

Azure Devops 2

Friday, January 2, 2026 1:08 PM

1. Where is the codebase stored?

- Your **source code** is stored in **Azure Repos** (Git repositories) or any connected Git provider (like GitHub, Bitbucket).
- When you trigger a pipeline, Azure DevOps **clones** the repository into the build agent (either Microsoft-hosted or self-hosted).

2. How does Azure Portal run it?

- **Microsoft-hosted agent:**

Azure spins up a temporary **virtual machine** in the cloud for your pipeline job.

Steps:

1. VM starts → pipeline agent installed.
2. Your repo is cloned into that VM.
3. Build/test/deploy tasks run.
4. VM is destroyed after the job finishes.

- **Self-hosted agent:**

You install an agent on your own machine/server.

Steps:

1. Pipeline triggers → your machine picks up the job.
2. Code is either already present or cloned locally.
3. Tasks run using your hardware.
4. No time limit (unlimited minutes).

These are VMs provided by Azure DevOps that spin up automatically when you run a pipeline.

so suppose it triggered a pipeline, and VMs got triggered and build the job - now my server is live, so now would the VM be destroyed? if destroyed, then won't my server be down again?

Excellent question! Here's the key point:

When you use **Microsoft-hosted agents**, the VM that runs your pipeline is **ephemeral**—it exists only for the duration of the job. After the pipeline finishes, the VM is **destroyed**.

But this does **NOT** affect your actual server or application because:

- The VM is **only for building and testing your code**, not for hosting your live application.
- After the build, the pipeline typically **deploys your code to your real server or cloud service** (e.g., Azure App Service, Kubernetes, VM, etc.).
- Your production server remains live because the deployment step pushes the built artifacts to

it.

So:

- VM spins up → builds your code → deploys to your server → VM is destroyed.
- Your server stays up because it's independent of the build VM.

Do you want me to **illustrate this with a simple diagram** showing the flow:
Pipeline Trigger → Temporary VM → Build → Deploy → VM Destroyed → Server Live?

Great question! Let me explain **how the VM pushes build artifacts to your production server** in a clear, step-by-step way:

What happens after the VM builds your code?

The VM is temporary, but before it is destroyed, it **deploys the build artifacts** (your compiled code, packages, etc.) to your production environment. This is done using **deployment tasks** in your pipeline.

How does the deployment work?

1. Build Phase (on VM)

- The VM compiles your code and creates build artifacts (e.g., .zip, jar, Docker image).
- These artifacts are stored temporarily on the VM.

2. Publish Artifacts

- The pipeline uses tasks like PublishBuildArtifacts or PublishPipelineArtifact to upload artifacts to **Azure DevOps Artifacts** or a storage location (Azure Blob, etc.).

3. Deploy Phase

- The pipeline then runs deployment tasks to push these artifacts to your **production server** or **cloud service**.
- Common deployment methods:
 - **Azure Web App Deploy** (for web apps)
 - **Azure VM Copy Files + SSH** (for VMs)
 - **Kubernetes Deploy**
 - **FTP/SFTP** (for on-prem servers)
 - **Docker Push** (for container registries)

4. VM Destroyed

- After deployment is complete, the VM is deleted.
- Your production server now has the updated code and stays live.

Key Idea

The VM is **never your production server**. It's just a temporary build machine. Your actual server (or cloud service) is separate and remains live because the pipeline **deploys the code to it before the VM is destroyed**.

Do you want me to **draw a simple diagram showing this flow**:

Pipeline Trigger → Temporary VM → Build → Push Artifacts → Deploy to Server → VM

Destroyed → Server Live?

Or should I **write a sample YAML pipeline that includes build + deploy steps?**

The pipeline then runs deployment tasks to push these artifacts to your production server or cloud service.

Ok, Now i have 2 questions:

1. Is the pipeline running in the temporary VM?
2. So, production server is already live - but now the pipeline will push the newer changed artifacts to the production server - right? to update the server....How is it doing so - pls lucidly

Great questions! Let's break them down clearly:

1. Is the pipeline running in the temporary VM?

Yes.

When you use **Microsoft-hosted agents**, the entire pipeline (build, test, deploy steps) runs inside that temporary VM.

- The VM is created when the pipeline starts.
- The pipeline agent is installed on that VM.
- All tasks in your pipeline execute there.
- After the job finishes, the VM is destroyed.

2. Production server is already live, but pipeline pushes newer artifacts to update it — how does that happen?

Here's the **lucid flow**:

- Your production server (or cloud service) is **separate** from the build VM and is already running the old version of your app.
- After the VM builds the new version, the pipeline uses **deployment tasks** to **push the new artifacts to the production server**.

How does the VM push artifacts?

- The pipeline includes a **deployment stage** (after build).
- Common methods:
 - **Azure Web App Deploy** → Deploys directly to Azure App Service.
 - **Copy Files Over SSH** → Copies files to your on-prem or cloud VM via SSH.
 - **FTP/SFTP** → Uploads files to a server.

- **Azure CLI / PowerShell** → Executes commands to update resources.
- **Docker Push + Kubernetes Apply** → Updates containers in a cluster.
- Once deployment is complete, your production server now runs the **new version** of the app.
- The temporary VM is then destroyed, but your production server stays live because it has the updated code.

- So the VM is **just a builder and deployer**, not your live server.
- Your server stays up because the pipeline **copies the new build to it before the VM disappears**.

Would you like me to:

- **Draw a simple diagram showing this entire flow visually** (Trigger → VM → Build → Deploy → VM Destroyed → Server Live)?
- Or **write a sample Azure DevOps YAML pipeline with build + deploy steps** for clarity?

so what is our production server here? kubernetes service?

Great follow-up! Yes, your **production server** can absolutely be a **Kubernetes cluster** (or any other hosting environment). Let me explain:

What is the production server in this context?

- It's the environment where your application runs for end-users.
- This could be:
 - **Azure Kubernetes Service (AKS)** → for containerized apps.
 - **Azure App Service** → for web apps.
 - **Azure Virtual Machines** → for traditional deployments.
 - **On-prem servers** → via SSH/FTP.
 - **Other cloud services** → AWS, GCP, etc.

How does the pipeline deploy to Kubernetes?

If your production environment is **Kubernetes**, the pipeline will:

1. Build your application (e.g., create Docker images).
2. Push the image to a container registry (Azure Container Registry or Docker Hub).
3. Use kubectl or Helm tasks in the pipeline to **update the Kubernetes deployment** with the new image.

So the flow is:

- **Temporary VM builds** → **pushes image** → **updates Kubernetes cluster** → **VM destroyed** → **Kubernetes stays live with new version**.

Do you want me to:

- **Draw a diagram showing this Kubernetes deployment flow?**
- **Or write a sample Azure DevOps YAML pipeline for Kubernetes deployment?**

How does the pipeline deploy to Kubernetes?

If your production environment is Kubernetes, the pipeline will:

1. Build your application (e.g., create Docker images).

Push the image to a container registry (Azure Container Registry or Docker Hub).

Now, my question is - suppose Azure Contain Registry got the new image- but how would the kubernetes cluster which serves the live server would know about it?

You just said that the temporary VM updates the kubernetes cluster, but i really ain't got the idea how!

How does Kubernetes know about the new image in Azure Container Registry (ACR)?

Kubernetes doesn't automatically detect new images in ACR. Instead, the pipeline explicitly **tells Kubernetes to update its deployment** using commands.

Here's the flow:

Step 1: Build & Push Image

- The pipeline builds a Docker image of your app.
- Tags it (e.g., `myapp:v2`).
- Pushes it to ACR (or Docker Hub).

Step 2: Update Kubernetes Deployment

- The pipeline uses `kubectl` or `Helm` commands to update the Kubernetes cluster.
- Example:

```
1 kubectl set image deployment/myapp myapp=acrname.azurecr.io/myapp:v2
```

This tells Kubernetes:

- For the deployment named `myapp`
- Replace the container image with the new one from ACR (v2).
- Kubernetes then:
 - Pulls the new image from ACR.
 - Performs a **rolling update** (gradually replacing old pods with new ones).
 - Keeps the service live during the update.

Step 3: Cluster Stays Live

- Your Kubernetes cluster is your **production server**.
- The temporary Azure VM only runs the pipeline and sends the update command.
- After the VM is destroyed, Kubernetes continues running the new version.

Key Idea

The pipeline doesn't just push the image to ACR; it also **executes deployment commands** to tell Kubernetes to use that new image.