AZURE DEVOPS

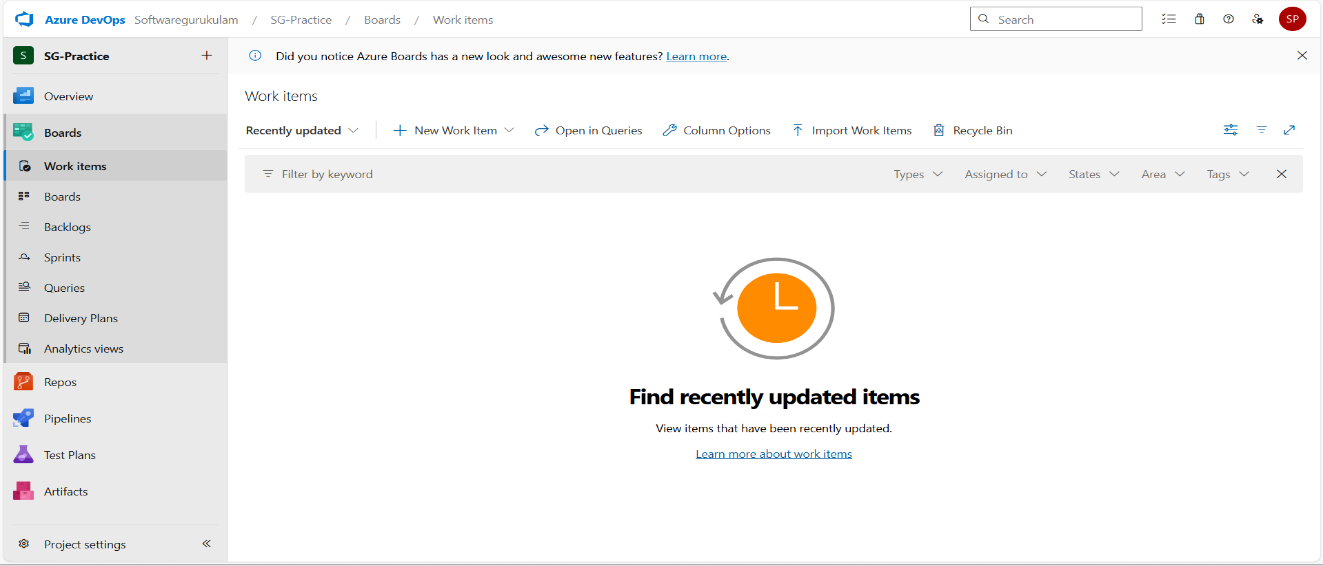
Task:5

Key features of Azure DevOps:

**🔷 1. Azure Boards**

A powerful tool for **work planning and tracking**.

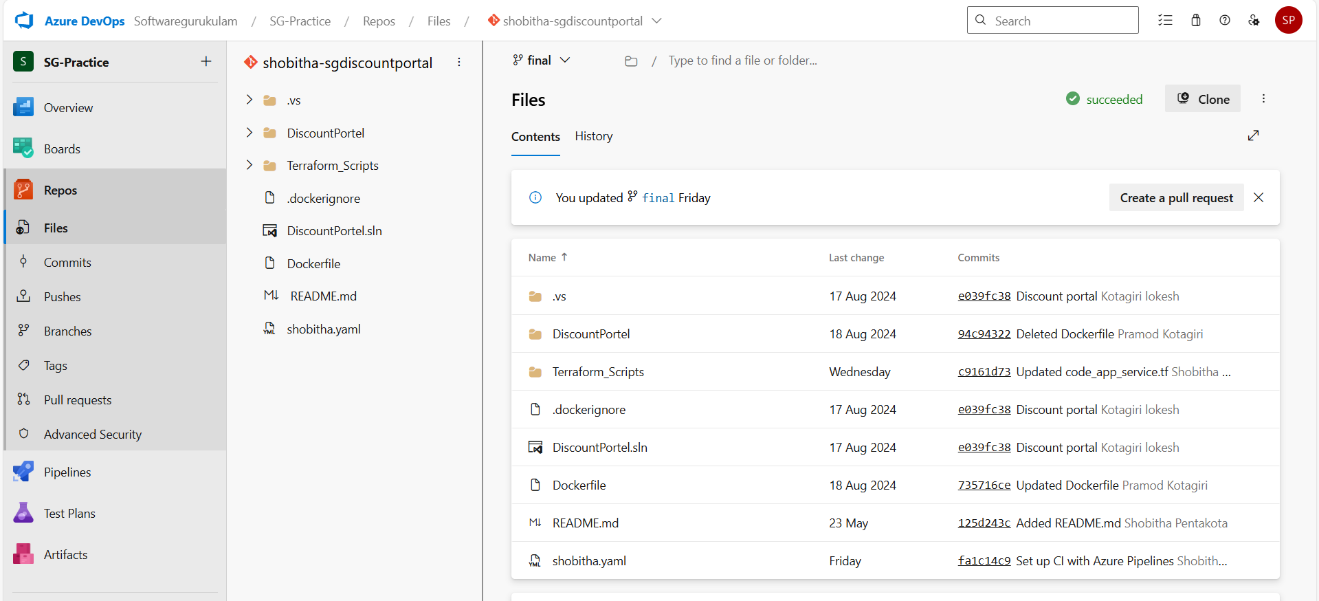
* **Features:**
  + Work items (User Stories, Bugs, Tasks, etc.)
  + Kanban boards
  + Scrum boards
  + Backlogs
  + Sprints
  + Dashboards and reporting
* **Use Cases:** Agile project management, tracking progress, prioritizing work.



**🔷 2. Azure Repos**

Cloud-hosted **version control systems**.

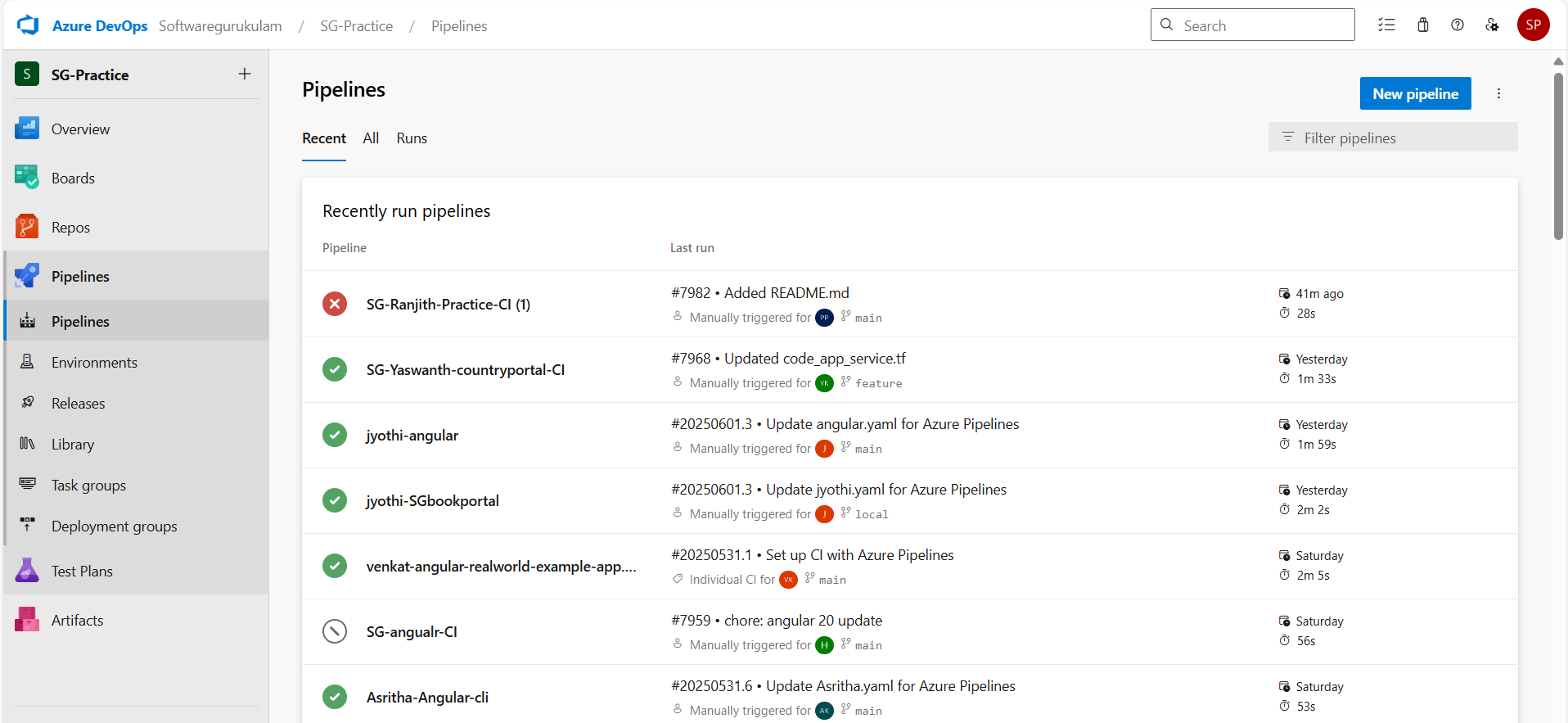
* **Features:**
  + Git repositories (distributed)
  + Team Foundation Version Control (TFVC) (centralized)
  + Branching, Pull Requests, Code Reviews
  + Policies for quality control (e.g., reviewer requirements)
* **Use Cases:** Source code hosting and collaboration.



**🔷 3. Azure Pipelines**

**CI/CD (Continuous Integration and Continuous Delivery)** system.

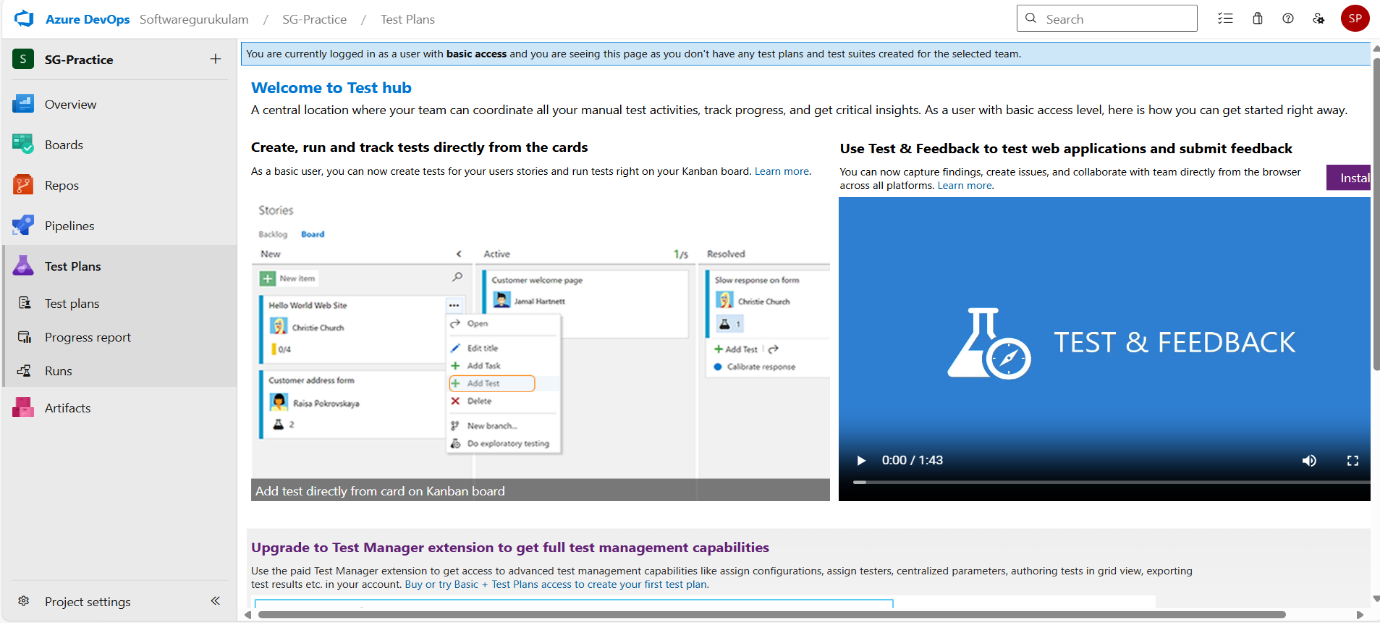
* **Features:**
  + Supports multiple languages/platforms (Node.js, Python, Java, .NET, etc.)
  + Integration with GitHub, Bitbucket, Docker, Kubernetes
  + Build and release pipelines (YAML or visual designer)
  + Hosted agents (Windows, Linux, macOS) or self-hosted
* **Use Cases:** Automate build, test, and deployment processes.



**🔷 4. Azure Test Plans**

Manual and automated **testing tools**.

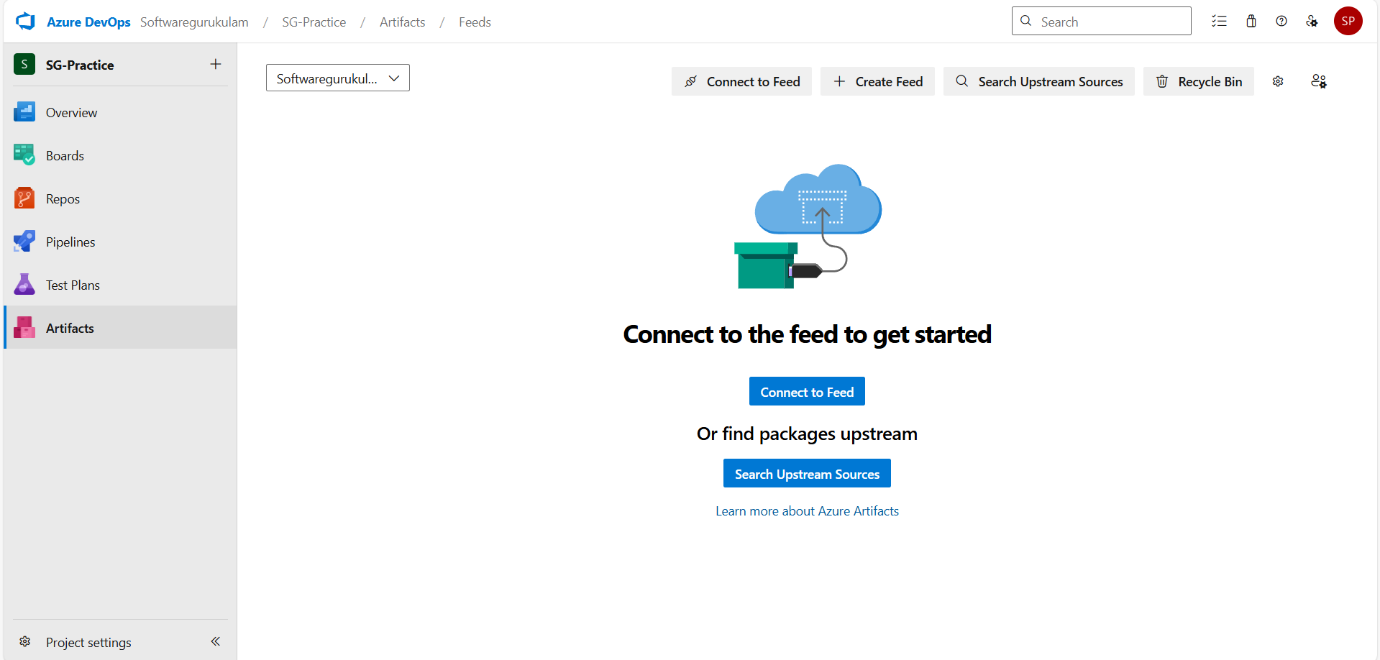
* **Features:**
  + Test case management
  + Manual/exploratory testing
  + Integration with Azure Pipelines for automated tests
  + Defect tracking
  + Test coverage and reporting
* **Use Cases:** QA processes, regression testing, test planning.



**🔷 5. Azure Artifacts**

**Package management** for your builds.

* **Features:**
  + Host NuGet, npm, Maven, Python, and Universal packages
  + Upstream sources to cache public packages
  + Versioning and retention policies
  + Integration with Pipelines
* **Use Cases:** Share and manage build artifacts and dependencies.



TASK:

**Step-by-step guide** to set up a **CI (Continuous Integration) pipeline in Azure DevOps Classic Editor** for a **.NET application**:

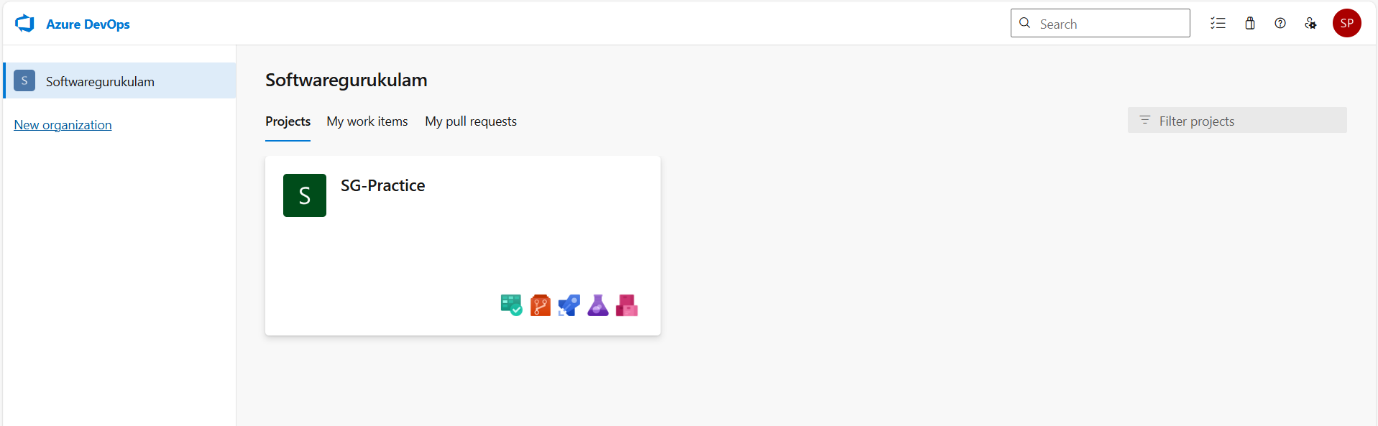
**✅ What is Continuous Integration (CI)?**

Continuous Integration (CI) is a software development practice where developers frequently integrate their code into a shared repository, typically multiple times a day. Each integration is automatically verified by an automated build and tests, allowing teams to detect problems early.

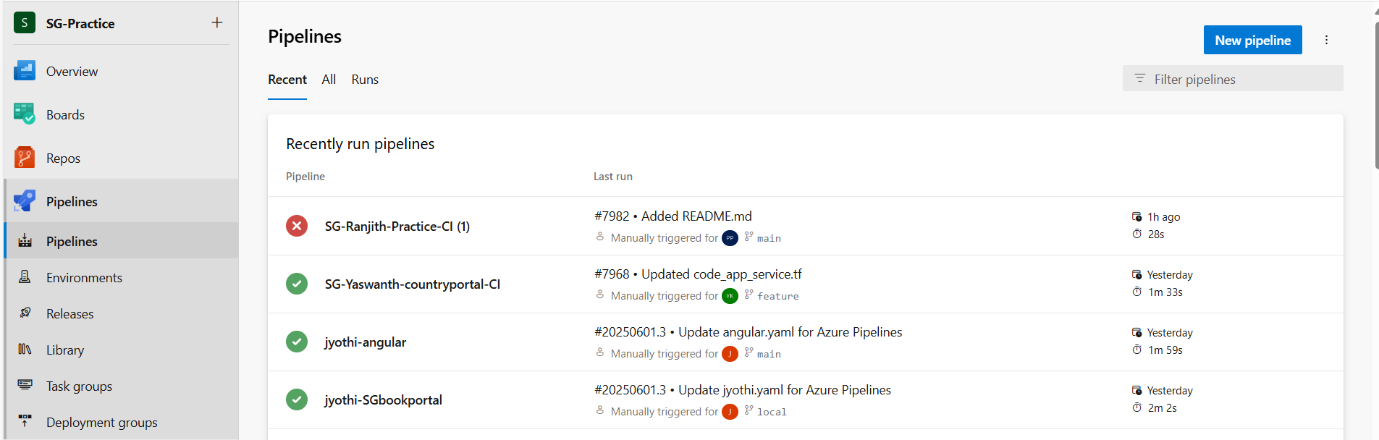
**🔨 Step-by-Step: CI Pipeline in Classic Editor**

**🔹 1. Navigate to Pipelines**

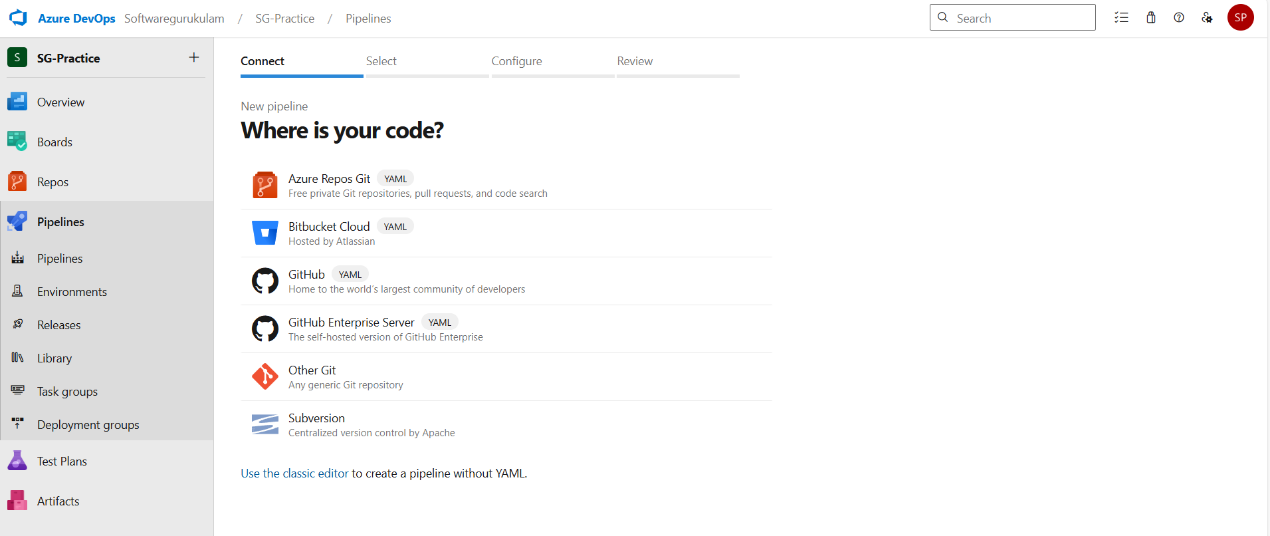
* Go to your Azure DevOps project



* Select **Pipelines > Builds**

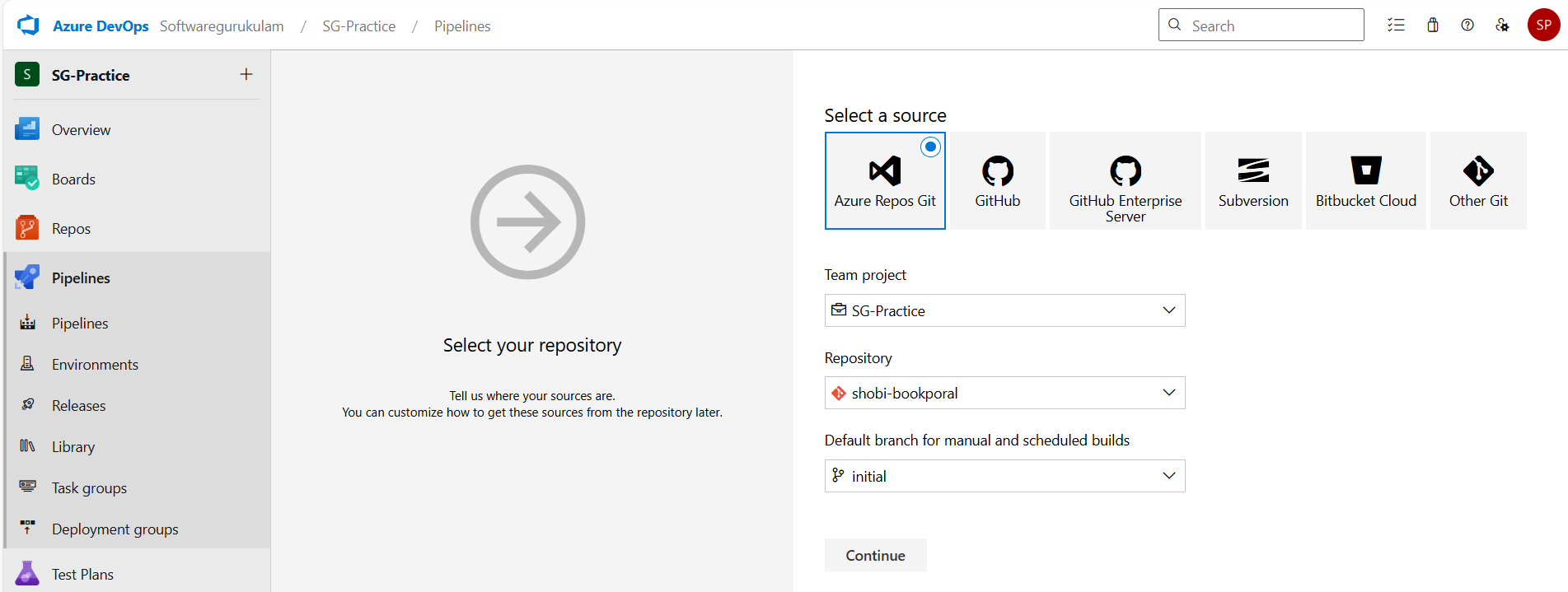


* Click **“New Pipeline”**



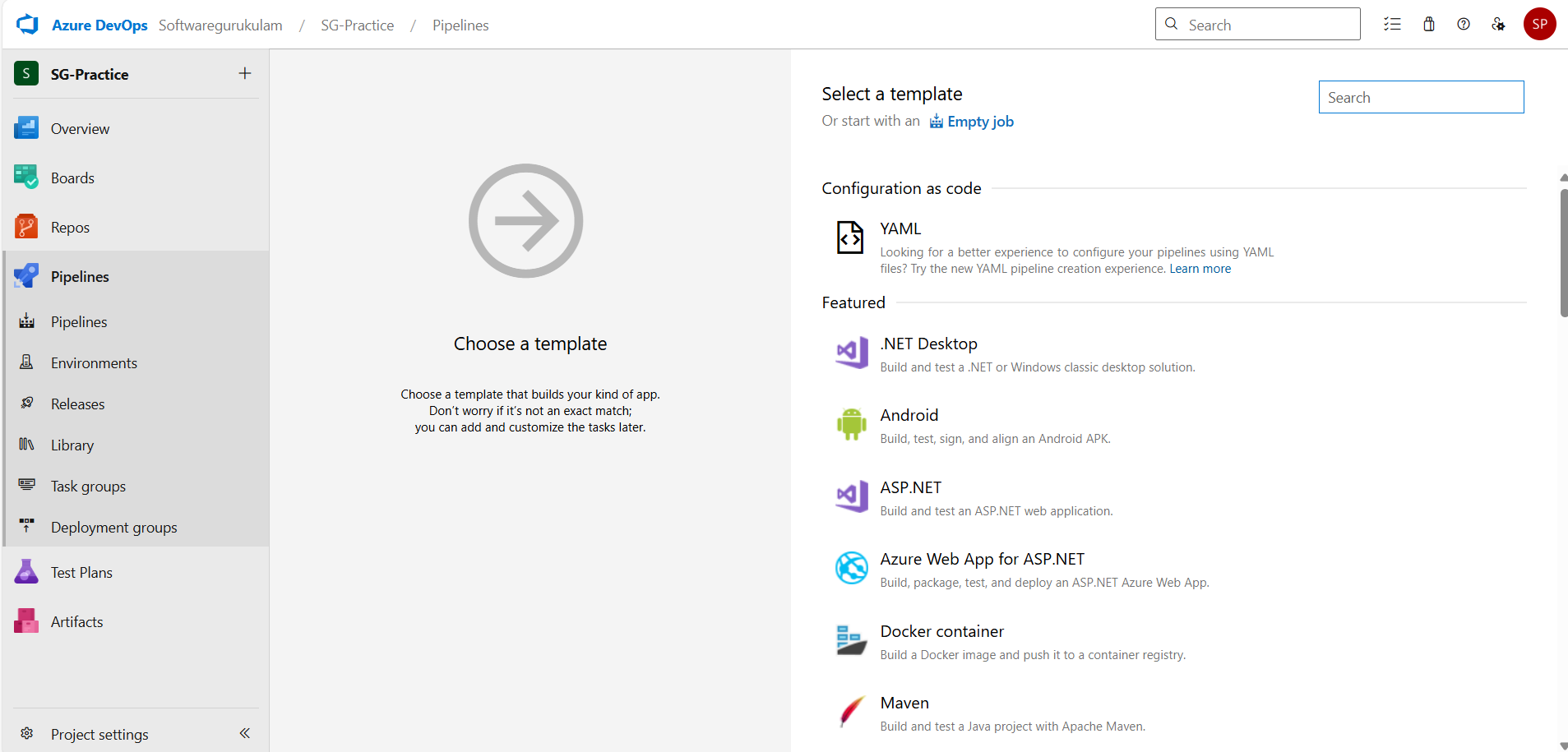
**🔹 2. Select Classic Editor**

* Choose **"Use the classic editor"**
* Select your **repository type** (Azure Repos Git, GitHub, Bitbucket, etc.)
* Choose the **repository** and branch (e.g., main or develop)
* Click **Continue**



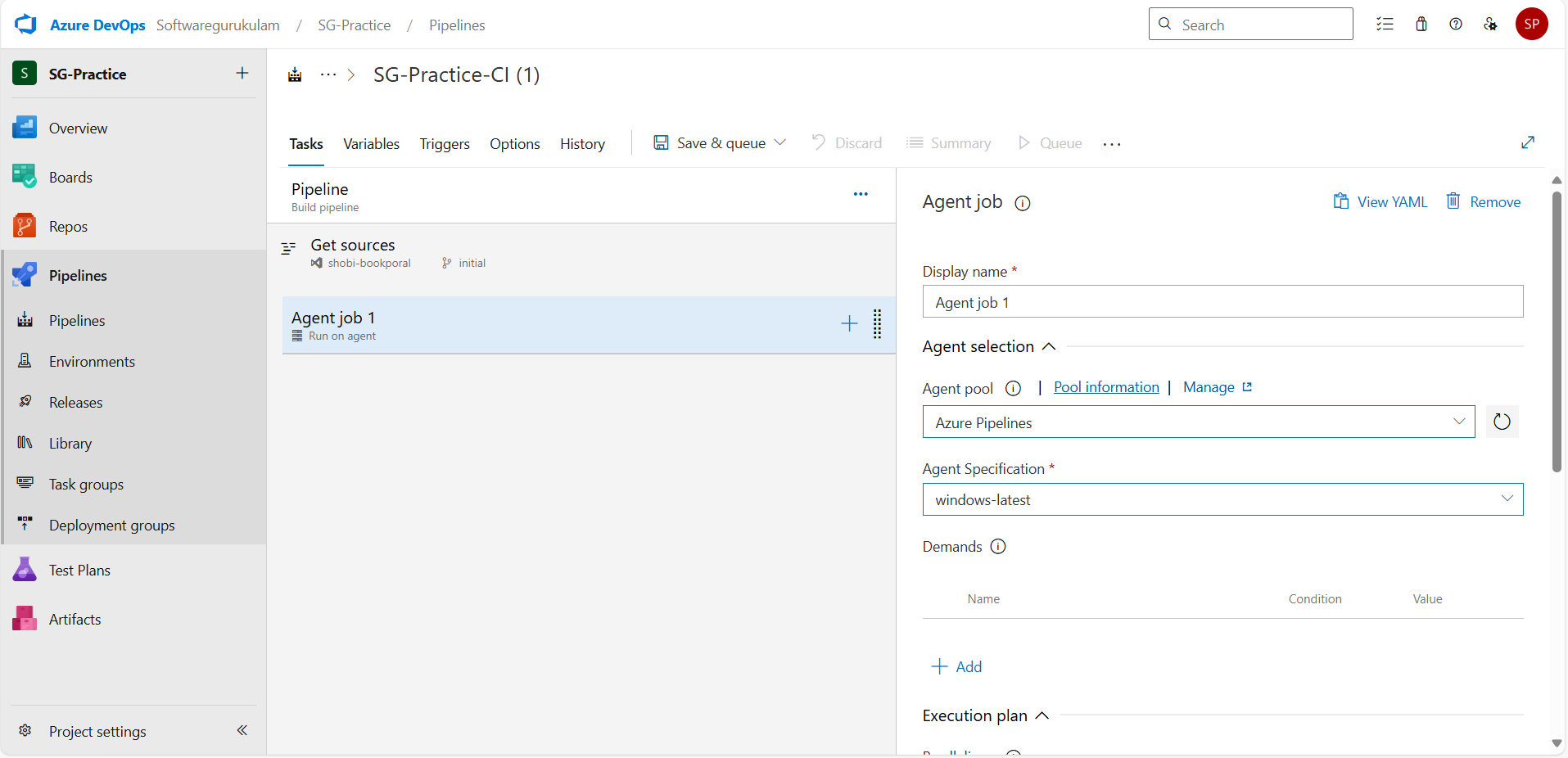
**🔹 3. Select a Template**

* Choose **“.NET Desktop”**, **“.NET Core”**, or start with an **empty job** (recommended for full control)
* Click **Apply**



**🔹 4. Configure Agent Job**

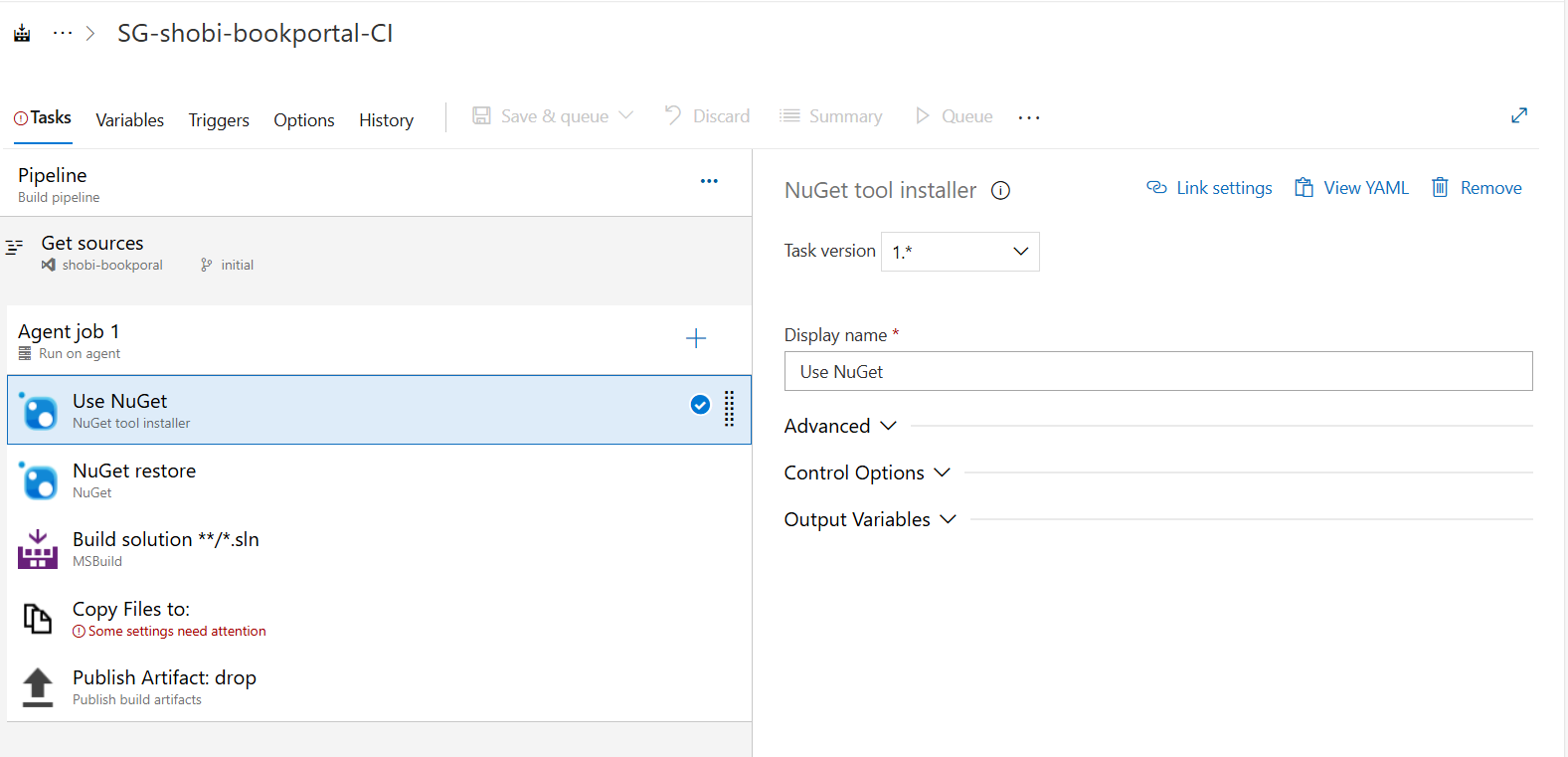
* Set **Agent Pool** (e.g., Azure Pipelines for Microsoft-hosted agents)
* Optionally rename the job (e.g., BuildJob)



🔹 **5. Add Tasks to the Pipeline**

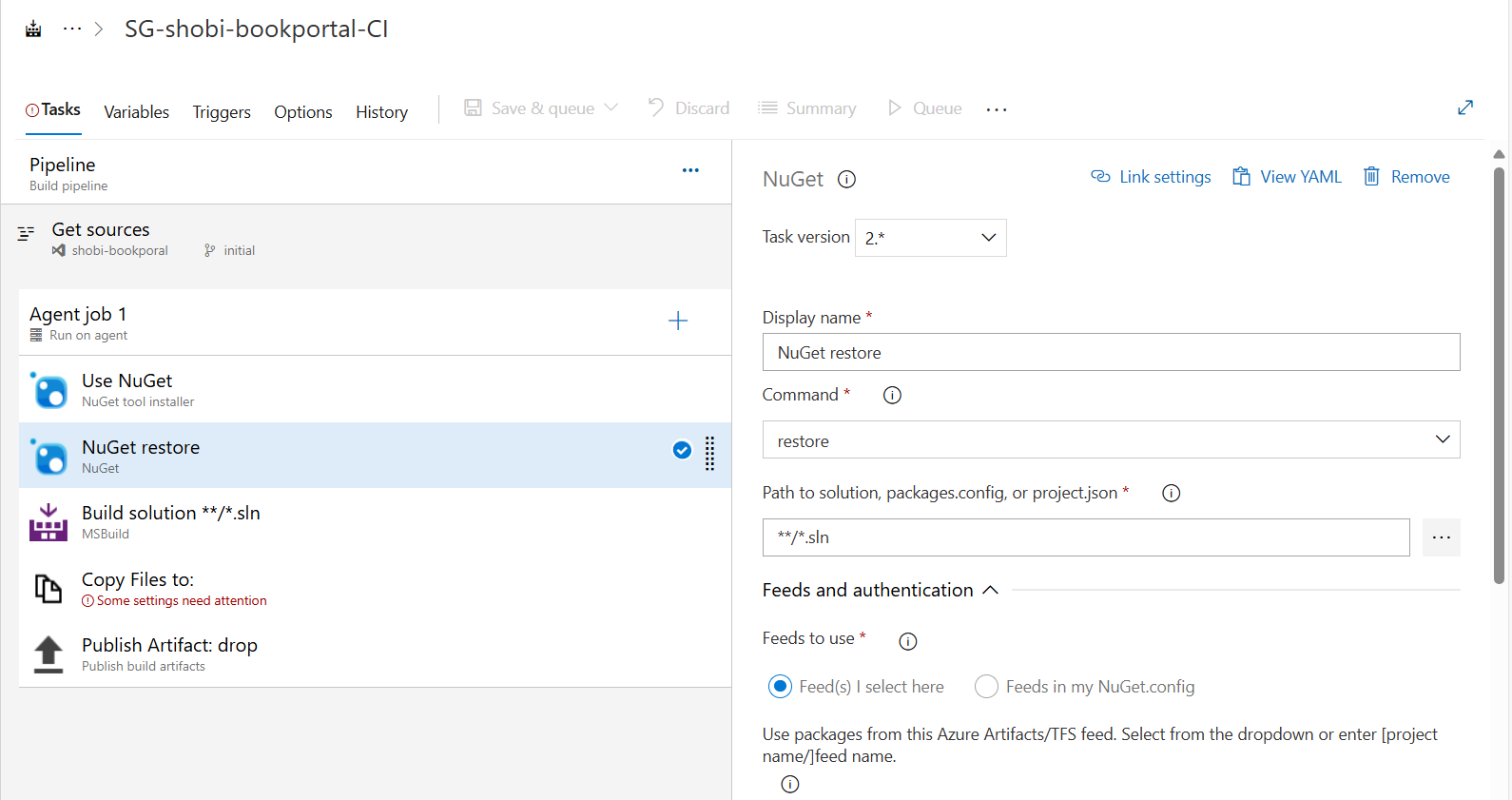
✅ **a.** NUGET TOOL INSTALLER:

The NuGet Tool Installer is commonly used in Azure DevOps pipelines or CI/CD environments to install a specific version of the NuGet CLI (Command Line Interface) before performing tasks like restoring, packing, or pushing NuGet packages.



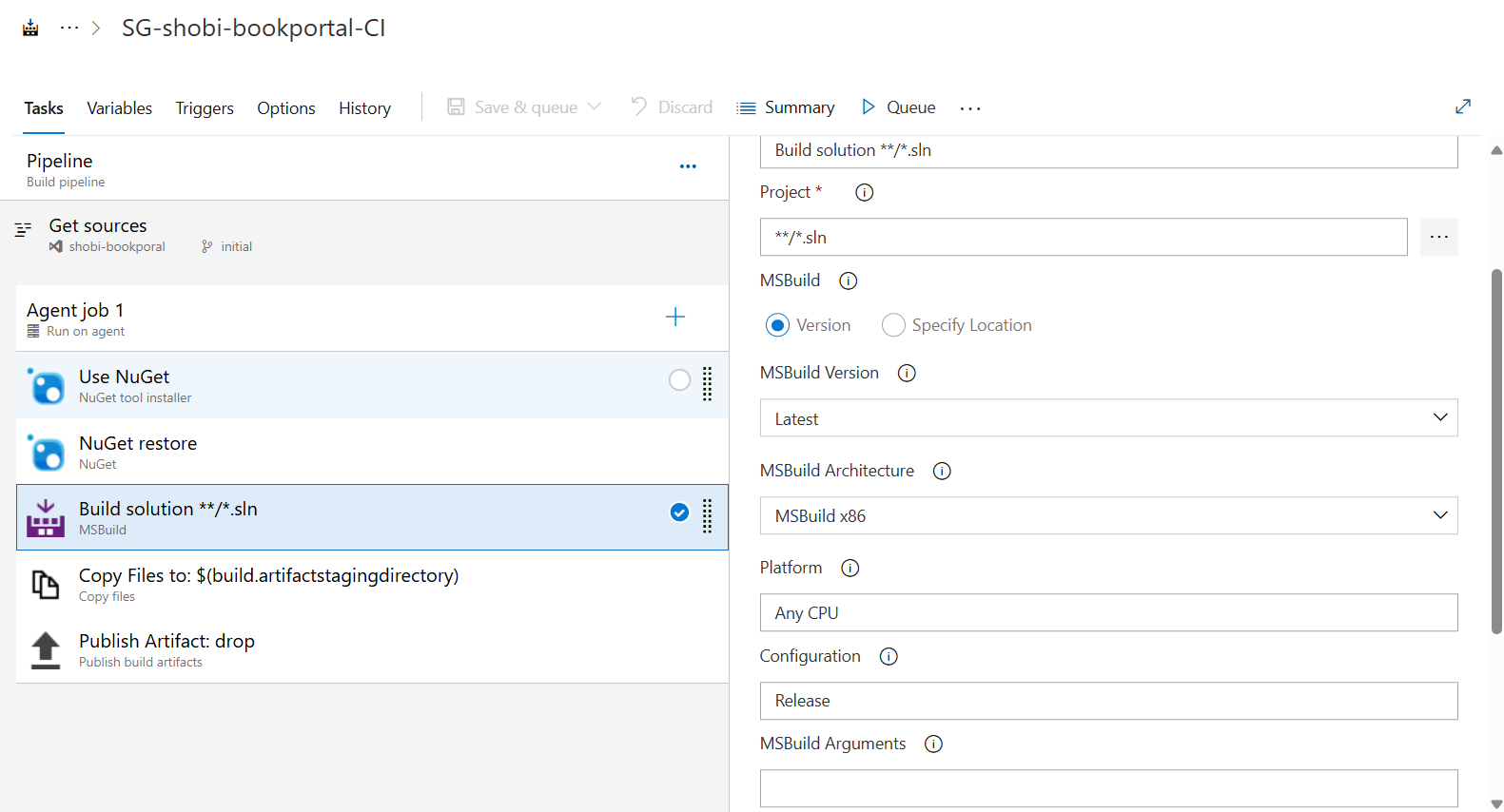
**✅ b. Restore NuGet Packages**

* Task: **NuGet**
* Display name: NuGet Restore
* Command: restore
* Path to solution: \*\*/\*.sln

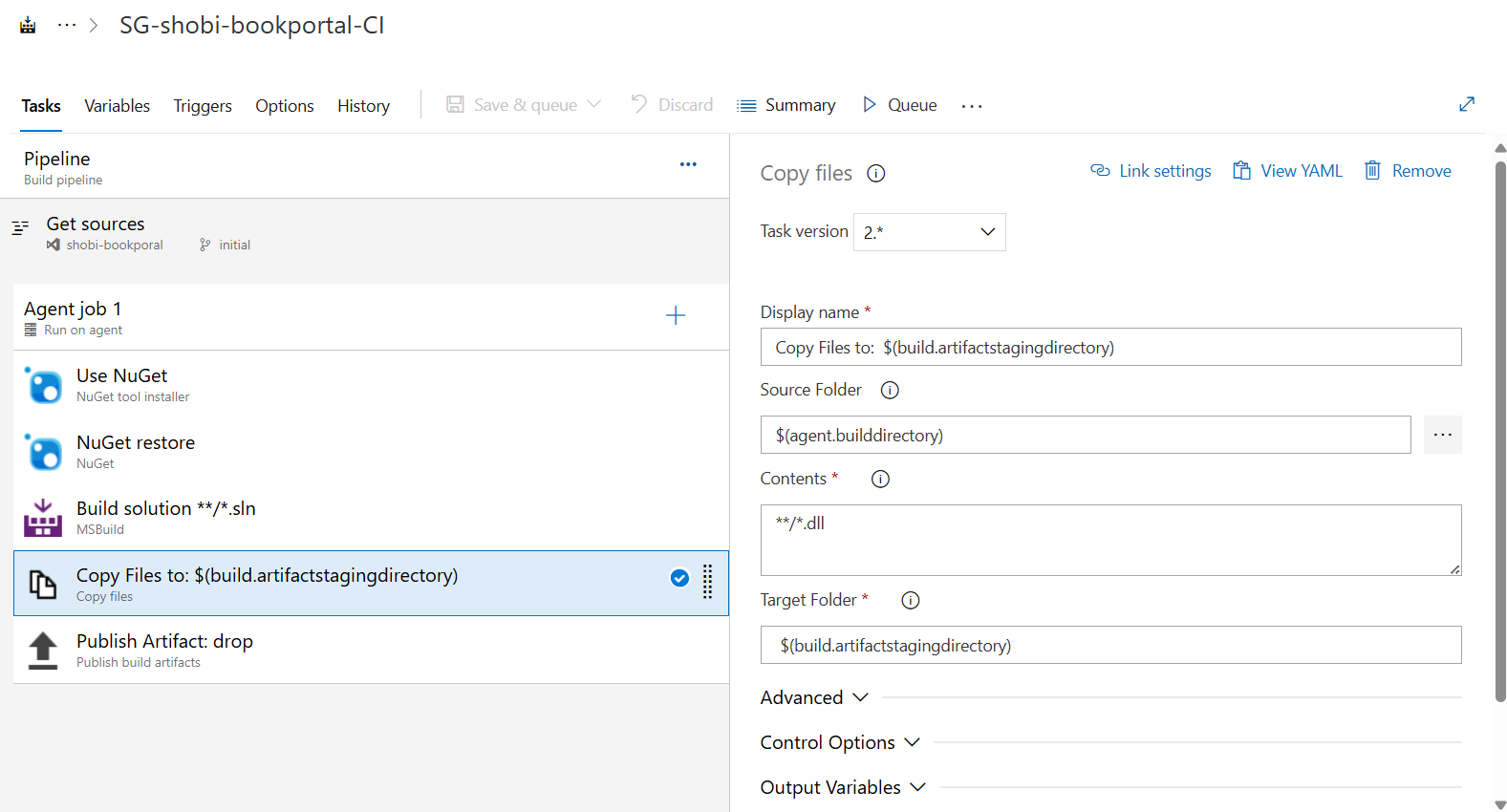


**✅ c. Build the Solution**

* Task: Visual Studio Build or .NET Core
  + For .NET Framework: Use Visual Studio Build
  + For .NET Core/5/6/7+: Use .NET Core CLI
* Settings:
  + Solution: \*\*/\*.sln
  + Configuration: Release

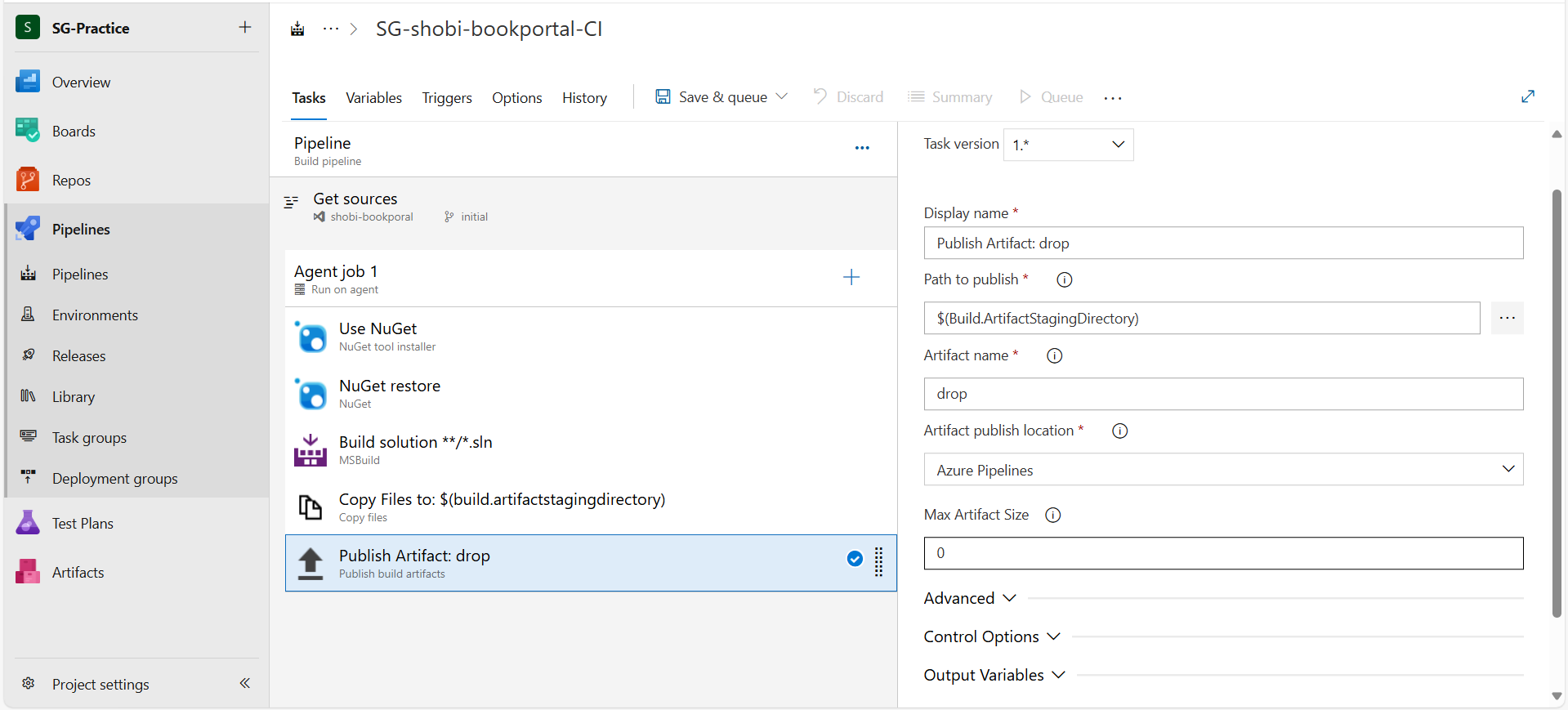


**✅ d.Copy files:**

****

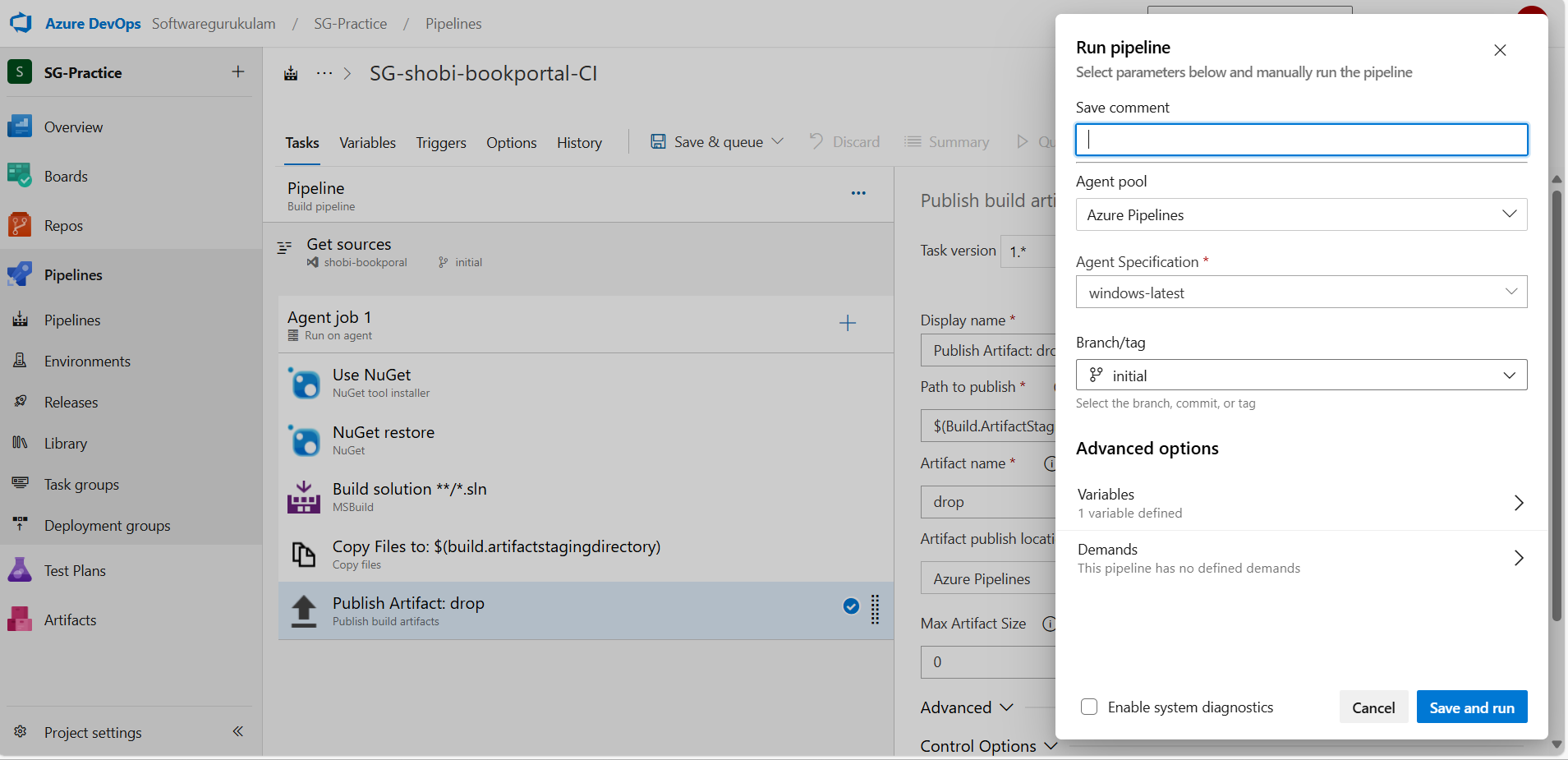
**✅ e. Publish Build Artifacts**

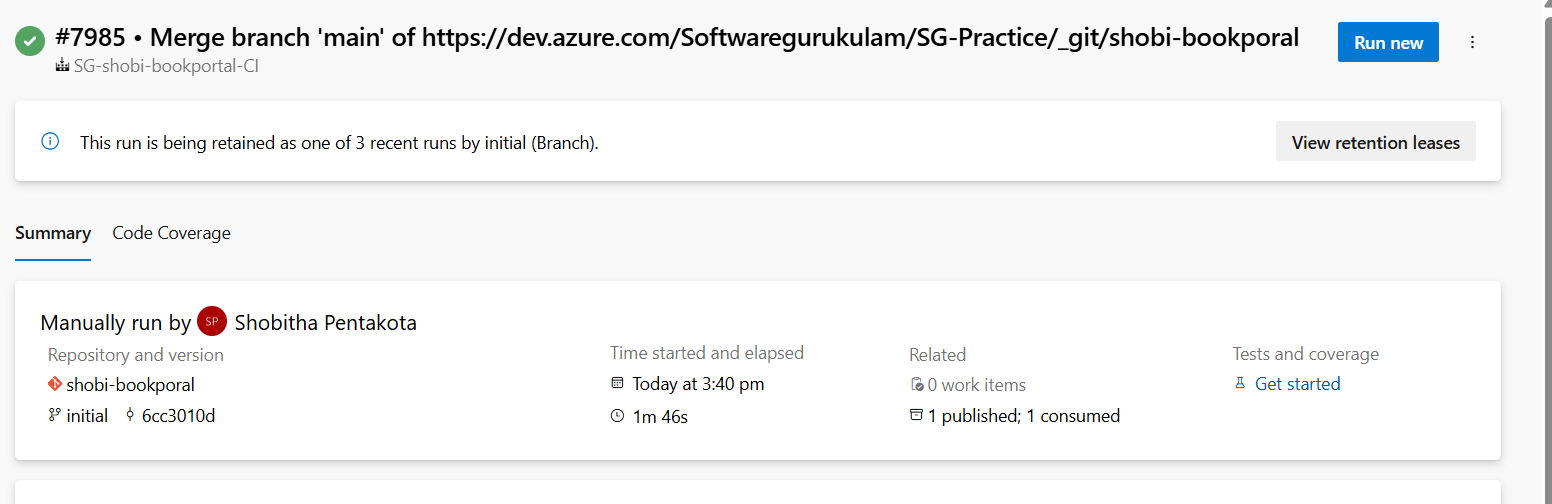
* Task: **Publish Build Artifacts**
* Path to publish: $(Build.ArtifactStagingDirectory)
* Artifact name: drop
* Artifact publish location: Azure Pipelines



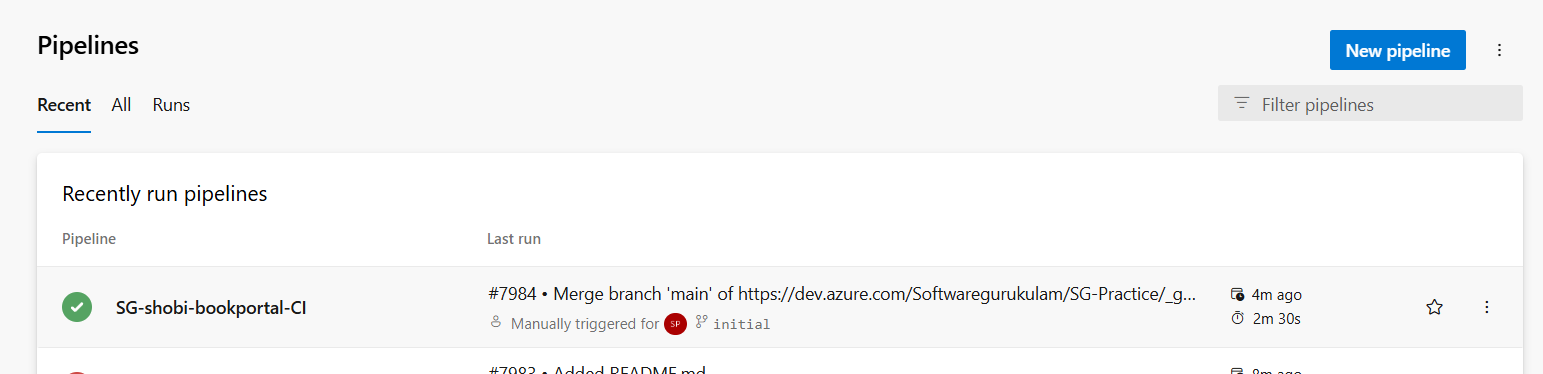
**🔹 6. Save and Queue**

* Click **Save & Queue > Save & Queue**
* Choose a branch to run the build on
* Click **Run**

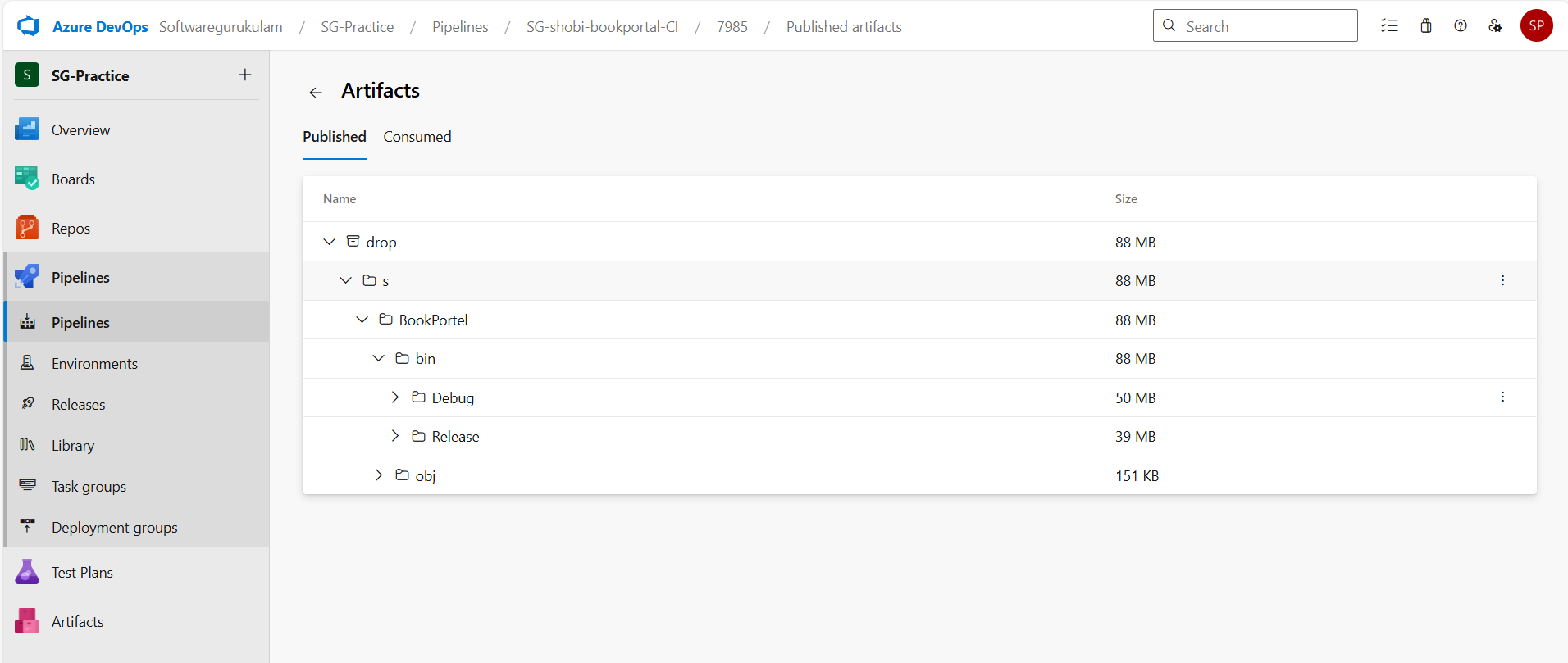








* Successfully build our pipeline(CI)……..>SG-shobi-bookportal-CI



* Artifacts published in bin folder…..dll files are store in bin folder.

**🔹 7. Enable Continuous Integration**

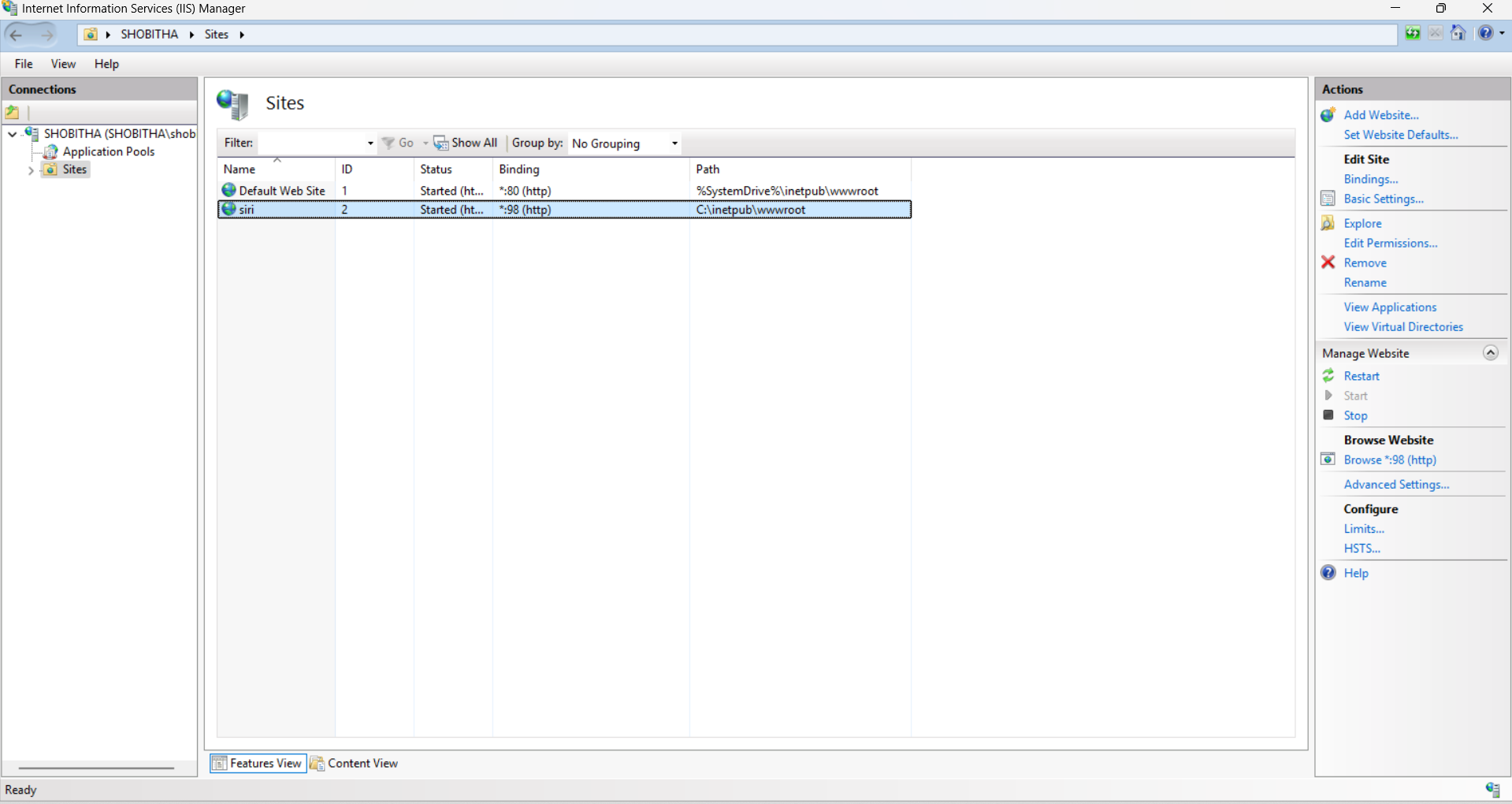
* Go to **Triggers tab**
* Enable **“Enable continuous integration”**
* Select the branch(es) to monitor
* This ensures the pipeline runs on each commit.

🚀 **CD Pipeline to Deploy .NET Application (Classic Editor):**

To set up Continuous Deployment (CD) for a .NET application using IIS on a Virtual Machine (VM) with a Deployment Group (typically via Azure DevOps), here's a complete step-by-step guide:

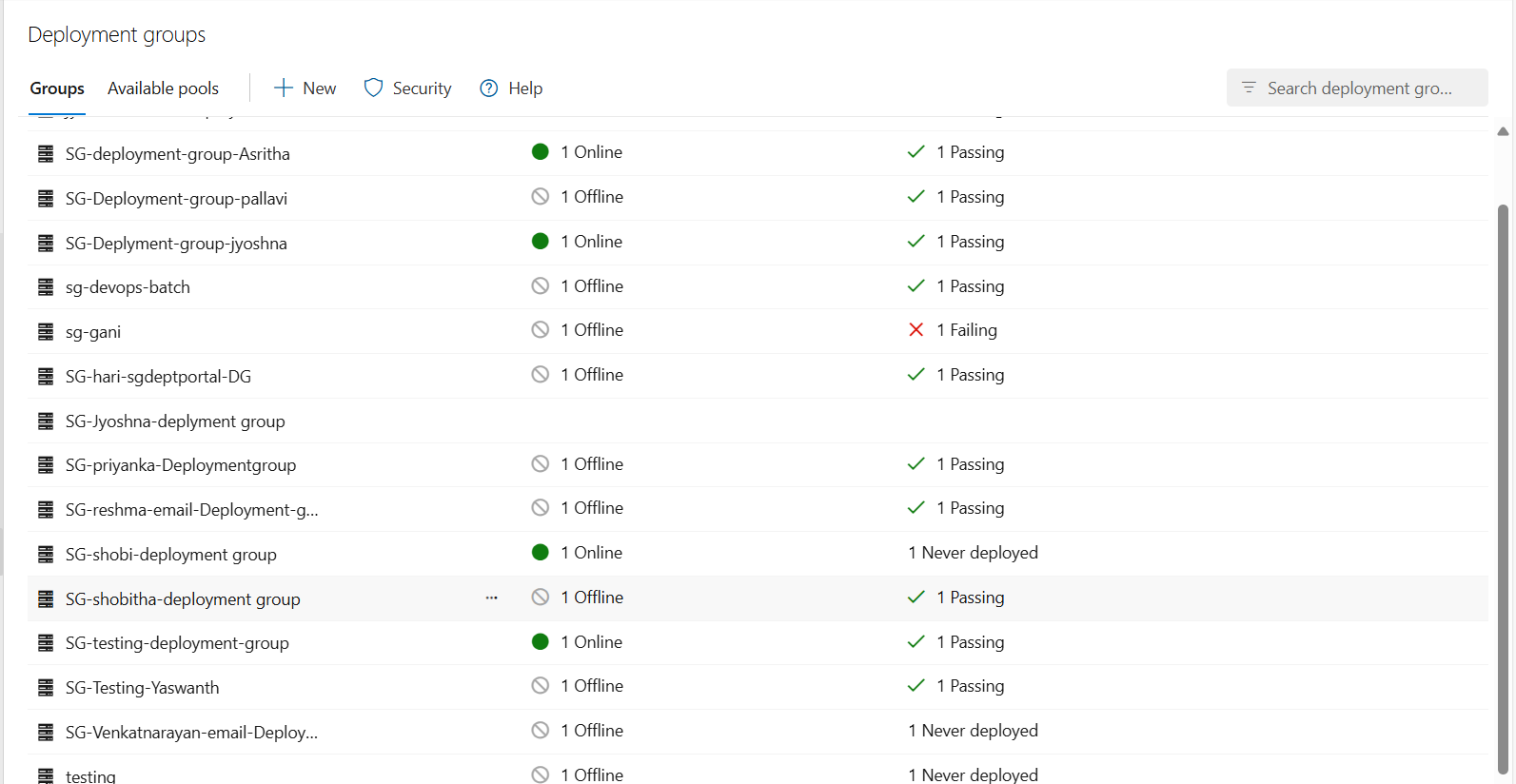
**🔧 Step 1: Configure IIS on the VM**

1. Open IIS Manager on the VM.
2. Create a new Website or Application:
   * Path: C:\inetpub\wwwroot\MyApp
   * Port: 80 (or any other, ensure firewall rule exists).
3. Ensure the .NET Hosting Bundle is installed (for ASP.NET Core apps).
4. Make sure permissions on the site folder allow Network Service or the agent account.

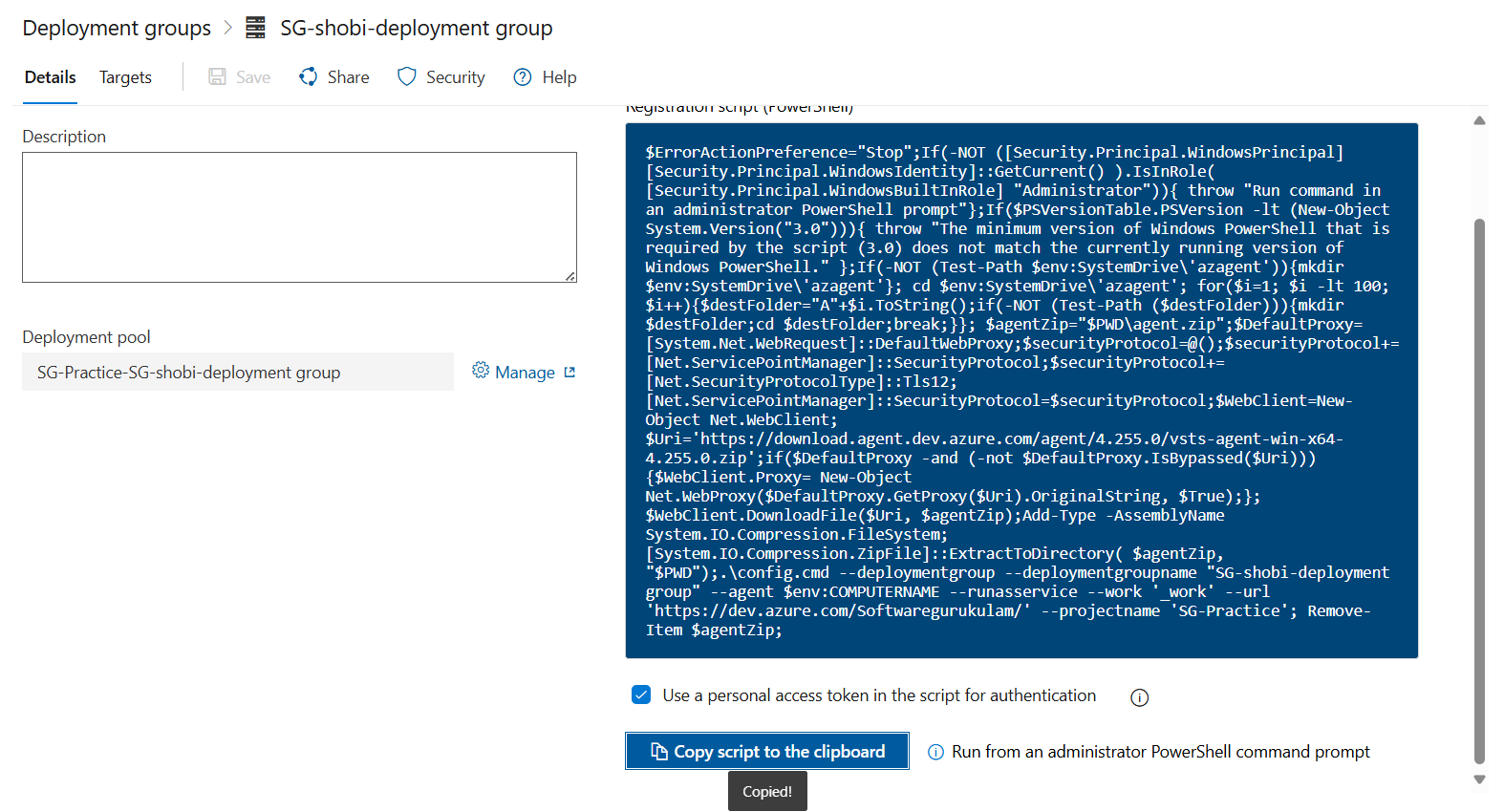


**🔗 Step 2: Set Up Deployment Group in Azure DevOps**

1. Go to Azure DevOps > **Pipelines > Deployment Groups**.
2. Click **+ New** > Give it a name, e.g., IIS-VM-Group.



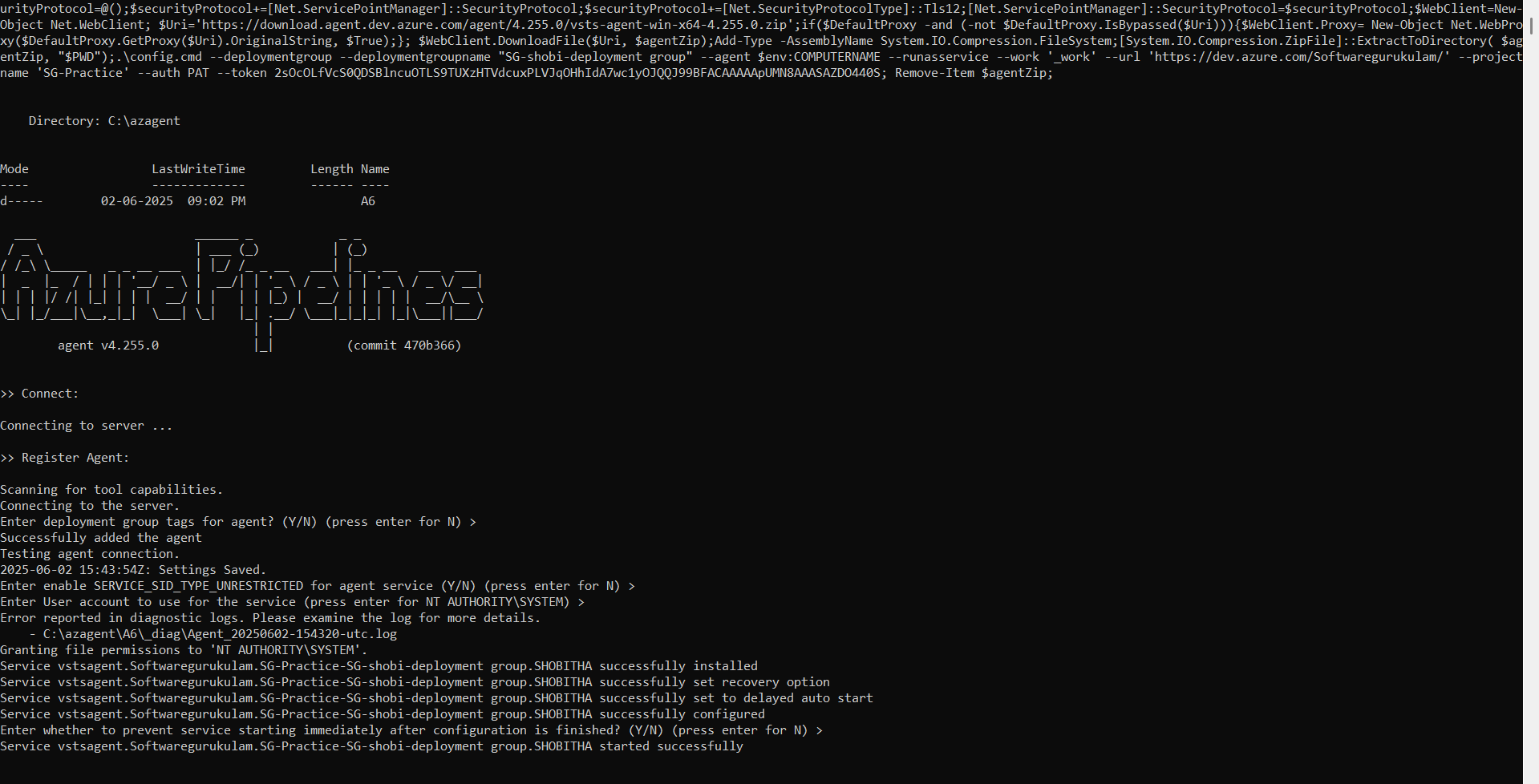
3.Copy the **PowerShell registration script**.



4.Run it as **Administrator on the VM**.

* + This installs the Deployment Group agent and connects it to Azure DevOps.

5.After successful installation, you should see the VM in the deployment group



**🏗 Step 3: Create Build Pipeline (Classic Editor)**

1. Go to **Pipelines > Pipelines**.
2. Click **+ New pipeline**.
3. Choose:
   * Where your code is (e.g., Azure Repos Git).
   * Choose **Use the classic editor** (link at bottom).
4. Select your repo and branch, then click **Continue**.
5. Choose **.NET Core** or **Empty Job**, depending on your app.
6. Add the following tasks:

**Tasks in Build Pipeline**

**Task 1: Restore NuGet packages**

* Task: **NuGet Tool Installer** (optional)
* Task: **NuGet Restore**
  + Path to .sln file

**Task 2: Build**

* Task: **.NET Core or MSBuild**
  + Path to .csproj or .sln
  + Configuration: Release

**Task 3: Publish**

* Task: **.NET Core Publish**
  + Output folder: $(Build.ArtifactStagingDirectory)\publish

**Task 4: Publish Artifact**

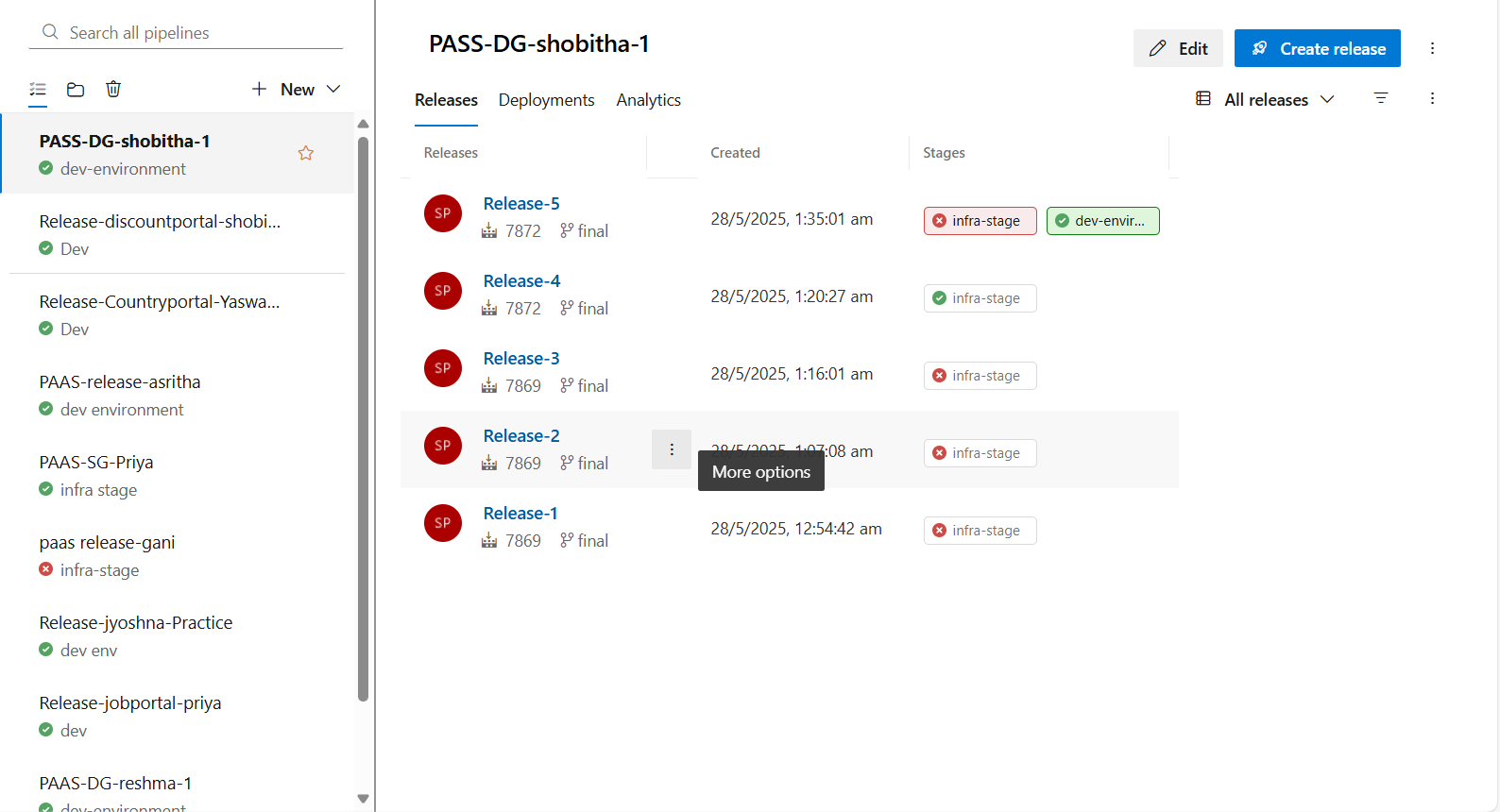
* Task: **Publish Build Artifacts**
  + Path: $(Build.ArtifactStagingDirectory)\publish
  + Artifact name: drop

1. Save & Queue the build.
2. After successful build, verify the artifact (drop) is published.

9.we already bulid the pipeline………….

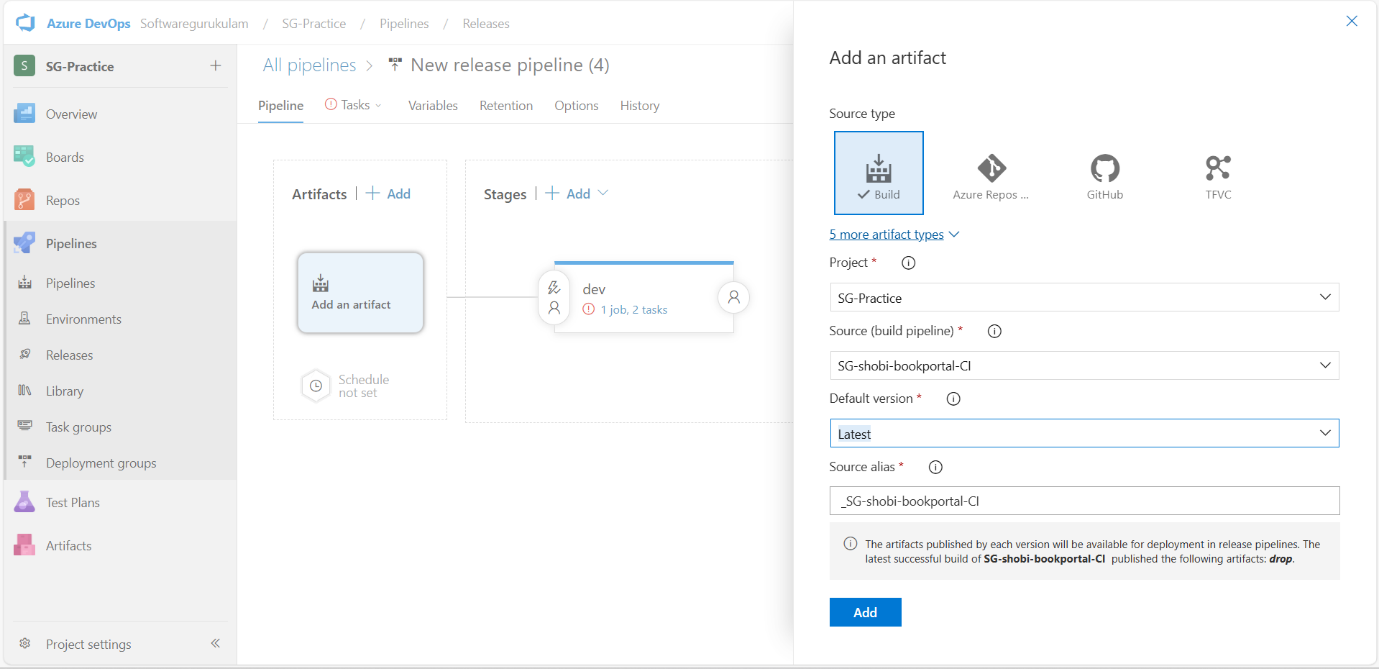
**🔄 Step 4: Create Release Pipeline (Classic Editor)**

1. Go to **Pipelines > Releases**.
2. Click **+ New pipeline**.



3.Click **Add an artifact**:

* + Source: Your build pipeline
  + Default version: Latest
  + Source alias: drop (or leave default)

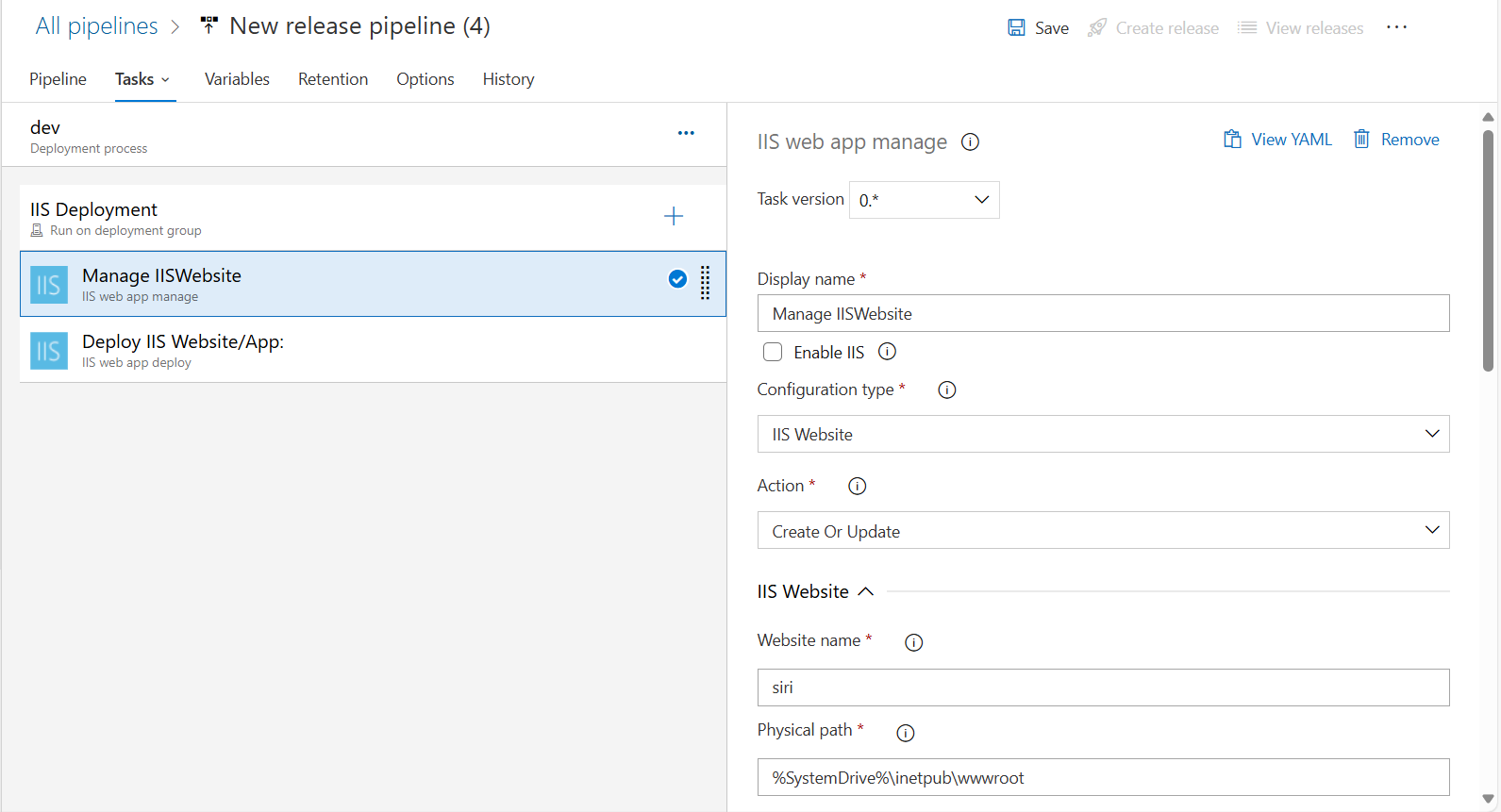


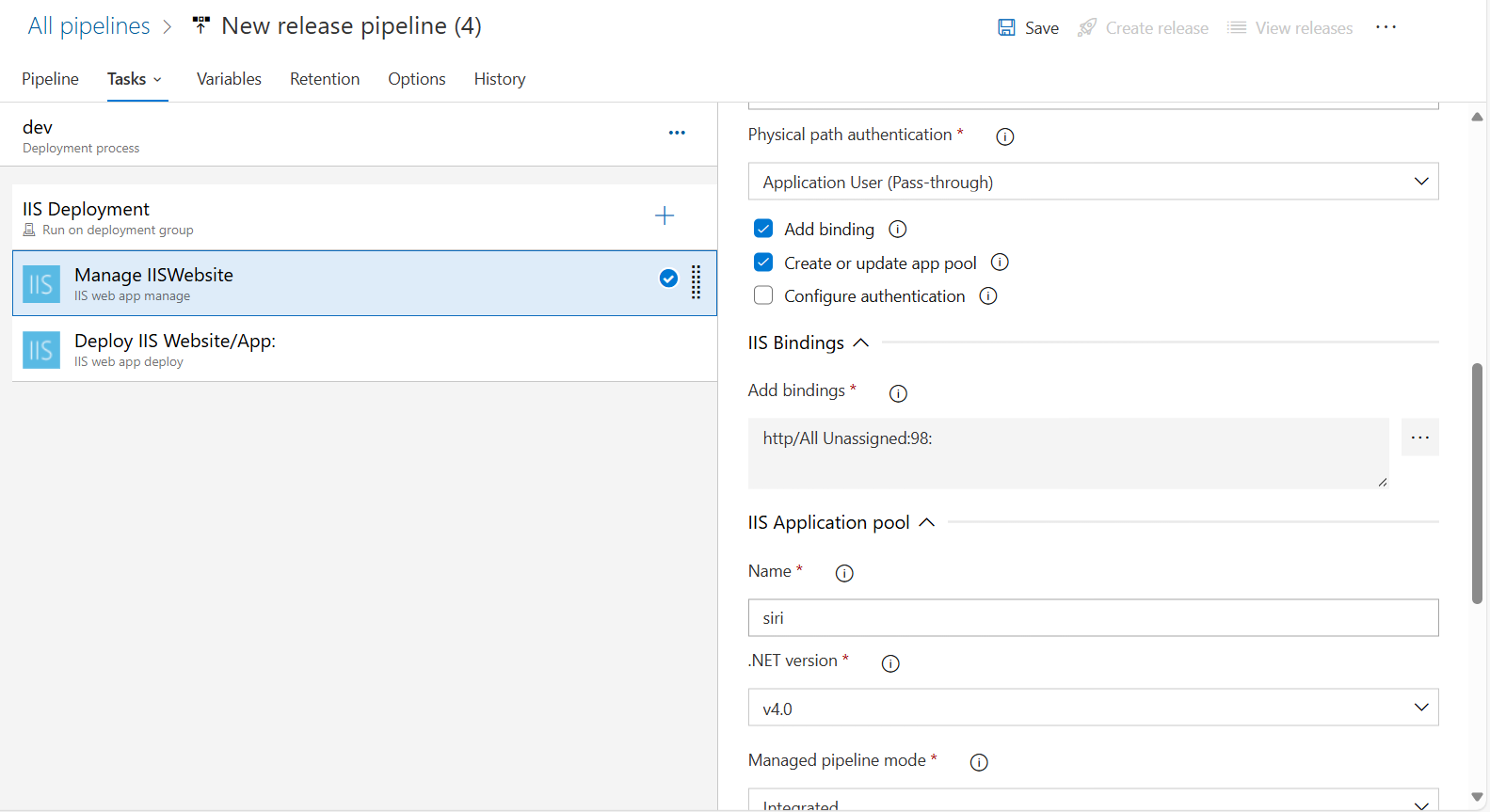
**Step5: Deploy to IIS (Deployment Group Job)**

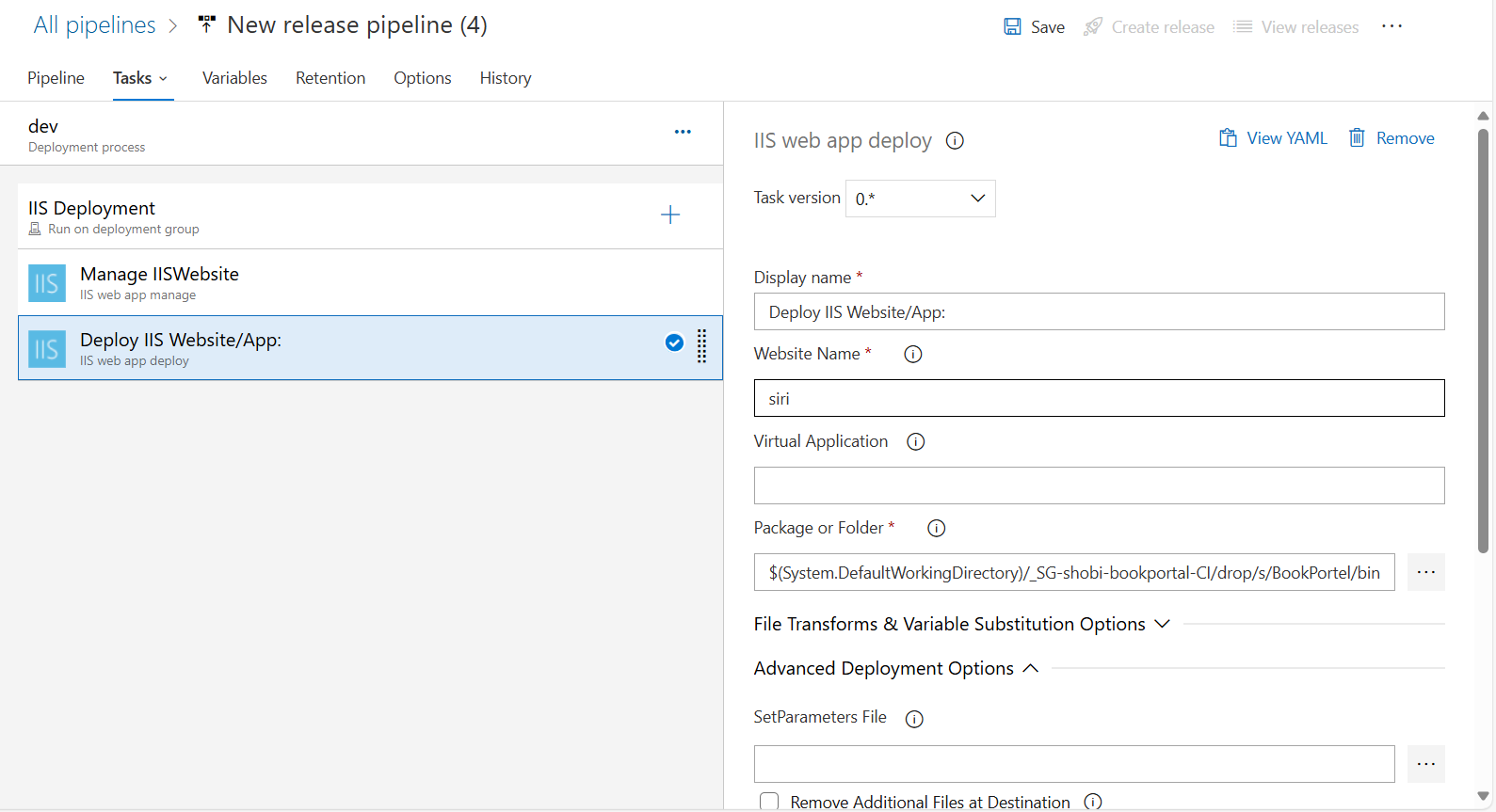
1. Click **Add a Stage** > Choose **Empty Job**.
2. Click on **Stage name** → rename to Deploy to IIS.
3. Click on the **job** link.
4. Change **Agent job type** to **Deployment group job**.
5. Select your Deployment Group (e.g., IIS-VM-Group).

TASK:

1)manage IIS website:

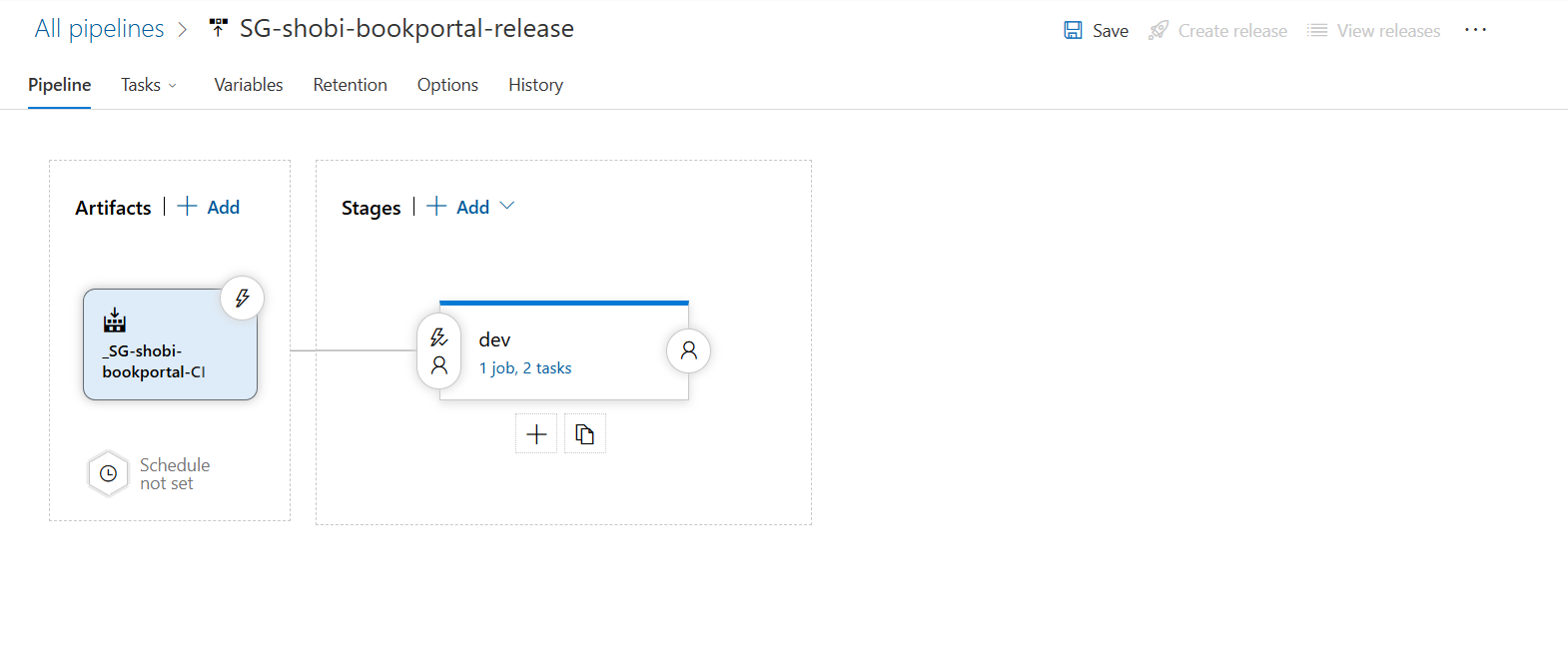


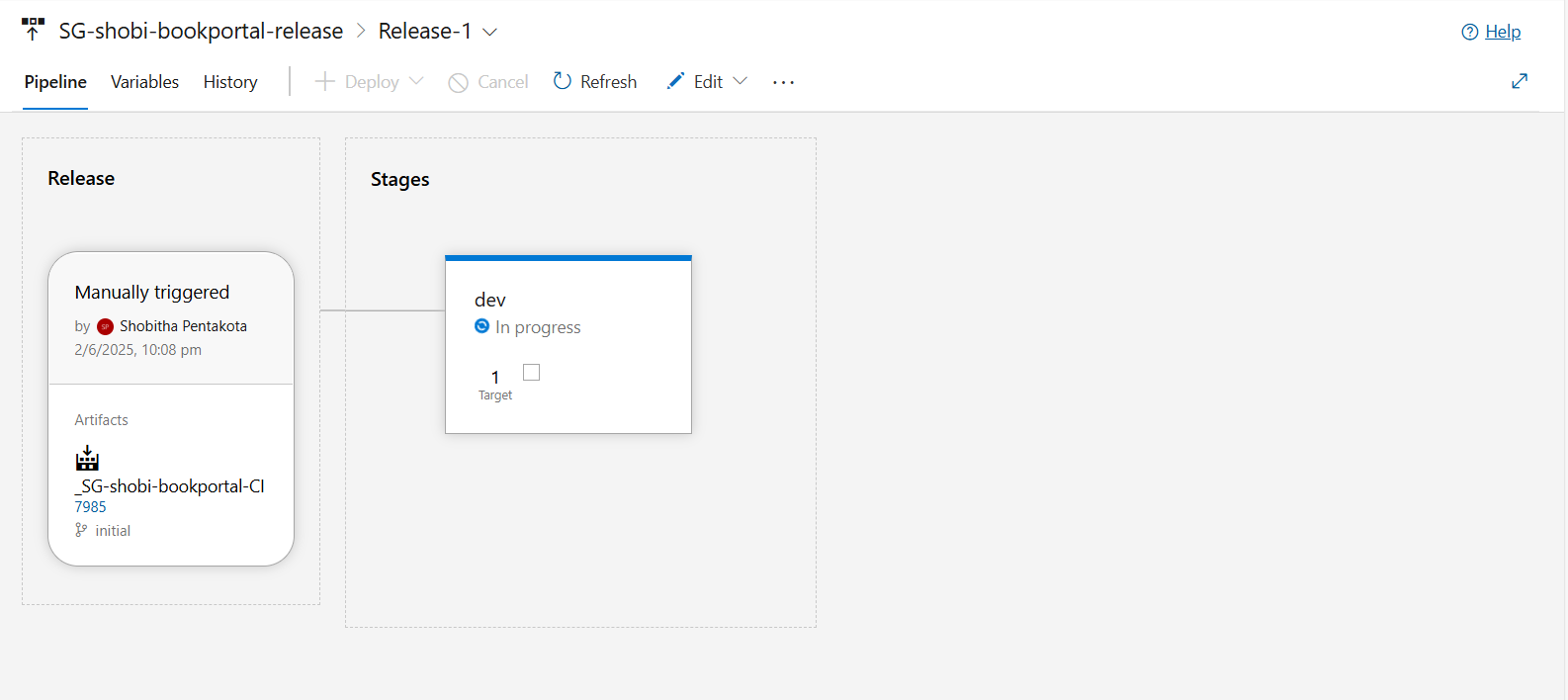


2) 

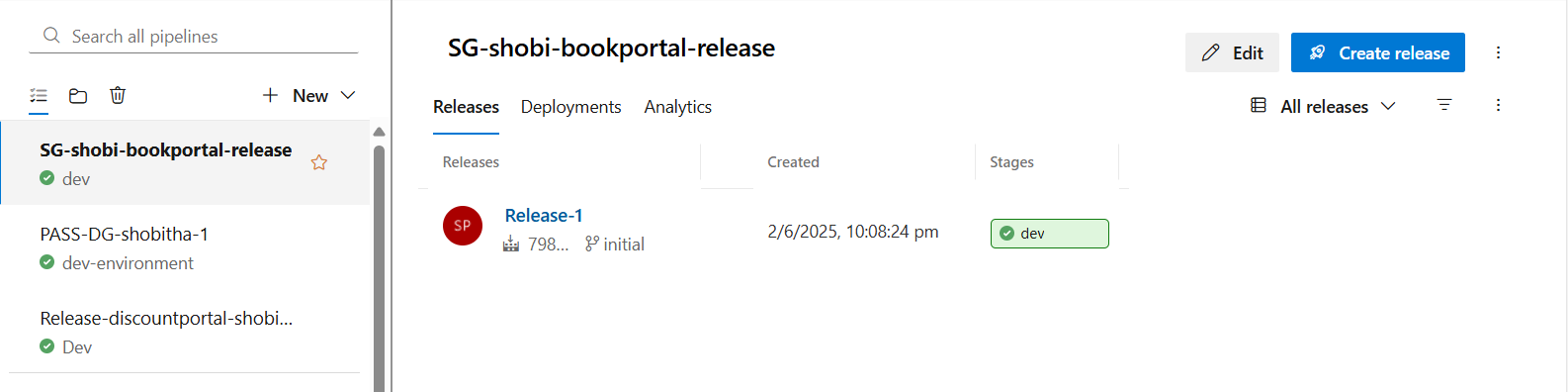
**🔁 Final Steps**

1. Save the pipeline.
2. Click **Create Release** → Select the stage → Deploy.
3. Open browser to the VM’s IP address or domain to verify deployment.





\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Release successful \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*



* **You want to access it from your host/local machine** (outside the VM) using the name <http://siri:98>.

