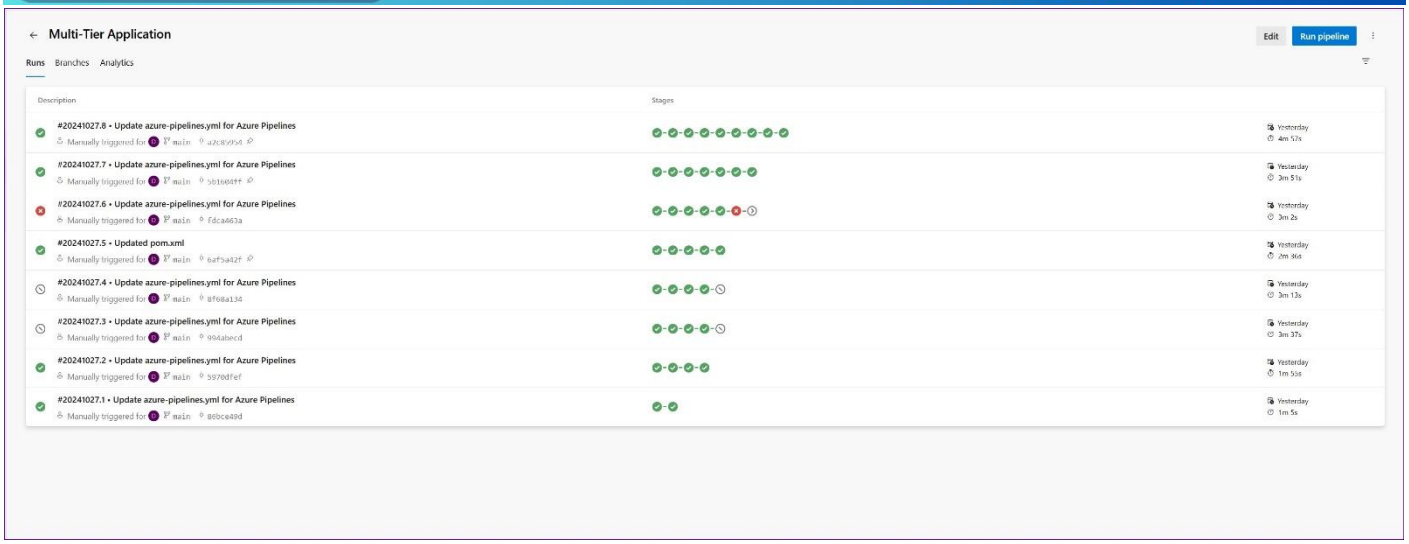
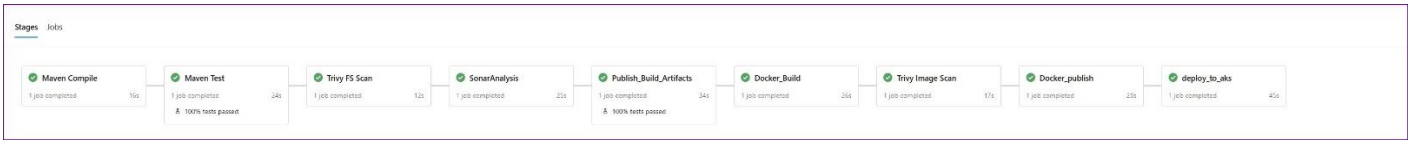
# DevOps Shack

## Detailed Documentation on Multi-Stage Pipeline YAML Configuration

**Introduction**

This document explains a comprehensive multi-stage YAML pipeline configuration for Azure DevOps. The pipeline automates a complete CI/CD workflow, handling tasks like code compilation, security scans, code quality analysis, Docker image creation, and application deployment to a Kubernetes cluster. Each stage executes in sequence on a specified agent pool, ensuring consistency and security across the deployment process.

---

**1. Pipeline Configuration**

**Pipeline Trigger Configuration**

```
trigger:
 - none
```

- **Explanation**: The pipeline does not trigger automatically on branch or path changes. It's configured for manual or pipeline-based triggers, which is ideal for pipelines integrated into broader CI/CD workflows.
- **Use Cases**: Manual or scheduled triggers are useful in scenarios where the pipeline is part of a complex process or should run only after specific conditions are met.

**Agent Pool Configuration**

```
pool:
  name: Aditya
  demands: agent.name -equals agent-1
```

- **Explanation**: All jobs run on a specific agent pool (Aditya) and agent (agent-1). This configuration is crucial for ensuring a controlled environment, especially when tasks have unique dependencies.
- **Use Cases**: Specific agent selection is helpful when tasks need distinct tools or configurations that may not be available across all agents.

---

**2. Stages and Jobs**

The pipeline comprises seven distinct stages, each addressing a unique part of the CI/CD process.

**Stage 1: Compile Stage**

```
- stage: Compile
  displayName: 'Compile Stage'
  jobs:
    - job: CompileJob
      displayName: 'Compile Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - script: mvn compile
          displayName: 'Compile-Step'
```

- **Purpose**: Compiles the source code to ensure it's error-free.
- **Key Points**: Running on a dedicated agent ensures a consistent compilation environment.

**Stage 2: Trivy File System Scan Stage**

```
- stage: Trivy_FS_Scan
  displayName: 'Trivy FS Stage'
  jobs:
    - job: TrivyFSJob
      displayName: 'TrivyFS Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - script: trivy fs --format table -o trivy-fs-report.html .
          displayName: 'TrivyFS-Scan-Step'
```

- **Purpose**: Scans the file system for vulnerabilities using Trivy.
- **Key Points**: Security scans identify any known vulnerabilities early, outputting results to trivy-fs-report.html.

**Stage 3: SonarQube Scan Stage**

```
- stage: SonarQube_Scan
  displayName: 'SonarQube Stage'
  jobs:
    - job: SonarQubeJob
      displayName: 'SonarQube Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - task: SonarQubePrepare@5
          inputs:
            SonarQube: 'sonar-svc'
            scannerMode: 'CLI'
            configMode: 'manual'
            cliProjectKey: 'screte-santa'
            cliProjectName: 'screte-santa'
            cliSources: '.'
            extraProperties: |
              sonar.java.binaries=.
        - task: SonarQubeAnalyze@5
          inputs:
            jdkversion: 'JAVA_HOME'
```

- **Purpose**: Conducts code quality and security checks with SonarQube.
- **Key Points**: Results help in maintaining code quality and adhering to best practices.

**Stage 4: Build Stage**

```
- stage: Build
  displayName: 'Build Stage'
  jobs:
    - job: BuildJob
      displayName: 'Build Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - script: mvn package
          displayName: 'Build-Package-Step'
```

- **Purpose**: Builds and packages the application.
- **Key Points**: Ensures the application is ready for deployment by creating a deployable artifact, like a JAR or WAR file for Java projects.

**Stage 5: Docker Stage**

```
- stage: Docker
  displayName: 'Docker Stage'
  jobs:
    - job: DockerJob
      displayName: 'Docker Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - script: mvn package
          displayName: 'Build-Package-Step'
        - task: Docker@2
          inputs:
            containerRegistry: 'docker-svc'
            repository: 'adijaiswal/santa'
            command: 'buildAndPush'
            Dockerfile: '**/Dockerfile'
            tags: 'latest'
```

- **Purpose**: Builds a Docker image and pushes it to a Docker registry.
- **Key Points**: The Docker image is tagged as latest for easy identification as the most recent build.

**Stage 6: Trivy Image Scan Stage**

```
- stage: Trivy_Image_Scan
  displayName: 'Trivy Image Stage'
  jobs:
    - job: TrivyImageJob
      displayName: 'Trivy Image Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
```

```
      - script: trivy image --format table -o trivy-fs-report.html adijaiswal/santa:latest
        displayName: 'Trivy-Image-Scan-Step'
```

- **Purpose**: Runs a Trivy security scan on the Docker image.
- **Key Points**: This scan helps to identify any vulnerabilities in the image before it's deployed to production.

**Stage 7: Kubernetes Deployment Stage**

```
- stage: K8_Deploy
  displayName: 'K8_Deploy Stage'
  jobs:
    - job: K8_DeployJob
      displayName: 'K8_Deploy Job'
      pool:
        name: Aditya
        demands: agent.name -equals agent-1
      steps:
        - task: KubectlInstaller@0
          inputs:
            kubectlVersion: 'latest'
        - task: Kubernetes@1
          inputs:
            connectionType: 'Kubernetes Service Connection'
            kubernetesServiceEndpoint: 'k8-service-connection'
            namespace: 'default'
            command: 'apply'
            useConfigurationFile: true
            configuration: 'k8-dep-svc.yml'
            secretType: 'dockerRegistry'
            containerRegistryType: 'Container Registry'
            dockerRegistryEndpoint: 'docker-svc'
            forceUpdate: false
```

- **Purpose**: Deploys the Docker image to a Kubernetes cluster.
- **Key Points**: Uses a Docker registry secret to ensure secure image pulling from the specified registry.

---

**3. Best Practices**

1. **Agent Pool Consistency**: Maintain the same agent pool (Aditya) and agent (agent-1) across all stages for environmental consistency.
2. **Security Scans**: Regularly update Trivy and SonarQube to detect the latest vulnerabilities.
3. **Pipeline Triggers**: Consider adding branch-based triggers to automate pipeline execution for new code merges.
4. **Kubernetes Deployment**: Use a dedicated namespace for each environment (e.g., development, staging, production) to avoid conflicts. Implement rollback strategies, like Kubernetes' rolling updates.

5. **Artifact Management**: Store artifacts and Docker images in a secure, versioned repository for traceability.

---

**4. Conclusion**

This multi-stage YAML configuration streamlines the CI/CD process from code compilation to Kubernetes deployment. By integrating Trivy and SonarQube, the pipeline maintains high security and quality standards. Docker and Kubernetes enhance scalability, and the consistent use of a controlled agent pool ensures reliable execution. This setup is adaptable, allowing for additional testing stages or integration with tools like Helm or Terraform based on project needs.