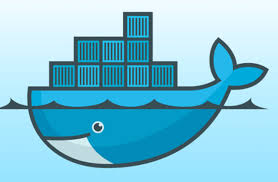
Docker

**🐳 What is Docker?**

Docker is an open-source platform used to build, run, and manage containers. Containers package an application and all its dependencies into a single image, ensuring it runs consistently across different environments.

****

**🧱 Why Docker?**

| **Benefit** | **Explanation** |
| --- | --- |
| Portability | Run anywhere (dev, test, prod) |
| Isolation | Apps run in separate containers |
| Lightweight | Shares OS kernel; faster than VMs |
| Scalable | Ideal for microservices architecture |
| CI/CD Friendly | Easily integrated with DevOps pipelines |

**📦 Docker Concepts:**

**Docker Image**: Set of instructions to create the container in docker platform.

* Built using **docker build -t myimage .**

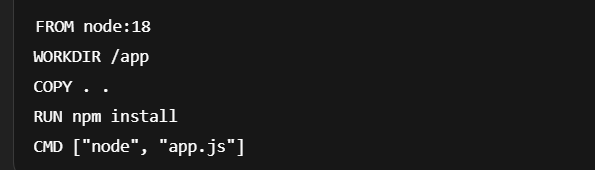
**Docker Container**: A running instance of an image.

* Run it with **docker run -it myimage**

**Docker Engine**: The service that manages containers, installed on the host system.

* On Linux, it runs as dockerd; on Windows/macOS, it uses WSL or a VM underneath.

**Dockerfile**: A text file with step-by-step instructions to create a Docker image.



**Docker Registry**: A server to store and share Docker images. Default is Docker Hub.

**Nginx in Docker**: You can run Nginx directly from Docker Hub with:

* + docker run -d -p 8080:80 nginx

**Docker Hub** is the official cloud-based container image registry provided by Docker. Think of it as a GitHub for Docker images — a place to store, share, and discover container images.

**🔍 What You Can Do With Docker Hub**

1. **Pull images** (download) to run containers on your machine
   * Example: docker pull nginx
2. **Push images** (upload) that you create to Docker Hub
   * Example: docker push yourusername/myapp
3. **Host public or private images**
   * Public: anyone can see/download
   * Private: only you or your team can access
4. **Use official images**

Verified images from trusted sources like nginx, mysql, node, ubuntu, etc.

1. **Automate builds** (CI/CD integration)
2. **Tag versions** of your images (like v1, latest, etc.)

**🔧 Docker Architecture**

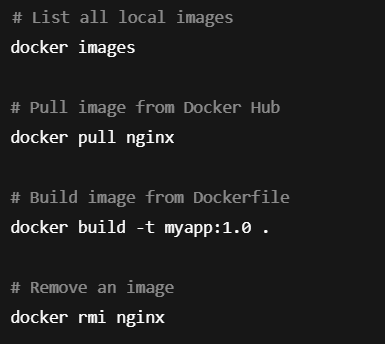
* **Docker Client** – Runs commands.
* **Docker Daemon** – Builds/runs containers.
* **Docker Images** – Templates for containers.
* **Docker Containers** – Running instances.
* **Dockerfile** – Instructions to create an image.
* **Docker Hub** – Public image registry.

🛠️ **Common Docker Commands:**

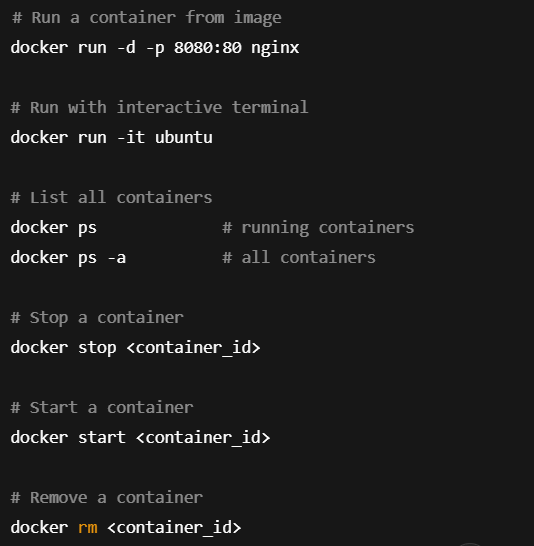
**Installation Check:**

****

**🔹 Docker Image Commands:**

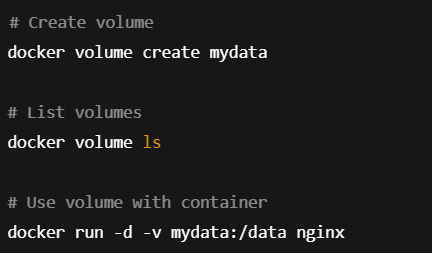
****

**🔹 Docker Container Commands**

****

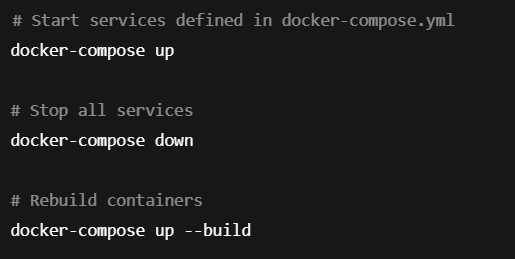
**🔹 Docker Volume Commands:**

* Volumes are used to persist data even after the container stops or is removed.
* Ideal for databases, logs, or anything that should not be lost between container restarts.

****

**🔹 Docker Compose Commands:** Docker Compose is a tool for defining and running multi-container Docker applications

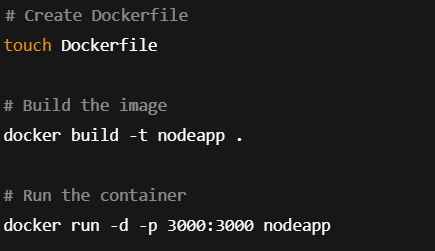
docker-compose.yml is used to define and run multi-container Docker applications.

****

**🧪 Useful Debug Commands**

****

📁 **Docker Use Case Example: Node App:**

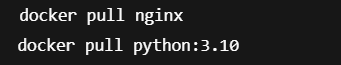


🛠️ **Common Docker Hub Commands:**

1. Login to Docker Hub:



1. Pull an image from Docker Hub:



1. Run a container from an image:



1. Tag your image before pushing:



1. Push your image to Docker Hub:



1. Search for images:

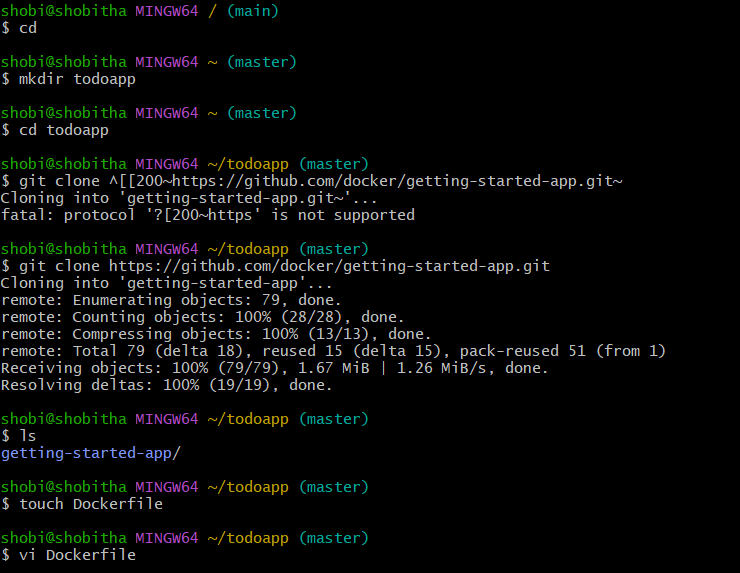


**Docker Swarm** is Docker's native clustering and orchestration solution for managing a group of Docker Engines as a single, virtual system. It allows you to easily deploy, manage, and scale containerized applications across multiple machines (nodes) in a cluster.

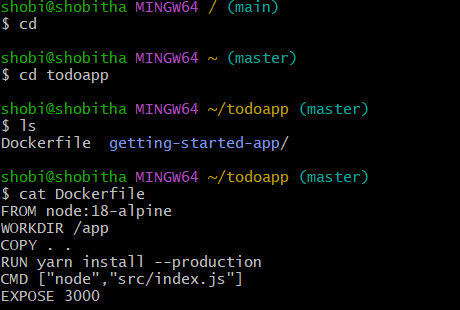
* Docker Hub = Cloud storage for Docker images
* You use docker push and docker pull to interact with it
* Useful for sharing, deploying, and collaborating on containerized applications.

✅ **Step-by-Step Process to Dockerize and Deploy Your To-Do List App:**

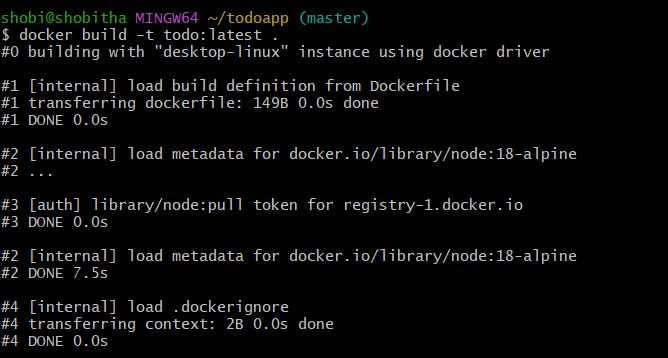
**✅ Step 1: Clone the Repository from GitHub**

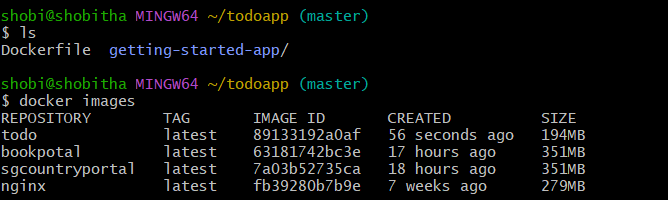


✅ **Step 2: Create a Dockerfile:**

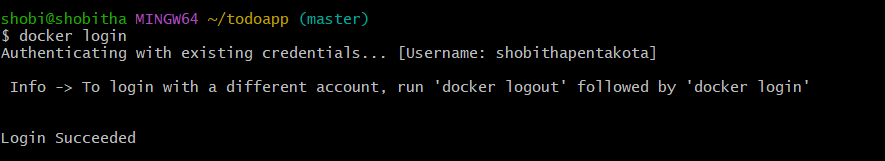


✅ **Step 3: Build the Docker Image:**

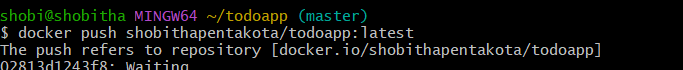


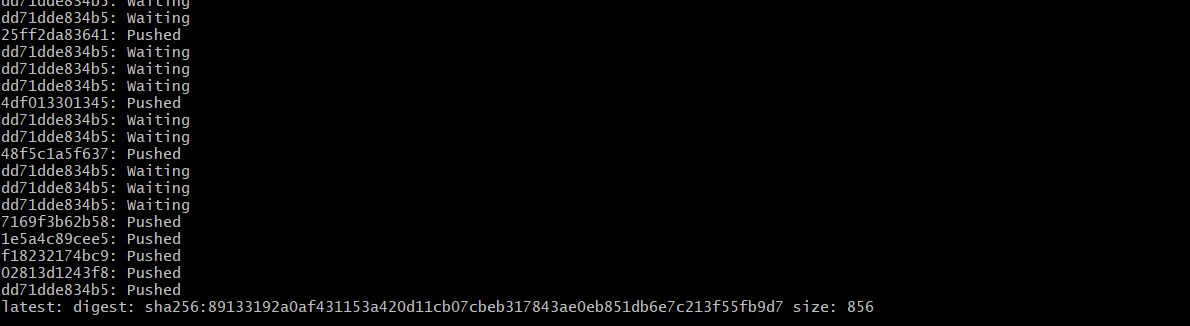


✅ **Step 4: Log In to Docker Hub:**

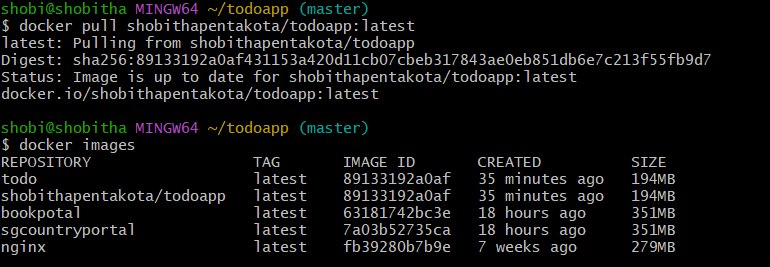
****

✅ **Step 5: Push the Image to Docker Hub:**

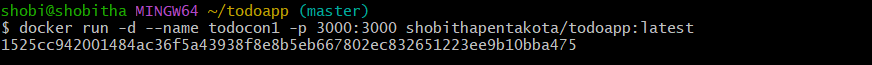




✅ **Step 6: Pull the Image on Any Machine:**

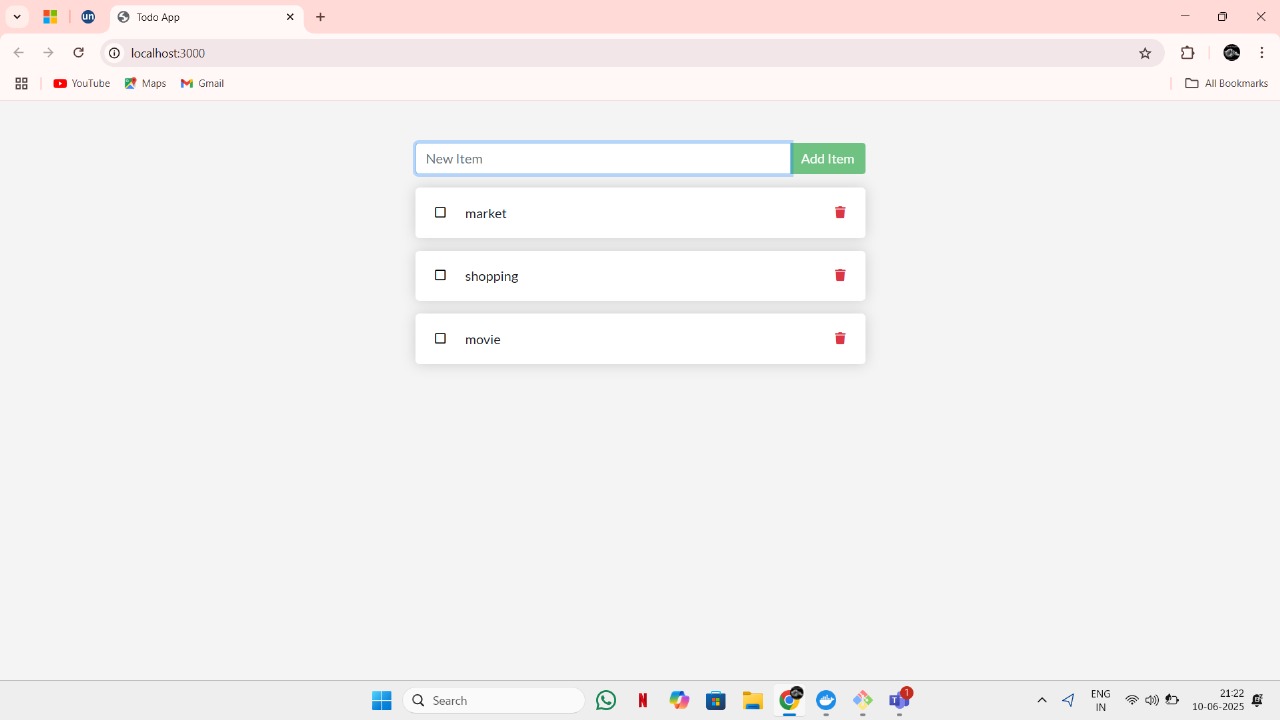


✅ **Step 7: Run the Docker Container:**



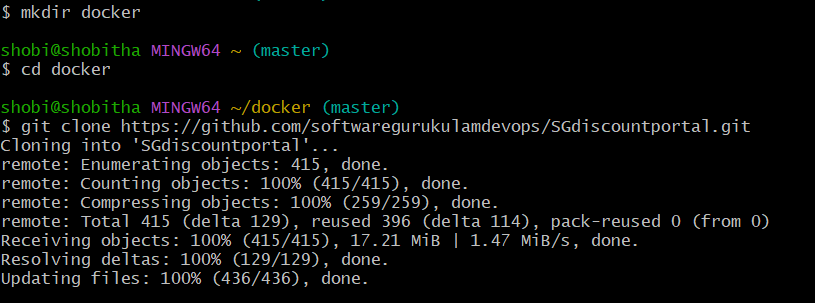
Then open your browser:

<http://localhost:3000>

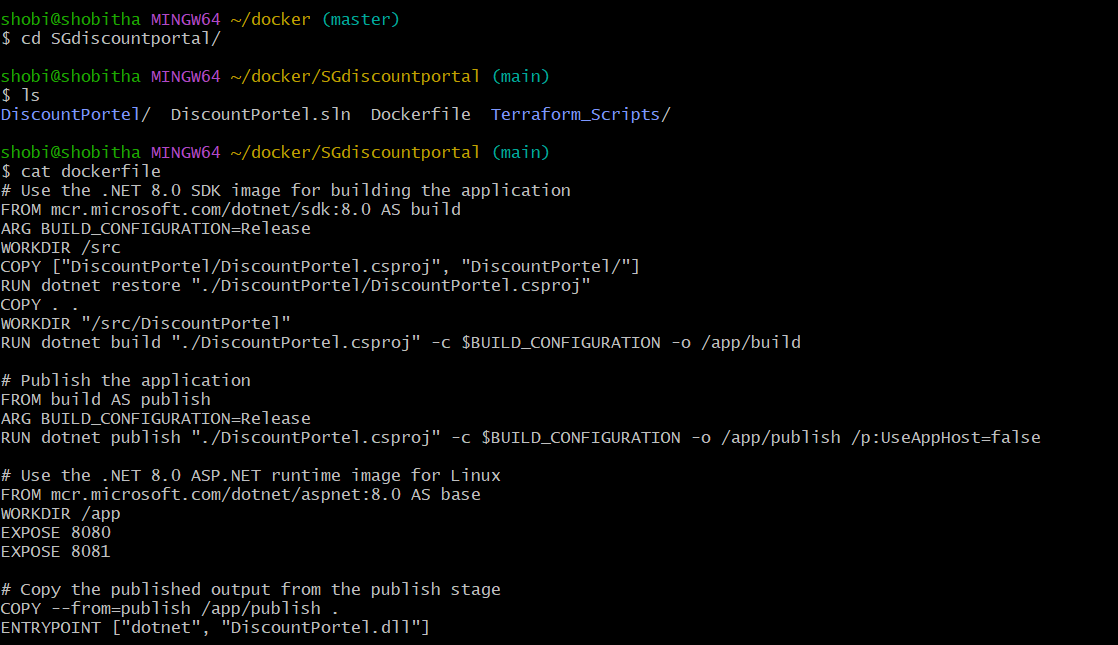


* **Step-by-step guide for deploying a .NET application using Docker**.

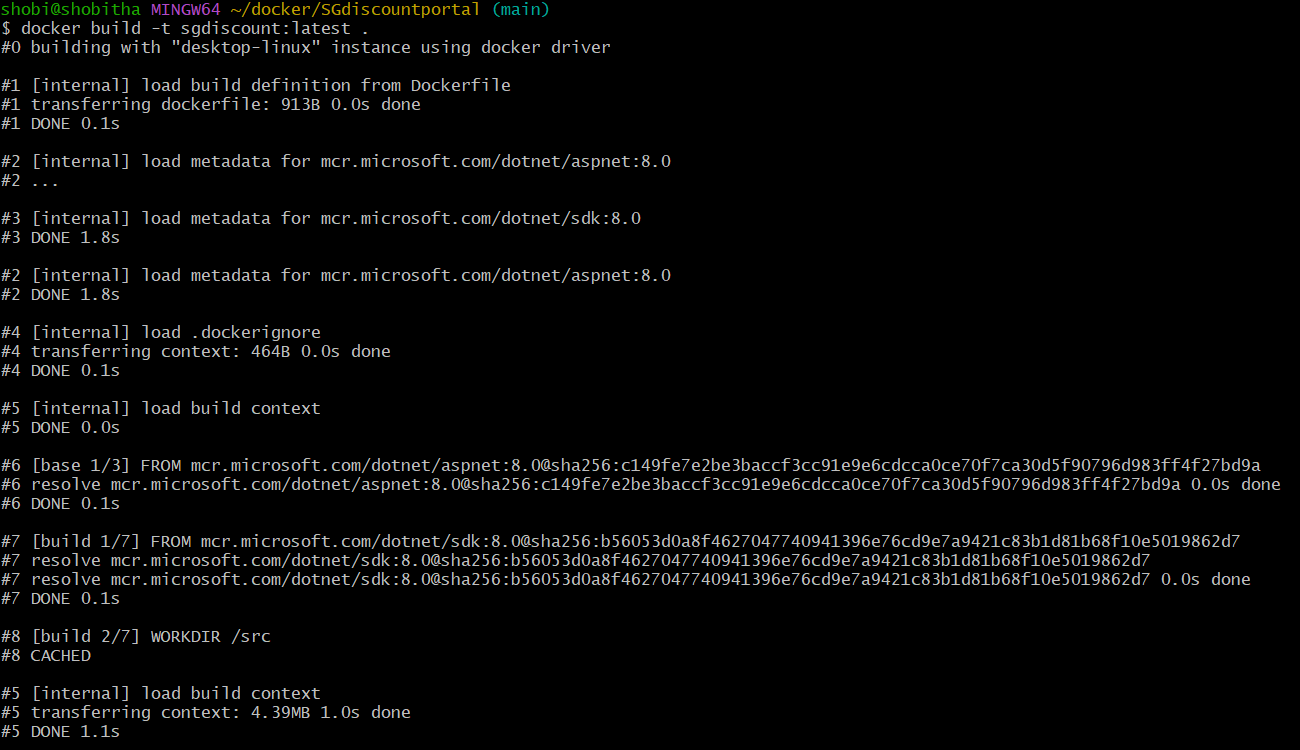
**✅ Step 1: Clone the Repository from GitHub**

****

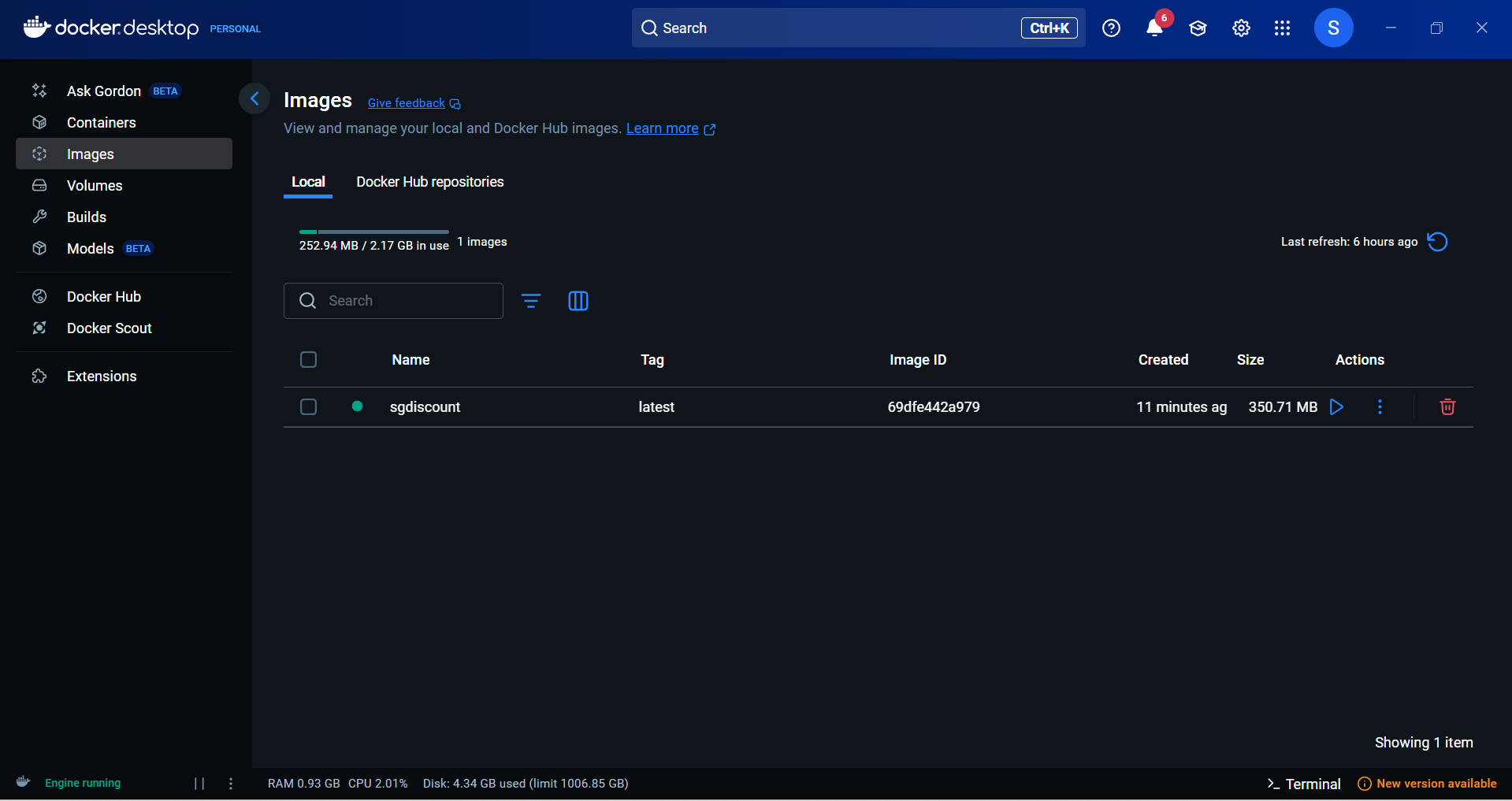
✅ **Step 2: Create a Dockerfile:**



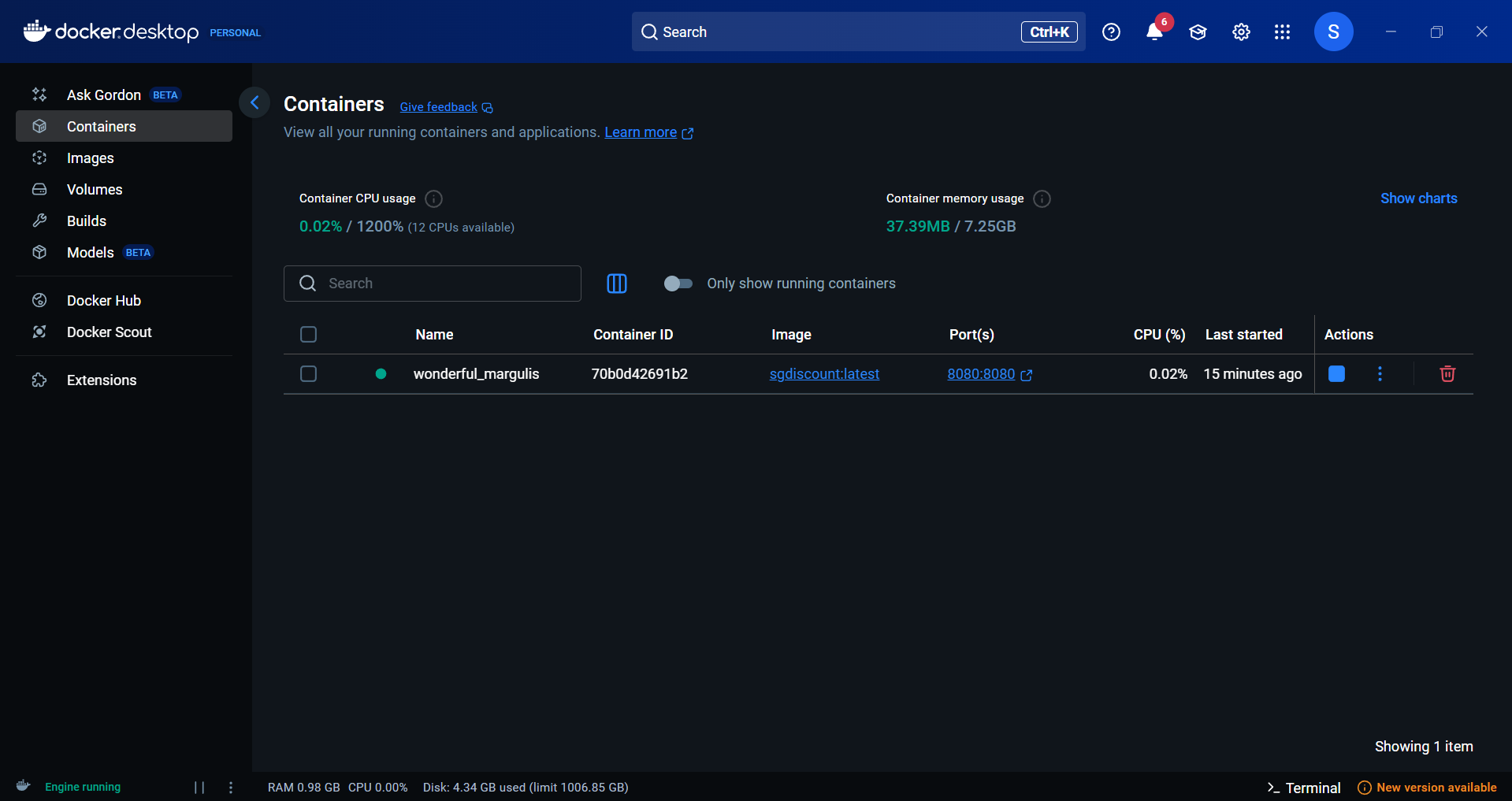
✅ **Step 3: Build the Docker Image:**



✅ **Step 4**: goto docker desktop check image is created or not.

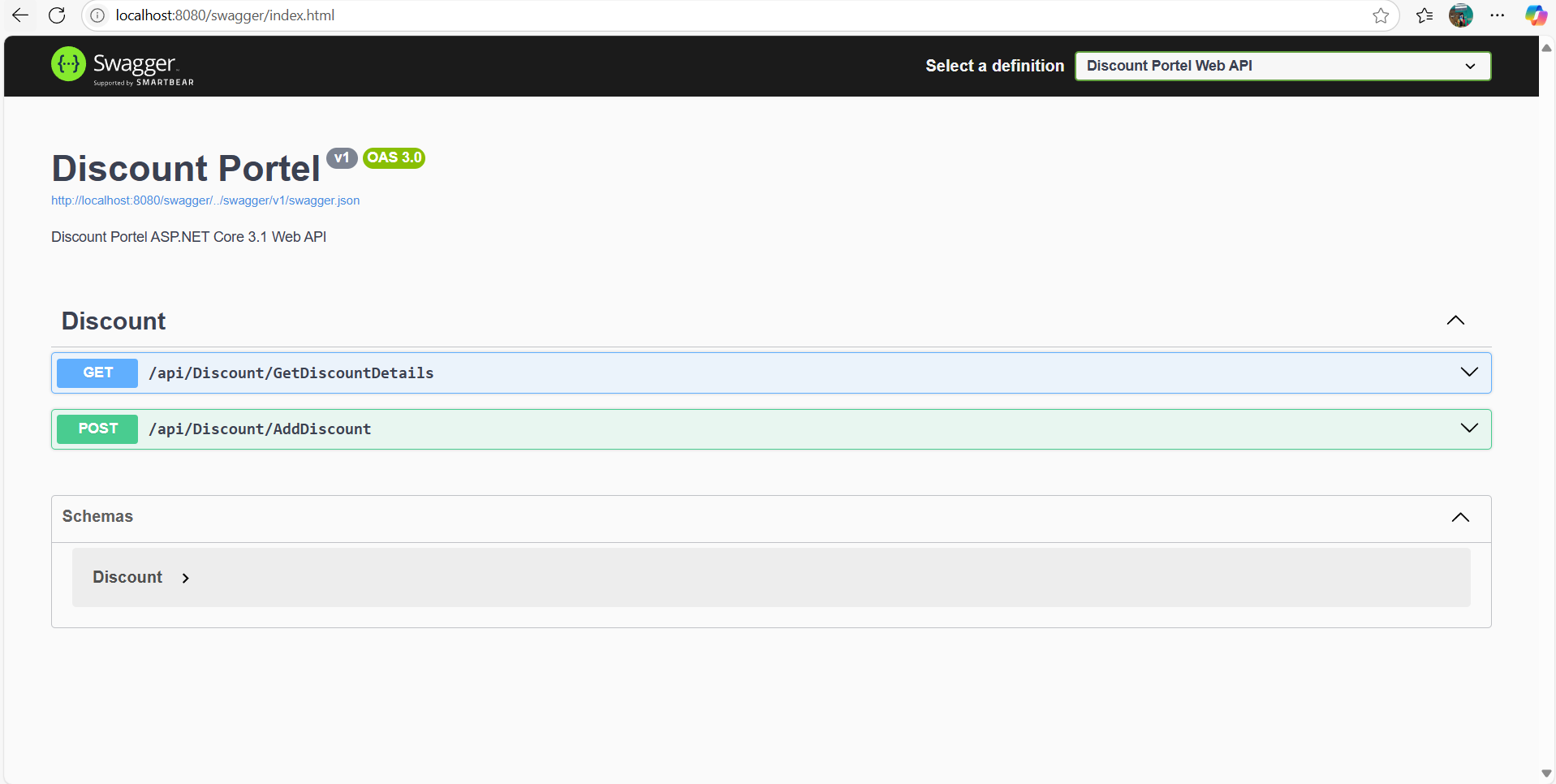


✅ **Step 5**:run the container by clicking action symbol. Port number should be 8081,8080.

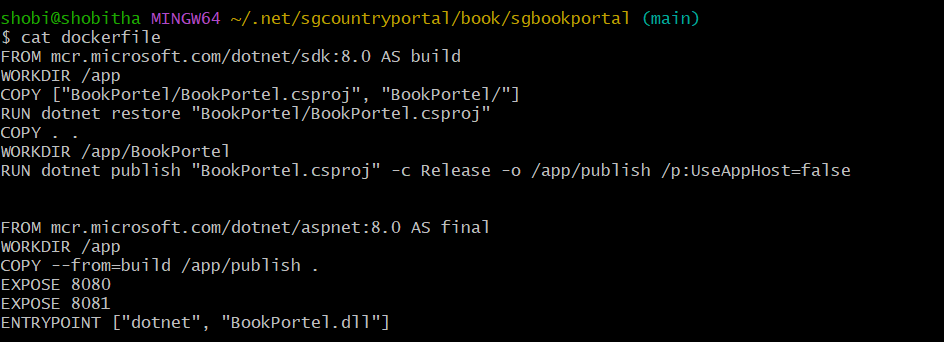


✅ **Step 6:**click on port….localhost:8080/swagger

We deploy our .net application successfully .



**Docker file code for .net application :**

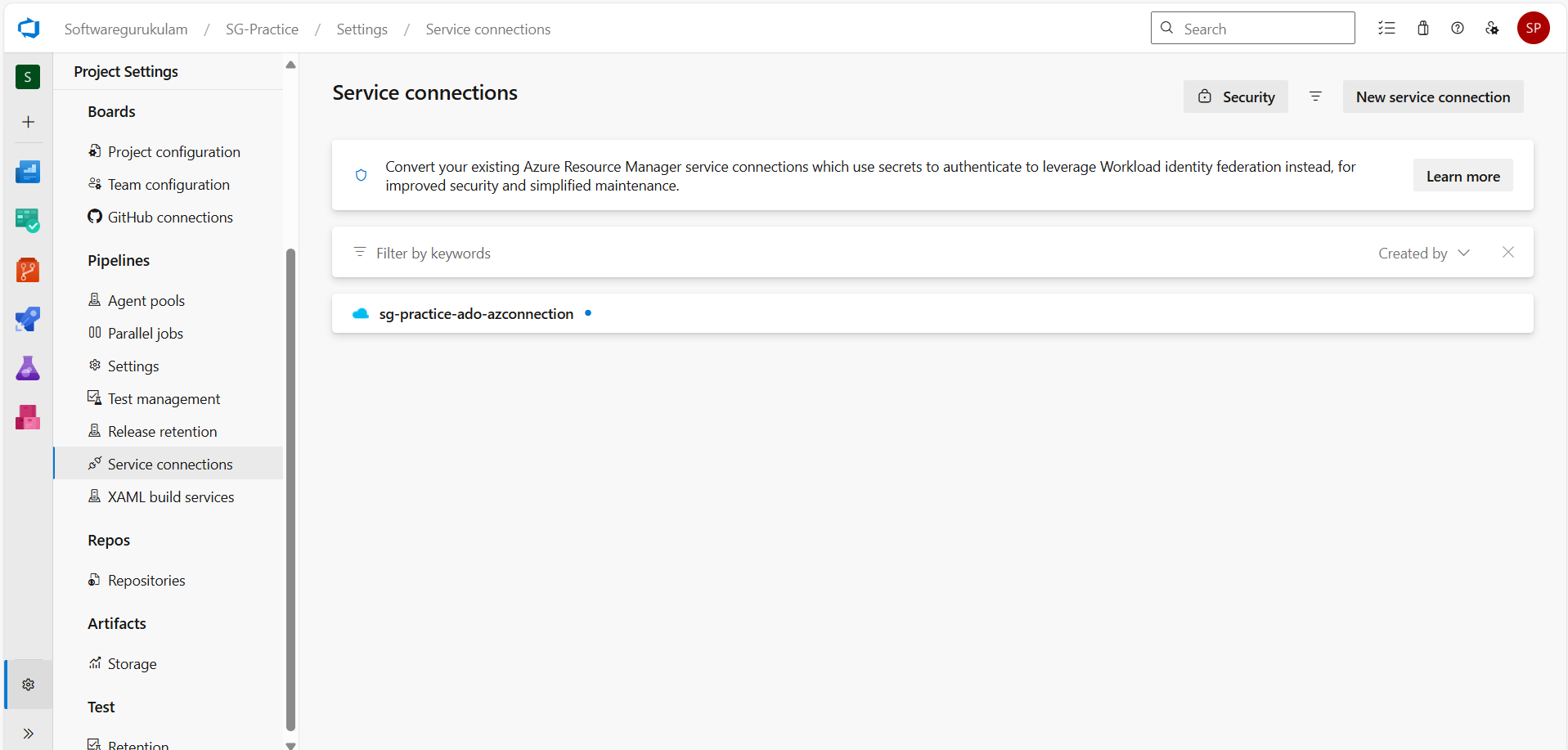
****

**TASK:**

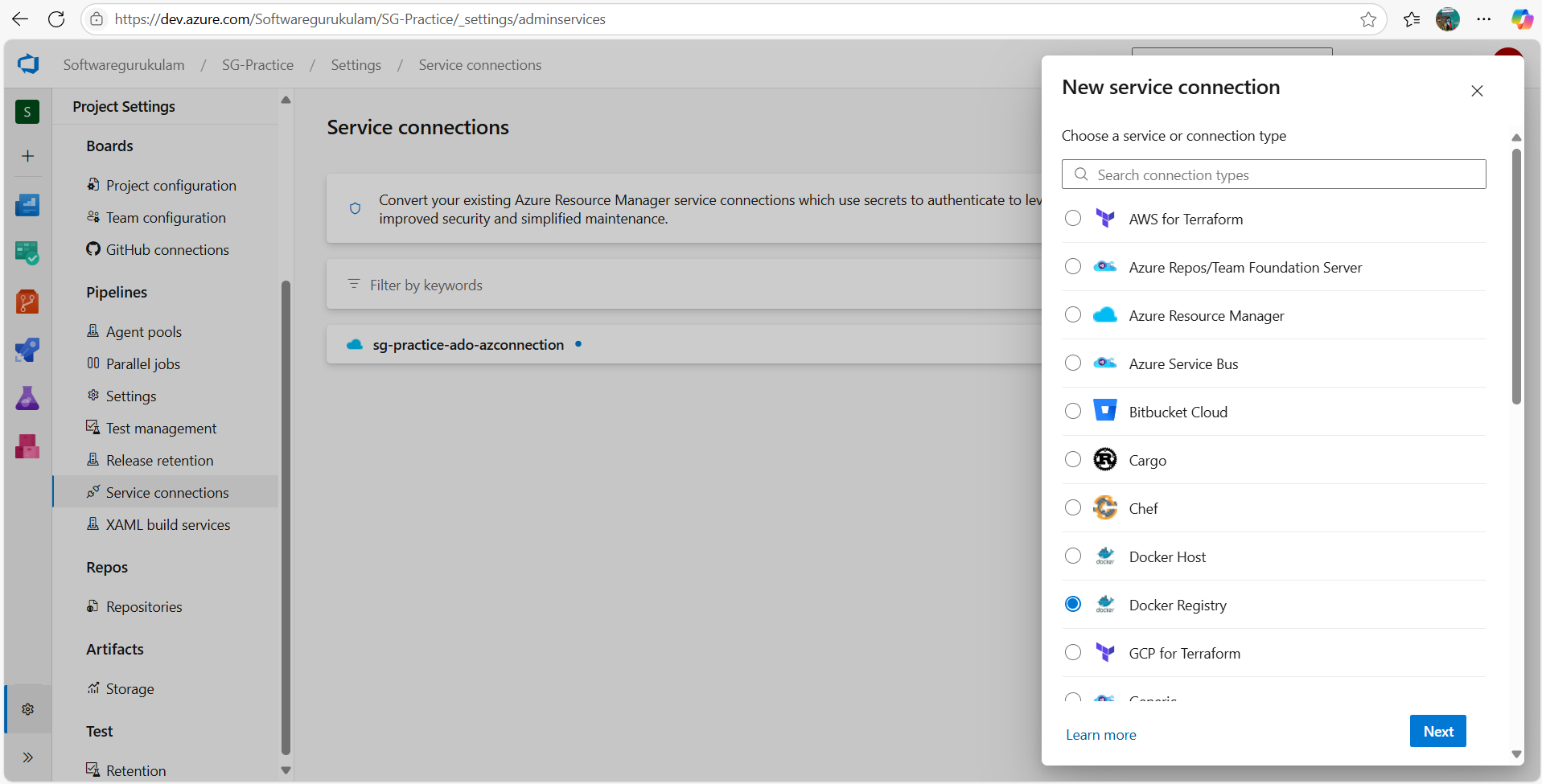
Step-by-step guide to build a Dockerfile in Azure DevOps using the Classic Editor and push the image to Docker Hub using a Service Connection:

**🔑 Step 1: Create a Docker Hub Service Connection**

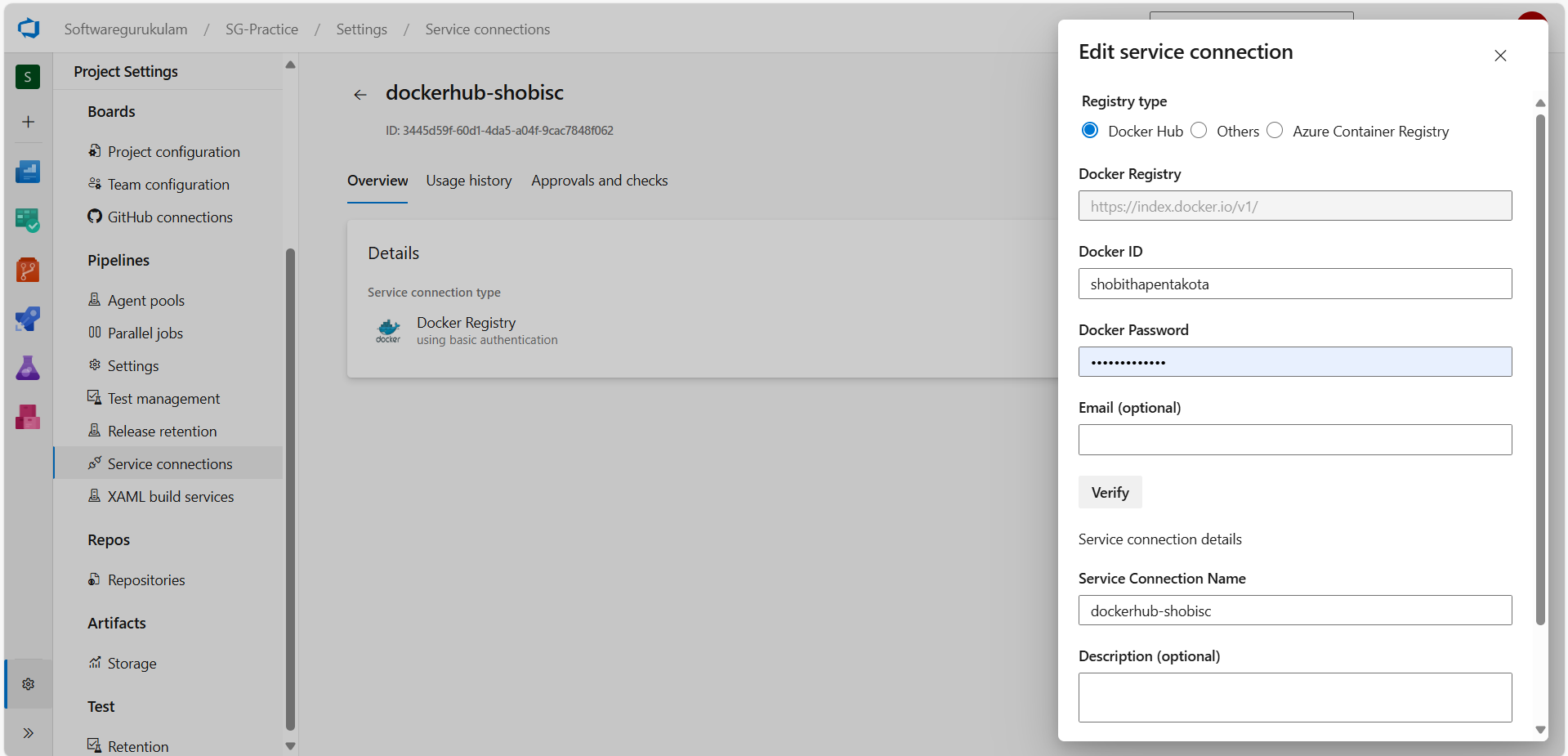
1. Go to your Azure DevOps project.
2. Go to Project Settings > Service Connections.



1. Click New service connection > Docker Registry.



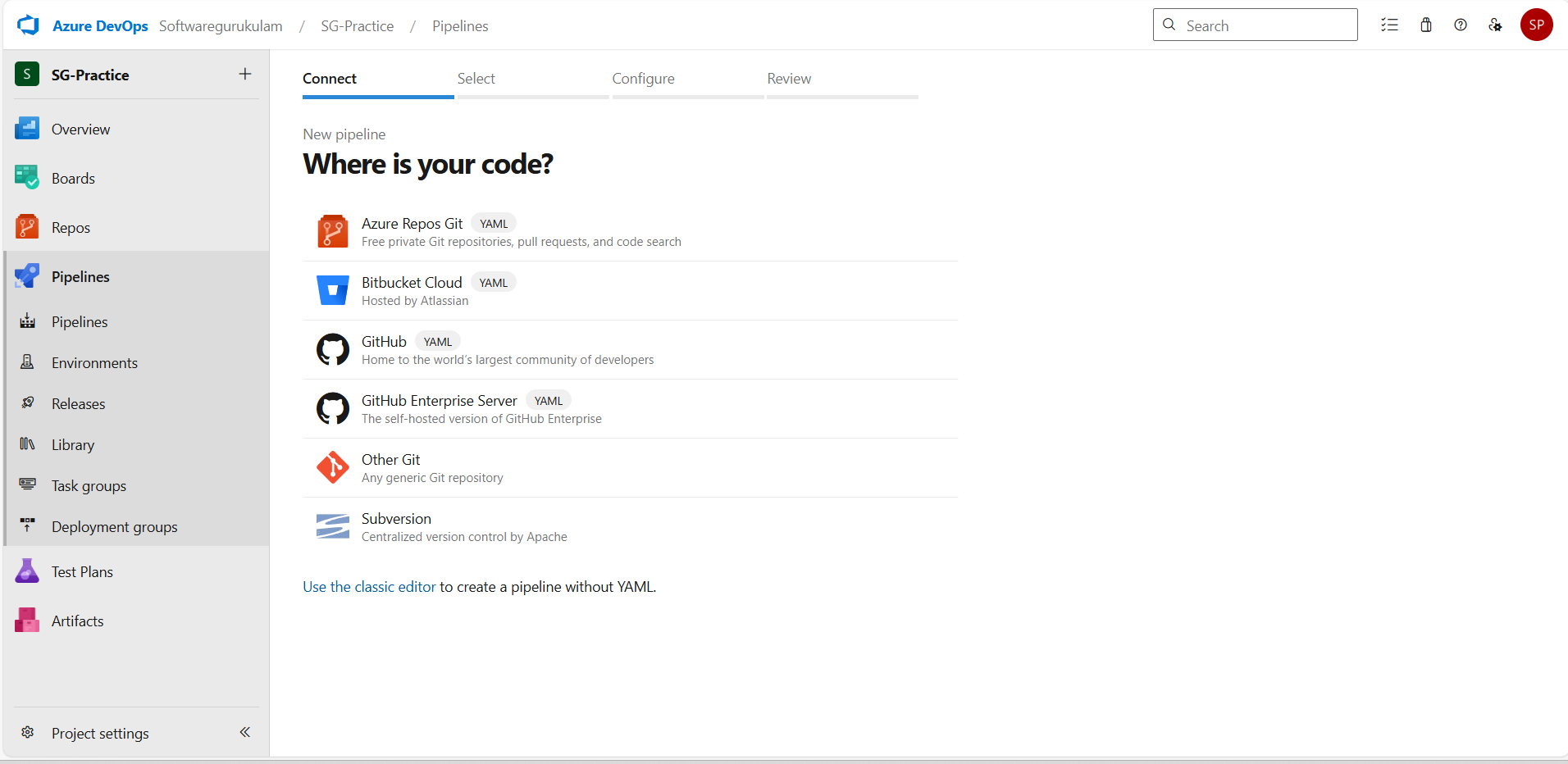
1. Select:
   * Registry type: Docker Hub
   * Docker ID: your Docker Hub username
   * Password: your Docker Hub password or access token
   * Registry URL: https://index.docker.io/v1/
2. Give a name like DockerHubConnection.
3. Click Verify and Save.



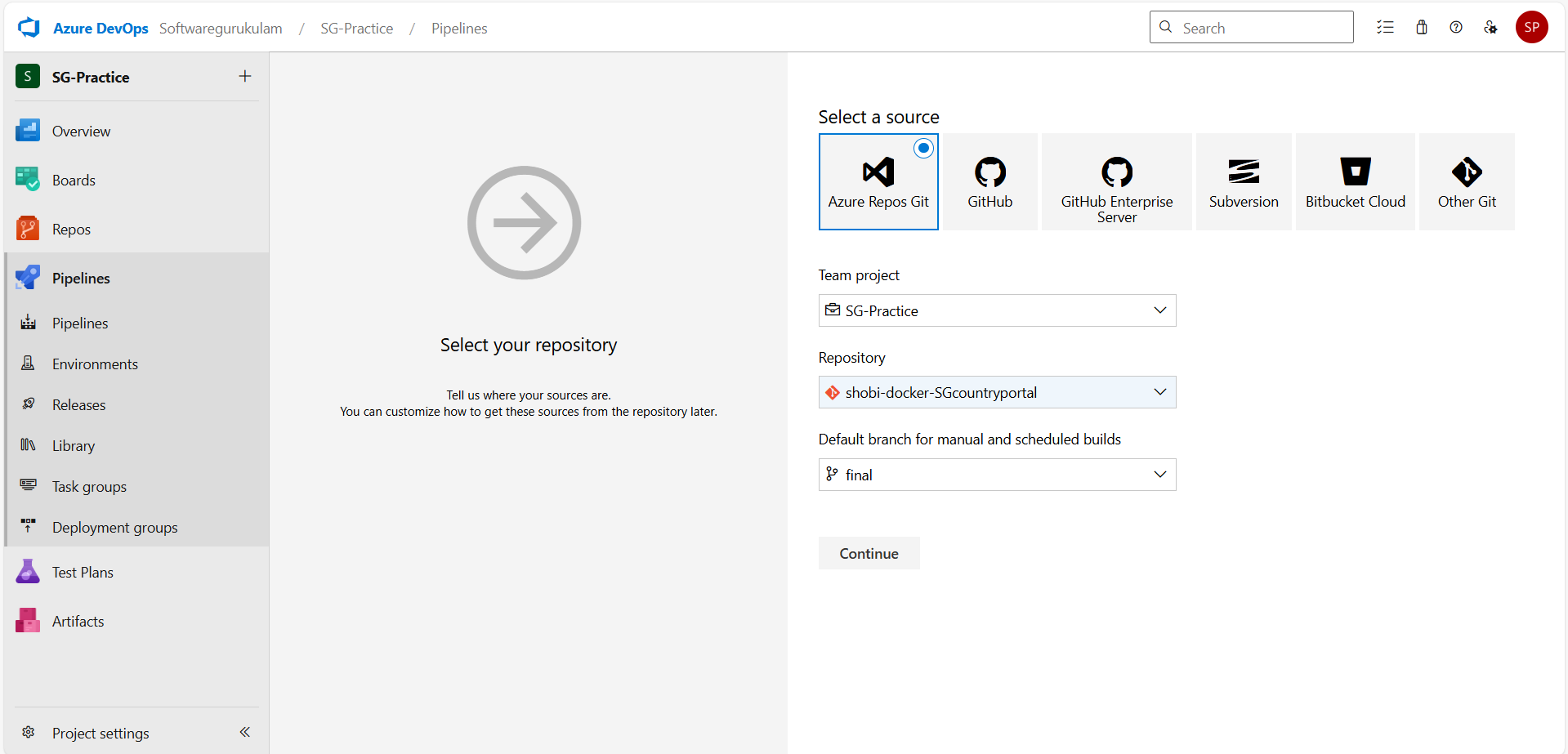
**🧱 Step 2: Create a Build Pipeline Using Classic Editor**

**1. Navigate to Pipelines > Create Pipeline**

* Select **"Use the classic editor"** (bottom of the screen).



* Choose your repository (Azure Repos Git or GitHub).
* Click **Continue**.



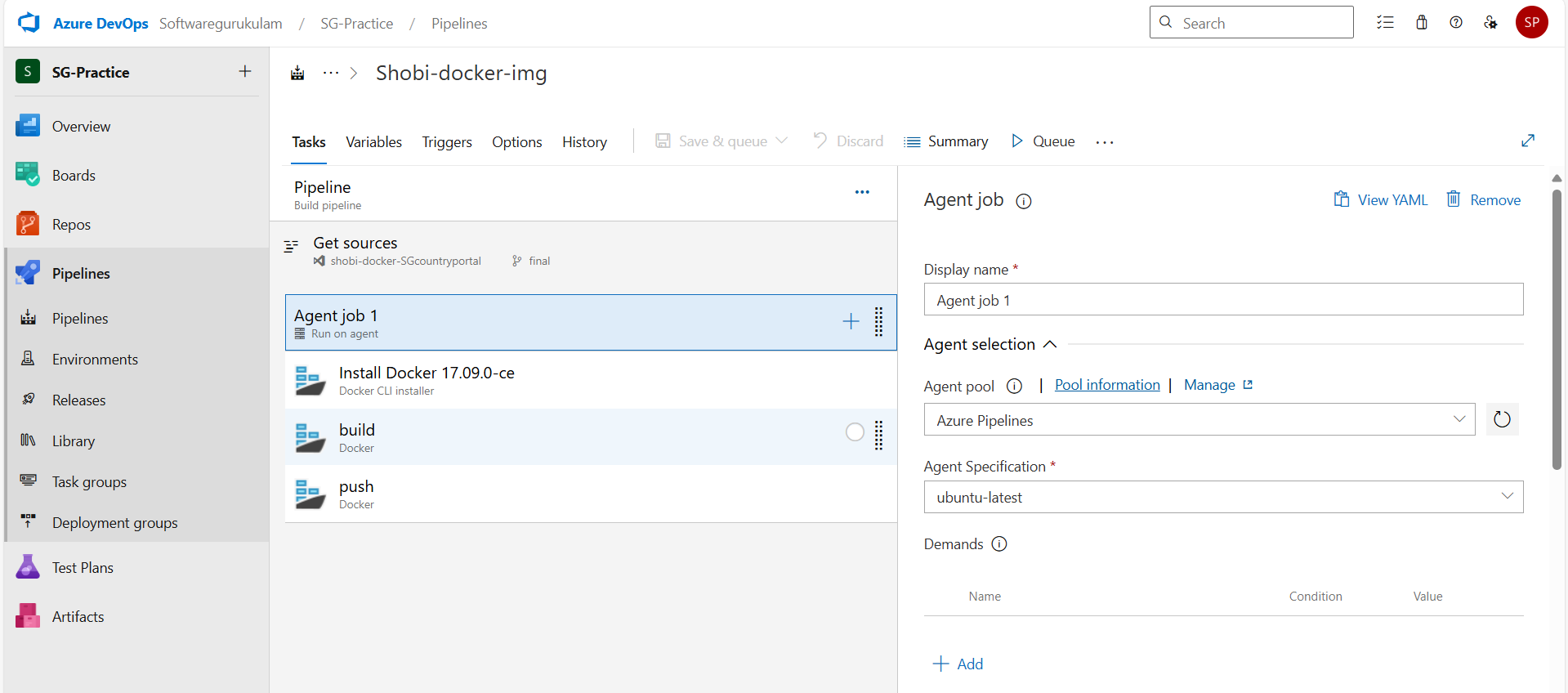
**2. Select a Template**

* Choose **"Empty Job"**.

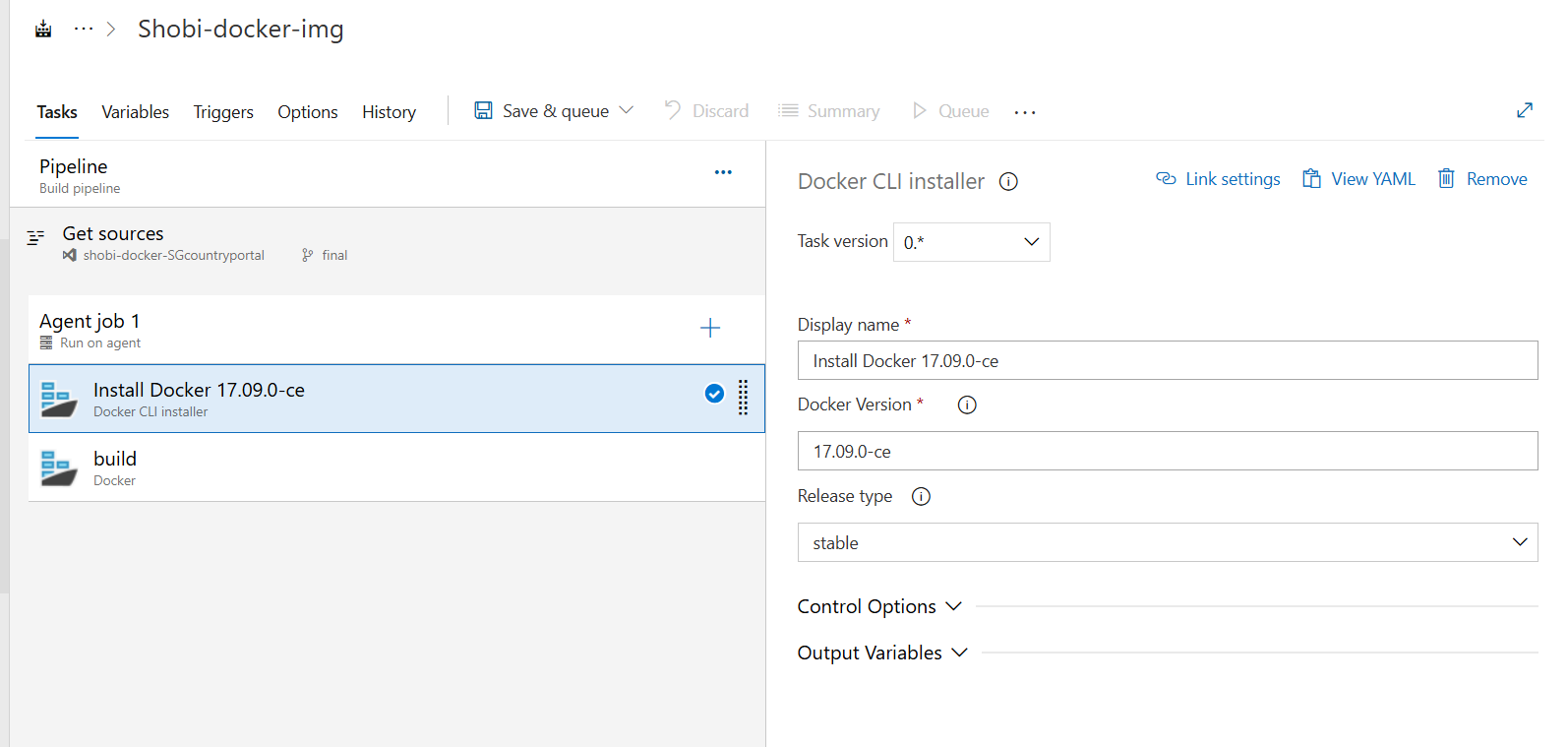
**🧩 Step 3: Configure the Build Pipeline**

**Agent job:**

* Set **Agent pool** to Azure Pipelines
* Set **Agent Specification** to ubuntu-latest

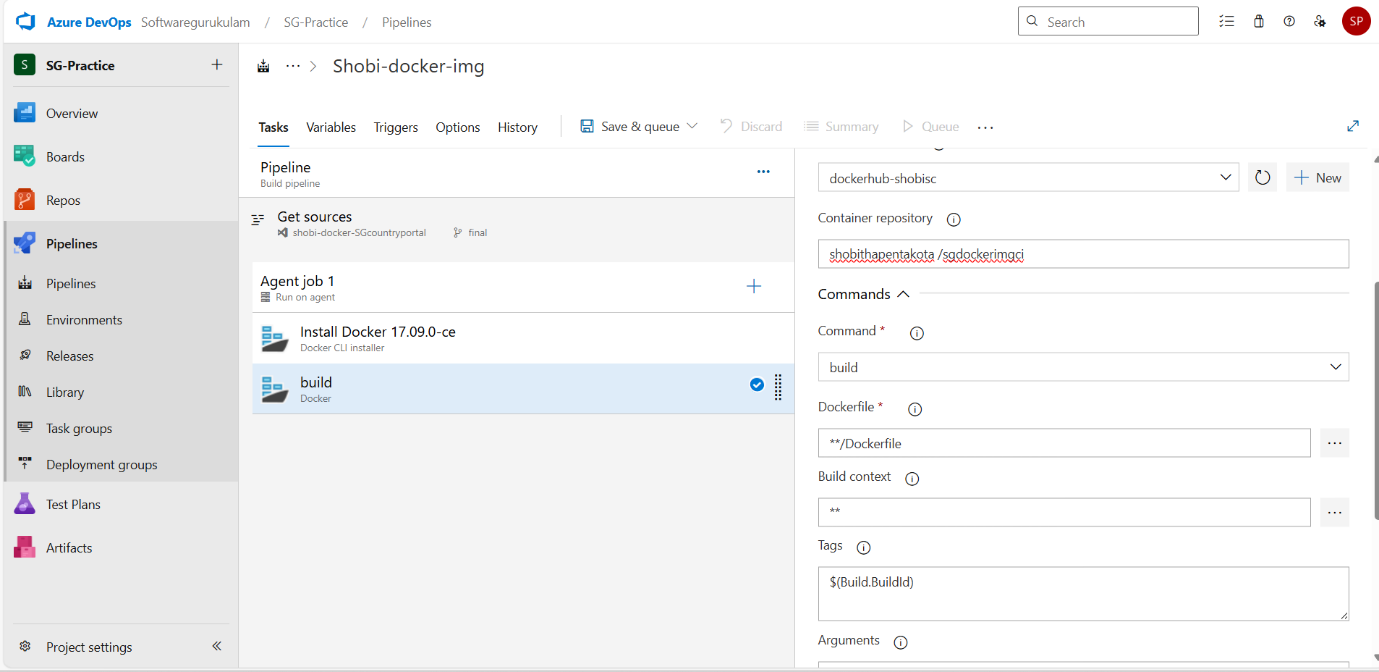


Task-1: docker cli installer.



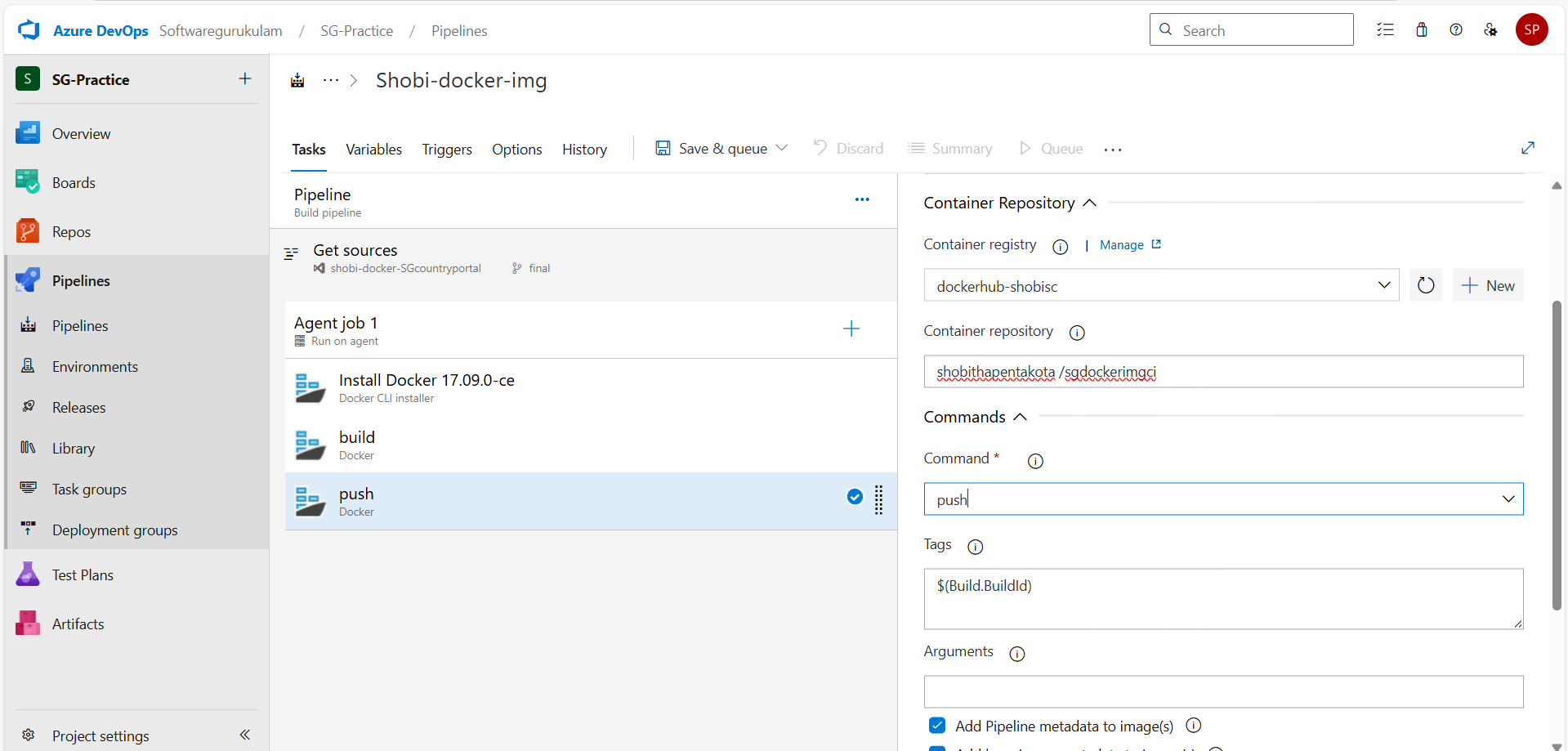
**Task 2: Docker - Build**

* Add another **Docker** task.
* **Display name**: Build Docker Image
* **Command**: build
* **Dockerfile**: Path to your Dockerfile (e.g., $(Build.SourcesDirectory)/Dockerfile)
* **Image name**: yourdockerhubusername/yourimagename:$(Build.BuildId)



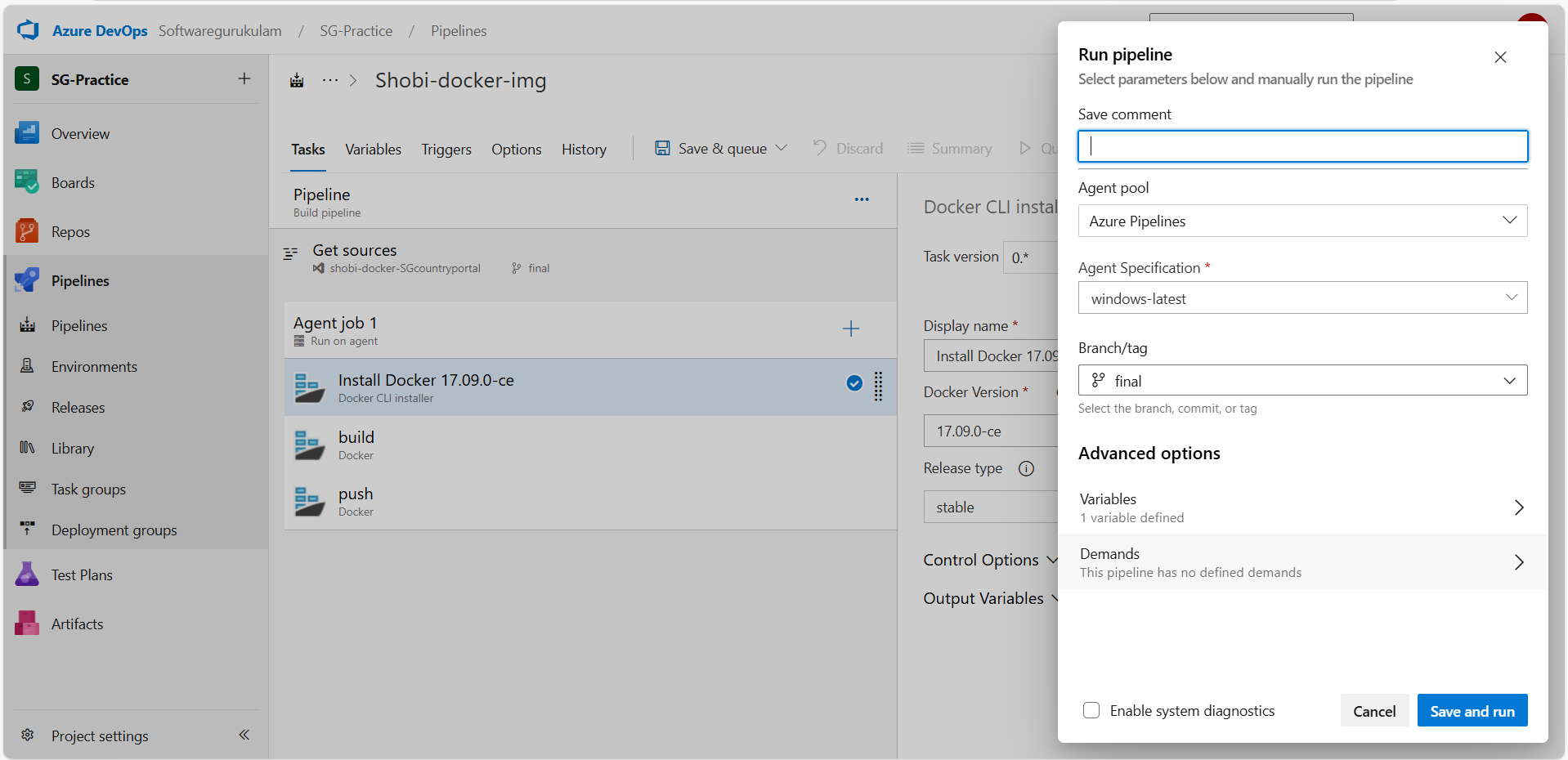
**Task 3: Docker - Push**

* Add another **Docker** task.
* **Display name**: Push Docker Image
* **Command**: push
* **Image name**: yourdockerhubusername/yourimagename:$(Build.BuildId)



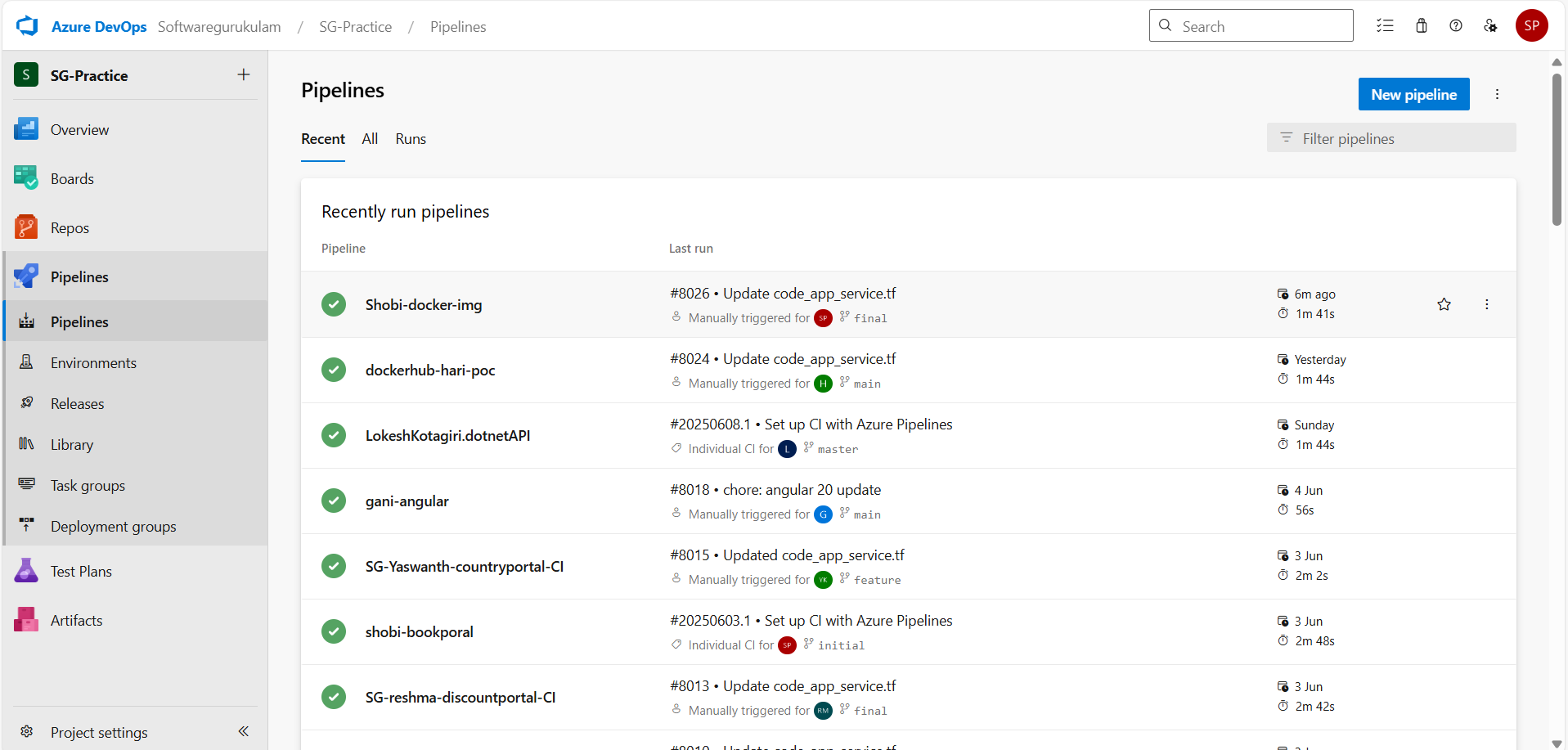
**📂 Step 4: Save and Run**

1. Click **Save & Queue > Save and run**.
2. Monitor the pipeline.
3. After success, your Docker image will be available in Docker Hub.

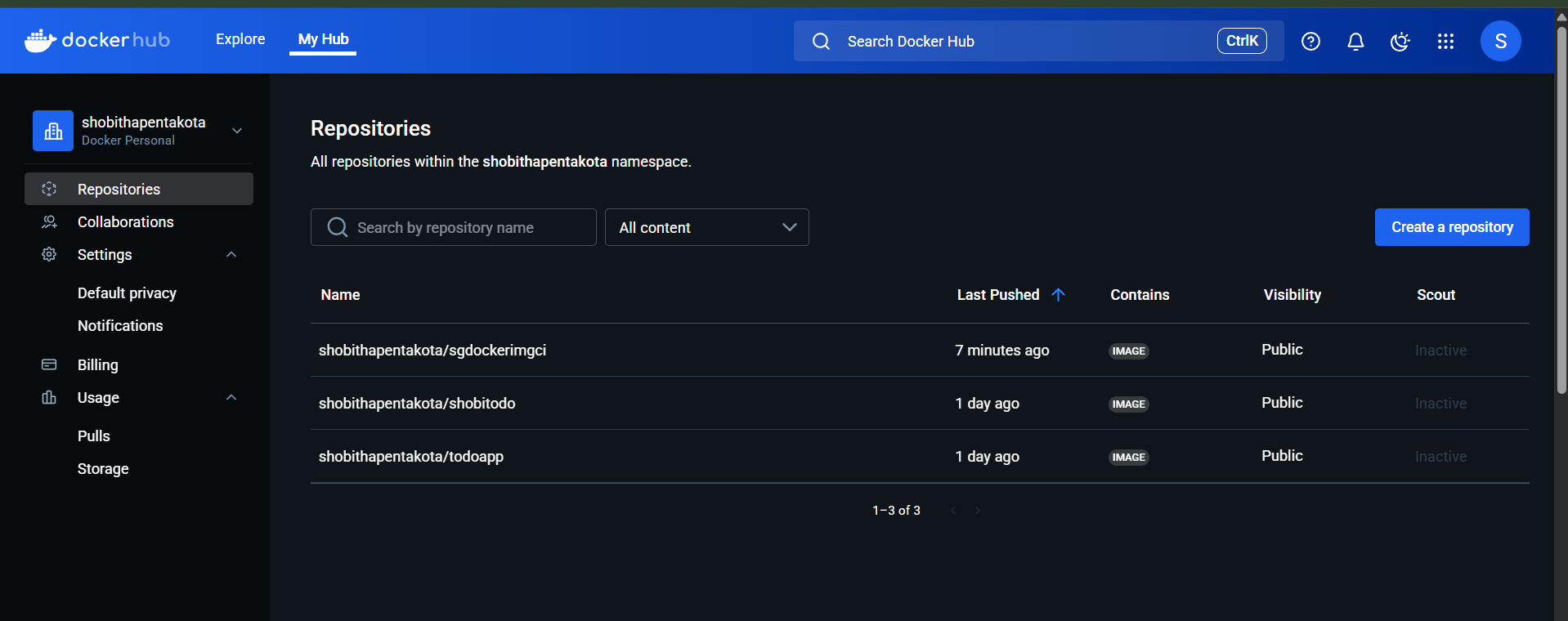


**✅ Result**

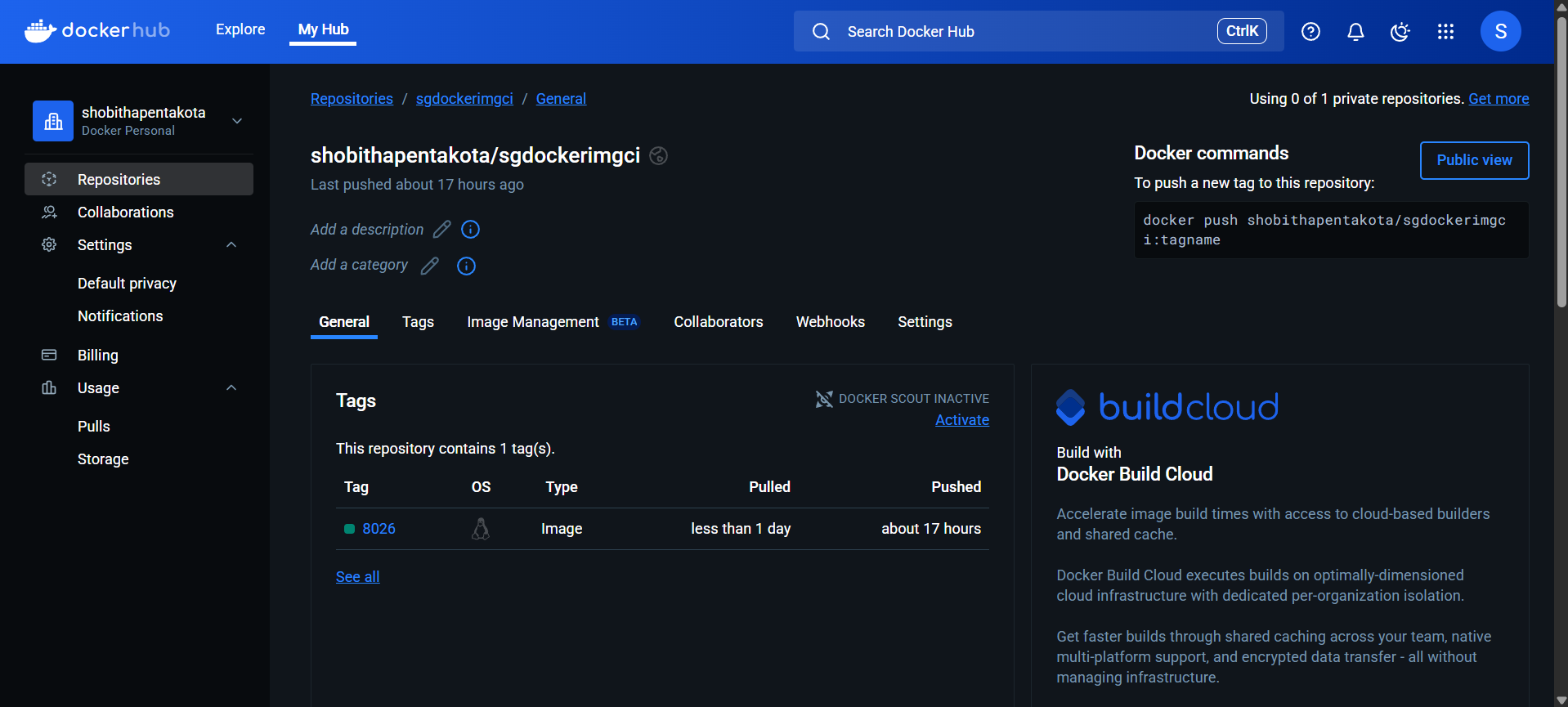
* Logs in to Docker Hub using a Service Connection.
* Builds the Docker image from your Dockerfile.
* Pushes the image to Docker Hub.



* Our pipeline run successfully. ……shobi-docker-img
* Builds the Docker image from your Dockerfile.



* This is our docker hub account and repo created and image successfully pushed to that repo….



* Sgdockerimgci…….Pushes the image to Docker Hub.