# Azure DevOps Environments and Approvals & Checks

**1. Introduction to Azure DevOps Environments**

Azure DevOps Environments provide a way to manage deployment pipelines for different stages like development, testing, staging, and production. These environments allow teams to implement policies, approvals, and gates before deploying software to a specific environment.

Azure DevOps Environments represent stages in the deployment pipeline. Each environment typically corresponds to a specific phase in the application lifecycle, such as:

* **Development (dev)**: Initial testing and integration.
* **Quality Assurance (QA)**: Comprehensive functional and performance testing.
* **Staging (stage)**: Pre-production validation with real-world data.
* **Production (prod)**: Live environment serving end-users.

**1.1 Benefits of Using Environments**

* Organizes deployments across different stages.
* Enhances security and compliance.
* Enables approval workflows for controlled releases.
* Provides traceability and audit logs.
* Allows deployment targeting using Kubernetes, virtual machines, or service fabric.

**2. Creating and Managing Environments**

**2.1 Steps to Create an Environment in Azure DevOps**

1. Navigate to your Azure DevOps project.
2. Select **Pipelines** > **Environments**.
3. Click **New Environment**.
4. Provide a name and select the appropriate resource type (e.g., Kubernetes, Virtual Machines, etc.).
5. Configure security roles for environment access.
6. Save and use it within deployment pipelines.

**2.2 Environment Security and Access Control**

* **Roles and Permissions:** Assign access roles such as Environment Admins, Contributors, or Viewers.
* **Branch Policies:** Restrict deployments to approved branches.
* **Service Connections:** Securely connect to cloud services for deployment.

**3. Approvals and Checks in Environments**

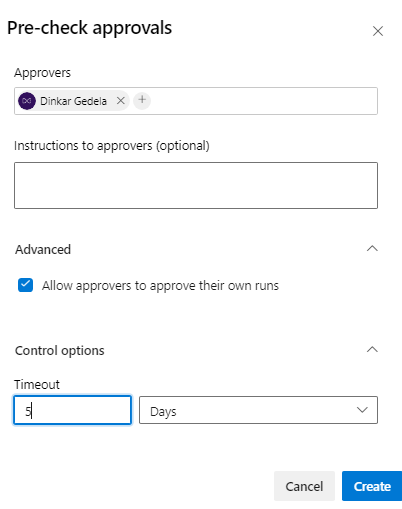
Approvals and checks are used to enforce policies before deployments can proceed. These can include manual approvals, business logic validation, security scanning, and compliance checks.

**3.1 Types of Approvals and Checks**

1. **Approvals:** Require a human reviewer to approve or reject a deployment.
2. **Branch Control:** Ensures only specific branches can deploy.
3. **Business Hours Enforcement:** Limits deployments to specific times.
4. **Service Now Change Management:** Integrates with ServiceNow for change control.
5. **Security and Compliance Scans:** Runs automated vulnerability scans before allowing deployment.
6. **Required Templates Check:** Ensures deployment follows predefined templates.

**3.2 Configuring Approvals and Checks**

1. Go to **Pipelines > Environments**.
2. Select the desired environment.
3. Click **Approvals and checks**.
4. Choose a check type (Manual Approval, Branch Control, Business Hours, etc.).
5. Configure settings like required approvers, delay time, and conditions.
6. Save and apply to deployment pipelines.



**4. Best Practices for Approvals and Checks**

* Use **Manual Approvals** for critical production deployments.
* Implement **Business Hours Enforcement** to control deployment timing.
* Automate **security and compliance checks** using tools like SonarCloud or Azure Security Center.
* Leverage **branch controls** to ensure only stable code is deployed.
* Monitor and review **audit logs** for deployment traceability.

**5. Troubleshooting and Common Issues**

**5.1 Approval Request Not Triggering**

* Verify if the approval is enabled in the environment settings.
* Check user permissions for approval roles.

**5.2 Deployment Stuck on Checks**

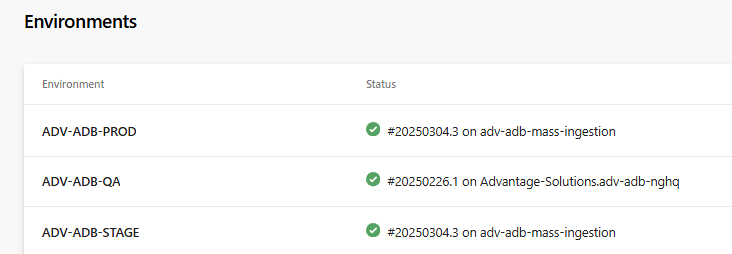
* Ensure external services (e.g., ServiceNow) are accessible.
* Check for misconfigured conditions in approvals.

**5.3 Permission Denied Errors**

* Review role assignments in environment security settings.
* Confirm service connections have required permissions.

**6. Example**

Below are the Azure DevOps environments available



Main.yml

trigger:

  tags:

    include:

      - v\*.\*.\*

pr: none

stages:

#QA

- stage: QA

  displayName: 'Deploy to QA'

  condition: succeeded()

  variables:

    - group: ADV-ADB-NGHQ-DEPLOY-QA

  jobs:

    - deployment: Deploy\_QA

      displayName: 'Deploy Databricks Assets to QA'

      environment: ADV-ADB-QA

      pool: ADV-TF-QA-EASTUS2-Data and Analytics Portfolio

      strategy:

        runOnce:

          deploy:

            steps:

            - template: /.ado/deploy-template.yml

              parameters:

                env: qa

#STAGE

- stage: STAGE

  displayName: 'Deploy to STAGE'

  dependsOn: QA

  condition: succeeded()

  variables:

    - group: ADV-ADB-NGHQ-DEPLOY-STAGE

  jobs:

    - deployment: Deploy\_STAGE

      displayName: 'Deploy Databricks Assets to STAGE'

      environment: ADV-ADB-STAGE

      pool: ADV-TF-STAGE-EASTUS2-Data and Analytics Portfolio

      strategy:

        runOnce:

          deploy:

            steps:

            - template: /.ado/deploy-template.yml

              parameters:

                env: stage

#PROD

- stage: PRODUCTION

  displayName: 'Deploy to PROD'

  dependsOn: STAGE

  condition: succeeded()

  variables:

    - group: ADV-ADB-NGHQ-DEPLOY-PROD

  jobs:

    - deployment: Deploy\_PROD

      displayName: 'Deploy Databricks Assets to STAGE'

      environment: ADV-ADB-PROD

      pool: ADV-TF-PROD-EASTUS2-Data and Analytics Portfolio

      strategy:

        runOnce:

          deploy:

            steps:

            - template: /.ado/deploy-template.yml

              parameters:

                env: prod

Below fields you can see the approvals and checks

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer screen

AI-generated content may be incorrect.

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**7. Conclusion**

Azure DevOps Environments and Approvals & Checks provide a robust mechanism for controlled, secure, and compliant software deployments. By leveraging these features, teams can ensure smoother releases with proper governance and security compliance.

### Release naming convention

The release version naming convention can be in the below format so that we can have proper version in the GitHub release. This is the most common versioning convention.

**vMajor.Minor.Patch-buildPath**

For example, **v2.3.3-boost**  
  
The major version number is incremented when there are major changes to the software, such as new features or breaking changes.  
The minor version number is incremented when there are minor changes to the software, such as bug fixes or performance improvements.  
The patch version number is incremented when there are small changes to the software, such as documentation updates or security fixes.

The build path is for setting up the path, in order to deploy which DAB file.