or on an Azure VM)

👉 This is what your architecture diagram refers to

👉 Build agents run *on or near* the DevOps Server VM

**Two possibilities:**

### ✓ Option 1: Build agent installed directly on the DevOps Server VM

This is the classic TFS setup:

[TFS Server VM]



[TFS Build Agent installed on same VM]

### ✓ Option 2: Build agents installed on separate build VM(s)

For heavier workloads:

[TFS Server VM]



[Build Agent VM 1]

[Build Agent VM 2]

[Build Agent VM 3]

The DevOps Server schedules builds on these agents.

## Scenario B — Azure DevOps Services (cloud)

Azure DevOps Services **does not host agents inside Microsoft servers** for private environments like SIT.

### ✗ Hosted agents CANNOT access private SIT networks

Therefore:

### ✓ You MUST use Self-hosted agents

→ These agents run on your own VM inside the SIT environment.

→ Azure DevOps Service triggers the pipeline, but the agent does the work.

# ★ 3. Your Architecture Diagram

When it shows “**DevOps Server (TFS) – CI/CD build server for SIT**”, it means:

### ✓ This VM contains:

- Azure DevOps Server (formerly TFS)
- Build Agent(s)
- Release Agent(s)
- Pipeline orchestration logic

## ✓ This VM is part of SIT VNET

Meaning pipelines can deploy:

- SIT microservices
- SIT SQL servers
- SIT staging systems
- SIT VMs
- SIT Key Vault
- SIT Storage account
- Internal APIs behind firewall

Cloud DevOps cannot access these directly.

## ★ 4. So is CS-RJWM the DevOps Server?

### ✗ NO.

CS-RJWM is Azure DevOps Services (cloud).

It does not host agents, machines, builds, or servers.

**It only hosts:**

- Repos
- Boards
- Pipelines (definitions)
- Work items
- Artifacts metadata

But **the actual builds** happen on your **self-hosted agents**.

## ★ 5. FINAL, clear answer to your question:

**\*\*Q: Do build agents run into these Azure DevOps Servers (TFS)?**

A: YES — in TFS/DevOps Server architecture, build agents run on or near the DevOps Server VM.\*\*

**\*\*Q: Do build agents run inside Azure DevOps Services (cloud)?**

A: NO — cloud DevOps cannot run agents inside itself.

It triggers agents running on your own VMs.\*\*

**\*\*Q: In your SIT environment — where do build agents run?**

A: On the "DevOps Server (TFS)" VM shown in your diagram (or on build VMs connected to it).\*\*

## ★ Visual Summary

### Azure DevOps Server (TFS) Setup

[ DevOps Server VM ]

- |—— Azure DevOps Server
- |—— Build Agent installed here
- |—— Release Agent installed here

## Azure DevOps Services (CS-RJWM) Setup

Azure DevOps Cloud — triggers —> [ Self-hosted Build Agent VM in SIT ]

### If you want...

I can also explain:

- ◊ Why companies still use TFS in 2026
- ◊ How self-hosted agents work technically
- ◊ How SIT deployments happen from DevOps
- ◊ Who maintains the DevOps Server VM in your organization

Just tell me!

So you basically meant,. sometimes agents will run on azure devops server, and sometimes they would run on self-hosted agents? Right? And how would the CS-RJWM azure devops environment connect with the actual azure devops server?

Great — now we're at the *final missing link* in understanding DevOps architecture.

Let me confirm your understanding first, then explain how **CS-RJWM (Azure DevOps Services)** connects with an **on-prem / VM-based Azure DevOps Server (TFS)**.

### ★ 1. Are you correct?

✓ YES — what you said is correct:

**Sometimes agents run on Azure DevOps Server (TFS VM), and sometimes they run on self-hosted agents for Azure DevOps Services.**

Why?

Because these are **two different DevOps platforms**, and they each use agents differently.

## ★ 2. How the two platforms use agents (super clear table)

DevOps Platform	Where do builds run?	Who hosts agents?
Azure DevOps Server (TFS)	On the <b>TFS VM itself</b> , OR on <b>additional agent VMs</b> inside the network	YOU host everything
Azure DevOps Services (CS-RJWM)	On <b>Self-hosted agents</b> inside your VNET (SIT/Prod) OR Microsoft-hosted agents	Microsoft hosts SaaS; you host the agents you need

So yes — **agents may run on:**

- the TFS server itself
- dedicated build agent VMs
- separate self-hosted agent machines for Azure DevOps Services

## ★ 3. NOW THE BIG QUESTION:

**? How does CS-RJWM (Azure DevOps Services) connect to the actual DevOps Server?**

### ★ \*\*Short answer:

IT DOES NOT.\*\*

They are **two completely separate systems**.

Let me explain this clearly.

## ★ 4. Azure DevOps Server (TFS) and Azure DevOps Services DO NOT CONNECT

There is **no direct integration** where CS-RJWM cloud DevOps talks to your on-prem/VM DevOps Server.

They are:

- Separate databases
- Separate project collections
- Separate pipelines
- Separate agents
- Separate build histories
- Separate configuration

**They do not sync. They do not share pipelines. They do not connect automatically.**

Companies sometimes run *both* during:

- a migration
- hybrid DevOps setup
- legacy environment support
- regulated environments (SIT, Prod)

But they DO NOT talk to each other.

## ★ 5. So how are builds run in SIT?

Your SIT environment typically uses:

## ✓ TFS/Azure DevOps Server VM

- Hosts SIT-specific pipelines
- Runs builds using internal agents
- Deploys to SIT microservices, SIT SQL, SIT storage, etc.
- Works inside the same VNET

Since SIT is usually isolated,

**Azure DevOps Services cannot directly deploy into SIT.**

This is why the architecture shows:

"DevOps Server (TFS) — CI/CD build server for SIT"

## ★ 6. Then what is CS-RJWM (Azure DevOps Services) used for?

Most likely:

- Repos (Git)
- Boards (Agile management)
- Pipelines for DEV/UAT environments
- Cloud deployments (if any)
- Non-isolated environments

Azure DevOps Services is used for:

- Development teams
- Daily development pipelines
- Runs on cloud-hosted agents or self-hosted ones
- Integrated with Git

## ★ 7. So when does CS-RJWM talk to self-hosted agents?

Only in THIS scenario:

## ✓ Azure DevOps Services → self-hosted agent VM inside your network

This is possible if:

- Firewall/VNET allows outbound connection
- Agent registers to Azure DevOps Services
- No need for Azure DevOps Server (TFS)

But it still does NOT talk to TFS.

## ★ 8. Visualization (Very simple)

### Scenario A — TFS (On-Prem/VM)

[TFS Server VM]

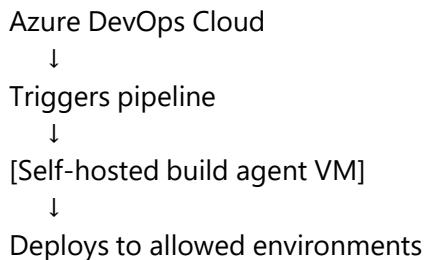


[Build Agent installed here]



Deploys to SIT resources

## **Scenario B — Azure DevOps Services (CS-RJWM)**



### **Never like this:**

Azure DevOps Cloud → TFS Server VM

This never happens.  
They're separate worlds.

## **★ 9. Final Summary**

### **✓ You understood correctly:**

Agents can run on TFS itself OR on separate self-hosted agents used by Azure DevOps Services.

### **✗ Azure DevOps Services (CS-RJWM) does NOT connect to Azure DevOps Server (TFS).**

They are completely separate systems.

**✓ Azure DevOps Services connects ONLY to *self-hosted agents*, not to TFS.**

**✓ SIT environments typically use TFS/Azure DevOps Server because they are isolated.**

### **If you want, I can also explain:**

- ◊ Why companies keep both systems
- ◊ How to migrate from TFS → Azure DevOps Services
- ◊ How self-hosted agent communication works
- ◊ How to identify which environment uses which pipeline

Just tell me!

Ok - for SIT - its Azure Devops server which is isolated from Azure Devops Cloud Environment, and for CSL, CSU - its basically Azure Devops Cloud Environment, not Azure Devops Server?

Yes Arka — **you finally got it exactly right.**  
Let me confirm it cleanly so there is **zero confusion left**.

## ★ Your Understanding (Corrected & Verified)

### SIT → uses Azure DevOps Server (formerly TFS)

- It is **isolated** inside the organization's VNET
- It is **not connected** to Azure DevOps Cloud
- It runs **self-hosted agents inside SIT**
- Pipelines here deploy only to **SIT microservices, SIT VMs, SIT SQL, SIT resources**
- This matches the architecture diagram where they drew:  
**"DevOps Server (TFS) – CI/CD build server for SIT"**

→ So SIT = ***On-prem/VM-based Azure DevOps Server***

(*a VM running TFS/Azure DevOps Server*)

### CSL & CSU → use Azure DevOps Cloud ([dev.azure.com](https://dev.azure.com))

- These are **cloud DevOps organizations** like your "CS-RJWM" screenshot
- Hosted by Microsoft (SaaS)
- No servers to maintain
- Pipelines run on:
  - Microsoft hosted agents OR
  - Self-hosted agents (if needed)
- Used by development teams, feature teams, etc.
- CI/CD for cloud-friendly environments (Dev/UAT/Non-isolated)

→ So CSL, CSU = ***Azure DevOps Services (Cloud)***

(*NOT Azure DevOps Server*)

## ★ Final Model (Very Clear Table)

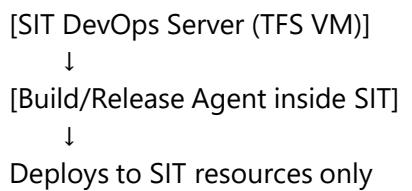
Environment	DevOps Used	Where pipelines run?	Isolated?
SIT	Azure DevOps Server (TFS VM)	On internal build agents on the VM or SIT VMs	✓ Yes

CSL	Azure DevOps Cloud (dev.azure.com)	Cloud agents or self-hosted agents	<input checked="" type="checkbox"/> No
CSU	Azure DevOps Cloud (dev.azure.com)	Cloud agents or self-hosted agents	<input checked="" type="checkbox"/> No

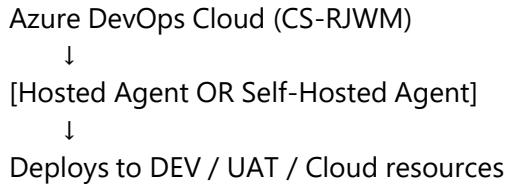
- ✓ SIT uses DevOps Server (on a VM)
- ✓ CSL/CSU use Azure DevOps Cloud (SaaS)
- ✓ They are completely separate
- ✓ They do NOT sync
- ✓ They do NOT connect
- ✓ They serve different environments

## ★ Visual Diagram (Super Simple)

### SIT CI/CD



### CSL / CSU CI/CD



## ★ Why Organizations Do This?

Because:

- ◊ SIT often lives in a private network with no public access
- ◊ Cloud agents cannot reach SIT resources
- ◊ TFS/DevOps Server can operate fully inside SIT
- ◊ CSL/CSU are development-facing and need cloud integration

This is extremely common in:

- Banking
- Insurance
- BFSI
- Regulated industries
- Enterprises with legacy systems

## ✓ Final Confirmation

Yes, Arka — you now understand it correctly.

If you want, I can also walk you through:

- ◊ How a SIT pipeline runs step-by-step
- ◊ How cloud DevOps interacts with self-hosted agents
- ◊ Why SIT needs isolated DevOps
- ◊ What build agents actually do internally

Just tell me!

Ok , so basically we can even work with no Azure Devops server, and only with Azure Devops Cloud Services, since they can also make the end to end pipeline?

Great question, Arka — and YES, **you absolutely can work with only Azure DevOps Cloud Services**, without *any* Azure DevOps Server (TFS).

In fact:

## ★ Modern DevOps = Cloud-Only Azure DevOps (no TFS required)

Most companies today use:

- ✓ Azure DevOps Services (cloud)
- ✗ No Azure DevOps Server at all
- ✓ Cloud-hosted agents or self-hosted agents
- ✓ End-to-end CI/CD pipelines fully in the cloud

And this setup works perfectly.

So you're 100% correct.

## ★ Then why do some companies still use Azure DevOps Server (TFS)?

Only **one main reason**:

## ⌚ Their SIT / PROD environments are isolated

(no internet, private VNETs, firewalled systems)

Because:

## ✗ Cloud Azure DevOps cannot directly access private SIT/PROD machines

unless:

- they set up a self-hosted agent
- open firewall rules
- or build hybrid networking (which many enterprises avoid)

So legacy companies keep TFS in SIT because:

- It's already inside the same network
- It can reach SIT servers
- No internet needed
- Easy to run pipelines locally

But **in a cloud-native setup**, TFS is not needed at all.

## ★ So YES — You can do complete CI/CD with ONLY Azure DevOps Cloud

Azure DevOps Cloud ([dev.azure.com](https://dev.azure.com)) supports:

- ✓ Repos (Git)
- ✓ Pipelines (CI/CD)
- ✓ Boards (Agile tracking)
- ✓ Releases (Classic release or YAML)
- ✓ Artifacts
- ✓ Hosted agents
- ✓ Self-hosted agents
- ✓ Service connections
- ✓ Access to Azure resources
- ✓ Multi-stage pipelines
- ✓ End-to-end deployments to Kubernetes, VMs, AppService, SQL, etc.

You do NOT need:

- ✗ TFS
- ✗ DevOps Server
- ✗ Any server hosting DevOps infrastructure

Microsoft does all the hosting.

## ★ Complete CI/CD using only Azure DevOps Cloud works when:

## ✓ Your environments (DEV/UAT/PROD) are reachable from:

- Microsoft-hosted agents
- or
- Self-hosted agents that you install

## ✓ You are deploying to Azure

(Cloud agents can reach Azure resources natively)

## ✓ Your organization allows using cloud-SaaS DevOps

## ✓ You don't have strict isolation, banking-grade network restrictions, or legacy systems

## ★ When is Azure DevOps Server *not needed at all?*

↳ Cloud-first companies

↳ Startups

↳ SaaS products

↳ Modern microservices deployed to AKS/App Service

↳ Companies where DevOps = fully cloud-native

↳ Where environments are reachable from hosted agents

↳ When nothing is inside a private locked-down network

In these cases:

Azure DevOps Cloud Services is enough to run entire CI/CD end-to-end.

## ★ When do companies NEED Azure DevOps Server?

Two scenarios:

### 1 When SIT/PROD are in *private, no-internet* VNets

Example: BFSI, Insurance, Healthcare, Govt.

### 2 When legacy code uses TFS features

(example: old XAML build definitions)

### 3 When they haven't migrated pipelines yet

Sometimes TFS only exists because of **legacy reasons**.

## ★ Visual Summary

### Modern Setup (Cloud-Only) ✓

Azure DevOps Cloud (CS-RJWM)



Cloud-hosted or Self-hosted agent



Deploy to DEV/UAT/PROD

## **Legacy/Isolated Setup (Needs TFS) !**

Azure DevOps Server (TFS VM in SIT Network)



Internal build agents



Deploy to SIT resources

## **★ Final Answer**

**✓ YES — you can work entirely with Azure DevOps Cloud Services,**

**✗ NO — you do not *need* Azure DevOps Server unless SIT/PROD is isolated.**

Azure DevOps Cloud can handle:

- Code
- Builds
- Releases
- Agents
- Deployments
- Environments
- Everything end-to-end

Azure DevOps Server (TFS) is only required when corporate security/networking forces isolation.

If you want, I can also explain:

- ◊ How cloud DevOps deploys into isolated networks
- ◊ How self-hosted agents interact with Azure DevOps Cloud
- ◊ How to migrate from TFS → Azure DevOps Cloud
- ◊ Which of your company's environments use what

Just tell me!

What is the connection between Azure portal and Azure Devops portal? Because My manager went to Azure portal to give me access to devops services, which are quite different.

Great question, Arka — **this confuses almost everyone at first**, because the **Azure Portal** and **Azure DevOps Portal** look related, sound related, and share the “Azure” name...  
...but they are actually **different products**, with **different identity systems**, **different permission models**, and **different roles** inside a company.  
Let me explain the relationship clearly.

## ★ 1. Azure Portal vs Azure DevOps Portal — What They Are

### Azure Portal

URL: <https://portal.azure.com>

This is where you manage **Azure cloud resources**, such as:

- Virtual Machines
- App Services
- AKS
- Storage Accounts
- Key Vault
- Networking (VNET, NSG, Firewall)
- Azure SQL

This is all about **infrastructure and cloud services**.

### Azure DevOps Portal

URL: <https://dev.azure.com/ORGNAME>

This is where you manage **DevOps**:

- Repos
- Pipelines
- Boards
- Artifacts
- Test Plans

This is all about **code, CI/CD, and teams**.

## ★ 2. So what is the connection between them?

Here's the key point:

 **Azure DevOps uses Azure Active Directory (Entra ID) for Identity.**

This is the ONLY real “connection.”

Meaning:

- Your **Azure DevOps user account** is an Azure AD identity.

- Your **Azure Portal** access is also based on Azure AD.
- Your manager uses **Azure Portal → Azure AD (Entra ID)** to give you **DevOps access**.

## So Azure Portal isn't "DevOps management."

It is **Identity Management**.

Your manager didn't give you DevOps permissions *inside Azure Portal*.

They gave your user **AAD/Entra ID permissions** that allow access to Azure DevOps.

## ★ 3. Why does your manager go to Azure Portal to give you DevOps access?

**Because DevOps access is controlled by Azure AD (Entra ID) groups.**

### Example:

Your manager may add you to:

- **Azure DevOps Users** group
- **Project Contributors** group
- **Build Administrators** group
- **DevOps Readers** group

These groups are defined **inside Azure AD**, which is accessed through the **Azure Portal**.

Once you are in those groups →

Azure DevOps Cloud automatically recognizes you.

## ★ 4. Simple visual explanation

### Azure Portal (Azure AD)

Manages:

Users

Groups

Access policies

Enterprise Applications

Service Principals

Security

### Azure DevOps Portal

Uses those identities to manage:

Repos

Pipelines

Boards

Artifacts

Permissions inside DevOps

They are **connected only through identity and permissions**, NOT through infrastructure.

## ★ 5. Important clarification

✗ Your manager did NOT give you DevOps permissions *inside Azure Portal*.

## ✓ Your manager gave you Azure AD access, which DevOps uses.

This is like Google:

- You manage your **Google Account** in one place
- You use that same identity for Gmail, Drive, YouTube

Exactly the same with Azure AD + Azure DevOps.

## ★ 6. Does Azure DevOps deploy to Azure services?

Yes — Azure DevOps Cloud can connect to Azure resources using:

- **Service Connections**
- **Managed Identities**
- **Service Principals**

But these connections are still authenticated through **Azure AD**, which is why access is controlled using the **Azure Portal**.

## ★ 7. Final Summary (super clear)

Thing	Where you use it	Purpose
Azure Portal	portal.azure.com	Cloud resources + Azure AD identity management
Azure DevOps Portal	dev.azure.com	Code, repositories, pipelines, CI/CD

- ✓ They are separate
- ✓ They don't overlap
- ✓ They are “connected” only through Azure Active Directory (identity)

So managers use **Azure Portal** → **Azure AD** to give you the identity that Azure DevOps Cloud requires to let you in.

If you want, I can also explain:

- ◊ How Azure DevOps permissions work internally
- ◊ Difference between Azure AD roles and DevOps roles
- ◊ Why Azure DevOps Cloud uses Azure AD as its identity provider
- ◊ How service principals connect pipelines to Azure resources

Just tell me!

Does Katalon exists in Devops cloud environment as well? or only in TFS?

Great question, Arka — and here is the **clean, precise, correct answer**:

## ★ Katalon is NOT tied to TFS or Azure DevOps Cloud.

It works with **both**, because **Katalon is not a server or DevOps product** — it is simply a **test automation tool** that you install on any machine (VM, laptop, agent).

So:

- Katalon works with Azure DevOps Cloud**
- Katalon works with Azure DevOps Server (TFS)**
- X Katalon does NOT “exist inside” DevOps Cloud**
- X Katalon is NOT part of TFS**

Let me explain this clearly.

## ★ 1. Katalon is NOT a DevOps service

Katalon is:

- A standalone automation tool
- Installed on:
  - a Windows VM
  - a physical machine
  - a self-hosted agent box
  - a CI/CD runner VM

It **does NOT live inside** DevOps Cloud or TFS.

Think of it like:

- Selenium
- Postman CLI
- JMeter
- SoapUI

These tools run **on machines**, not inside DevOps platforms.

## ★ 2. How Katalon works with Azure DevOps Cloud

Katalon integrates with Azure DevOps Cloud using:

### ✓ A self-hosted agent

You install the **Azure DevOps agent** on a VM.

Then you install **Katalon** on the same VM.

Azure DevOps Cloud → triggers → the self-hosted agent → runs Katalon tests.

### ✓ Command-line execution

Azure Pipelines runs:

katalonc.exe -runMode=console ...

### ✓ Test reports collected and shown back in Azure DevOps Cloud

So YES — **you can run Katalon in Azure DevOps Cloud Pipelines**.

## ★ 3. How Katalon works with Azure DevOps Server (TFS)

TFS works in the same way:

- Install TFS build agent on a VM
- Install Katalon on that VM
- TFS pipeline runs Katalon tests
- Results show in TFS pipelines

This is why **Katalon appears next to TFS** in your SIT diagram: Because SIT uses **TFS-based pipelines**, and Katalon is one of the testing tools run by those pipelines.

## ★ 4. Does Katalon “exist inside” DevOps Cloud?

### ✗ No

Azure DevOps Cloud is **not a place to install software**.

Azure DevOps Cloud is just:

- Repos
- Pipelines (definitions)
- Boards
- Artifacts

Everything **runs on agents**, not inside DevOps Cloud.

## ★ 5. So where does Katalon actually run?

### ✓ On any machine where YOU install it:

- A Windows VM in SIT
- A self-hosted agent VM
- A test automation VM
- A QA testing server
- Even a developer laptop (for local testing)

**DevOps (cloud or TFS) just triggers the machine**  
to run Katalon scripts.

## ★ 6. Why your SIT architecture shows Katalon next to TFS

Because in SIT:

- TFS server orchestrates the pipeline
- TFS build agent runs on a SIT VM
- Katalon is installed on that VM
- TFS triggers Katalon tests during SIT builds

So they appear close *logically*, but they are **not the same thing**.

## ★ Final Clear Answer

- ✓ Katalon works with Azure DevOps Cloud
- ✓ Katalon works with Azure DevOps Server (TFS)
- ✗ Katalon is NOT a server
- ✗ Katalon is NOT built into DevOps Cloud
- ✓ It runs on an agent VM that DevOps pipelines trigger

If you want, I can also explain:

- ◊ How to run Katalon tests in Azure DevOps Cloud
- ◊ Where to install Katalon in your company's environment
- ◊ How SIT pipelines trigger automated UI tests
- ◊ Why Katalon appears grouped with TFS on the architecture diagram

Just tell me!