AZURE DEVOPS PART-2

1. **Setting up SonarQube** on an Ubuntu VM using Docker.
2. **Adding a Linux (Ubuntu) agent** to Azure DevOps.
3. **Installing necessary tools** (Maven, Java, Docker, Trivy) on the VM.
4. **Setup Azure Artifacts Feed**
5. **Creating a classic build and release pipeline** in Azure DevOps.

Part 1: Setting up SonarQube on Ubuntu VM

Step 1: Create Ubuntu VM in Azure

1. Go to the Azure Portal.
2. Navigate to **Virtual Machines** > **Create Virtual Machine**.
3. Choose **Ubuntu 20.04 LTS** as the OS.
4. Select the size based on your requirements (minimum 2 vCPUs, 4 GB RAM for SonarQube).
5. Complete the configuration and click **Create**.
6. After the VM is created, SSH into the VM using:

bash

Copy code

```
ssh <username>@<your-vm-ip>
```

Step 2: Install Docker on the Ubuntu VM

1. Update the system and install Docker:

```
sudo apt update
sudo apt install apt-transport-https ca-certificates curl software-properties-common
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu
$(lsb_release -cs) stable"
sudo apt update
sudo apt install docker-ce
sudo usermod -aG docker ubuntu
newgrp docker
```

2. Verify Docker installation:

```
docker --version
```

Step 3: Run SonarQube in Docker

1. Pull and run SonarQube in a Docker container:

```
docker run -d -p 9000:9000 sonarqube:lts-community
```

2. Access SonarQube:
   - Open a browser and navigate to http://<your-vm-ip>:9000.
   - Default credentials: **admin/admin**.

Part 2: Install Maven, Java, Docker, and Trivy on the Ubuntu VM

Step 1: Install Java

1. Install OpenJDK:

```
sudo apt update
sudo apt install openjdk-11-jdk -y
```

2. Verify Java installation:

```
java -version
```

Step 2: Install Maven

1. Install Maven:

```
sudo apt update
sudo apt install maven -y
```

2. Verify Maven installation:

```
mvn -version
```

Step 3: Install Trivy

1. Install Trivy:

```
sudo apt update
sudo apt install wget apt-transport-https gnupg lsb-release -y
wget -qO - https://aquasecurity.github.io/trivy-repo/deb/public.key | sudo apt-key add -
echo "deb https://aquasecurity.github.io/trivy-repo/deb $(lsb_release -sc) main" | sudo tee -a
/etc/apt/sources.list.d/trivy.list
sudo apt update
sudo apt install trivy -y
```

Part 3: Add the Ubuntu VM as an Agent in Azure DevOps

Step 1: Download and Configure the Agent

1. **Go to Azure DevOps**:
   - Navigate to **Organization Settings** > **Agent Pools**.
   - Click **New Agent** and select **Linux**.
   - Copy the download URL and follow the steps provided in the Azure DevOps UI.
2. **Install and Configure the Agent** on the VM:

```
mkdir myagent && cd myagent

wget https://vstsagentpackage.azureedge.net/agent/2.186.1/vsts-agent-linux-x64-2.186.1.tar.gz

tar zxvf vsts-agent-linux-x64-2.186.1.tar.gz

./config.sh
```

3. **Provide the following details**:
    - Azure DevOps organization URL.
    - Personal Access Token (PAT).
    - Name your agent.
4. **Start the agent**:

    ```
    ./run.sh
    ```

Step 2: Verify Agent Registration

- Go to **Organization Settings** > **Agent Pools** and verify that the Linux agent is listed as available.

Part-3: Setting up an Azure Artifacts Feed

Step 1: Navigate to Artifacts in Azure DevOps

1. **Go to Azure DevOps**: [Azure DevOps Portal](#).
2. **Navigate to Your Project**: Select your project from the home page.
3. In the left sidebar, click on **Artifacts**.
4. **Create New Feed**:
    - Click **+ New Feed** at the top right.
    - Give your feed a name (e.g., MyArtifactsFeed).
    - Set the visibility:
        - **Private**: Accessible only to users in your organization.
        - **Public**: Anyone with the link can access it.
    - Click **Create**.

Now your feed is ready to host artifacts like Maven, npm, NuGet, or Python packages.

Step 2: Connect to the Azure Artifacts Feed

Depending on the package type you're using (Maven, npm, NuGet, or Python), you will need to connect your build system to Azure Artifacts.

For Maven:

1. **Go to the feed you created** (e.g., MyArtifactsFeed).
2. Click on **Connect to Feed**.
3. Select **Maven** from the list of package managers.
4. You will see instructions to add the feed to your Maven settings:
    - Add the feed URL to your ~/.m2/settings.xml file.

Example settings.xml:

```
<mirrors>
 <mirror>
   <id>my-artifact-feed</id>
   <mirrorOf>*</mirrorOf>
   <url>https://pkgs.dev.azure.com/<your-organization>/_packaging/<your-feed-name>/maven/v1</url>
 </mirror>
```

```
</mirrors>

<servers>
 <server>
   <id>my-artifact-feed</id>
   <username>my-username</username>
   <password>your-personal-access-token</password>
 </server>
</servers>
```

- Replace the placeholders (<your-organization>, <your-feed-name>, your-personal-access-token) with the actual values.
- The **Personal Access Token (PAT)** can be generated in **Azure DevOps** under **User Settings** > **Personal Access Tokens**.

Step 3: Push Artifacts to the Feed

For Maven:

1. **Add Distribution Management** in your pom.xml to configure where Maven will deploy the built artifacts:

Example pom.xml snippet:

```
<distributionManagement>
 <repository>
   <id>my-artifact-feed</id>
   <url>https://pkgs.dev.azure.com/<your-organization>/_packaging/<your-feed-name>/maven/v1</url>
 </repository>
</distributionManagement>
```

2. **Deploy the Artifact**: Once your build is ready, use the following Maven command to deploy the artifact:

```
mvn deploy
```

This will upload the JAR/WAR to your Azure Artifacts feed.

Part 5: Create Classic Build Pipeline

Step 1: Navigate to Pipelines

1. Go to **Pipelines** > **Builds**.
2. Click **New Pipeline** > **Use the classic editor**.
3. Select your **Azure Repo** or **GitHub Repo** as the source.

Step 2: Configure Build Pipeline Stages

Stage 1: Maven Authenticate

1. Add a **Maven task** in the pipeline.
2. In the settings, configure Maven to use your **settings.xml** for repository authentication.

Stage 2: Maven Compile

1. Add another **Maven task**.
2. Set the **Goal** to compile to compile the code.

Stage 3: Maven Package

1. Add another **Maven task**.
2. Set the **Goal** to package to create the JAR/WAR files.

Stage 4: Copy Files to Build Artifact Staging Directory

1. Add a **Copy Files** task.
2. Set the source folder to the location where the JAR/WAR is stored (e.g., $(Build.SourcesDirectory)/target).
3. Set the destination folder to $(Build.ArtifactStagingDirectory).

Stage 5: Publish Build Artifact

1. Add a **Publish Build Artifacts** task.
2. Specify the **path** as $(Build.ArtifactStagingDirectory)

Stage 6: Trivy File System Scanning

1. Add a **Command Line or Bash** task.
2. Use the following command to scan the file system:

```
trivy fs  --format  table -o report.html .
```

Stage 7: SonarQube Analysis

1. Add a **Prepare Analysis Configuration** task for SonarQube.
2. Configure the connection to your SonarQube server.
3. Add a **Run Code Analysis** task.

Stage 8: Deploy to Feed

1. Add a **Maven Deploy** task to deploy the artifact to your **Azure Artifacts feed** or an external Maven repository.

Stage 9: Docker Build and Push

1. Add a **Docker task** to build the Docker image:
   - **Command**: Build.
   - **Dockerfile**: Provide the path to the Dockerfile.
   - **Image name**: Set the name of the image.
2. Add another **Docker task** to push the image:
   - **Command**: Push.
   - **Container registry**: Select your Azure Container Registry or Docker Hub.

Part 5: Create Classic Release Pipeline

Step 1: Navigate to Releases

1. Go to **Pipelines** > **Releases**.
2. Click **New Pipeline**.
3. Select the build artifact from your classic build pipeline.

Step 2: Define Stages in the Release Pipeline

Stage 1: Kubectl Installer Task

1. Add a **Kubectl Installer** task to install kubectl on the agent.
2. This ensures kubectl is available to interact with your Kubernetes cluster.

Stage 2: Kubectl Apply Task

1. Add a **Kubectl Apply** task to deploy your Kubernetes manifests (e.g., deployment.yaml, service.yaml).
2. In the **Arguments** field, provide the path to the Kubernetes manifests:

```
kubectl apply -f $(System.DefaultWorkingDirectory)/manifests/deployment.yaml
```