**Introduction**

Azure DevOps provides a suite of capabilities for building, testing, and deploying applications through Continuous Integration and Continuous Deployment (CI/CD) pipelines. This document outlines these capabilities and provides YAML examples for implementing them in Azure Pipelines.

**1. Azure DevOps Capabilities**

Azure DevOps includes several services that support software development lifecycle:

**1.1 Azure Repos**

A version control system supporting Git repositories to manage source code and track changes.

**1.2 Azure Pipelines**

A CI/CD service for building, testing, and deploying applications.

**1.3 Azure Artifacts**

A package management system that hosts and shares Maven, npm, and NuGet packages.

**1.4 Azure Test Plans**

A toolset for manual and exploratory testing, as well as automated testing.

**1.5 Azure Boards**

An Agile planning tool for tracking work items, issues, and project progress.

Three ways to write pipeline-

* **YAML Pipelines** → Best for modern DevOps, fully automated CI/CD, and version control.
* **Classic Build** → Suitable for teams needing only CI and prefer UI-based setup.

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **YAML Pipelines (CI/CD)** | **Classic Build (CI)** | **Classic Release (CD)** |
| **Definition** | Code-based CI/CD pipeline | UI-based build pipeline | UI-based release pipeline |
| **Storage** | In repository as .yml file | Azure DevOps (not in repo) | Azure DevOps (not in repo) |
| **CI/CD Support** | Both CI & CD | Only CI (build) | Only CD (deploy) |
| **Version Control** | Yes, in Git | No (not stored as code) | No (not stored as code) |
| **Flexibility** | High (customizable) | Medium (task-based) | Medium (staged deployments) |
| **Ease of Use** | Harder to learn initially | Easier (UI-based) | Easier (UI-based) |
| **Best for** | Modern DevOps, GitOps | Legacy projects, UI-friendly CI | Enterprises needing manual approvals |

* **Classic Release** → Good for enterprise deployments requiring manual approvals

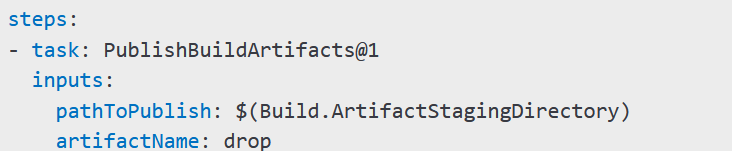
**1) Agents:** Agents execute pipeline jobs.



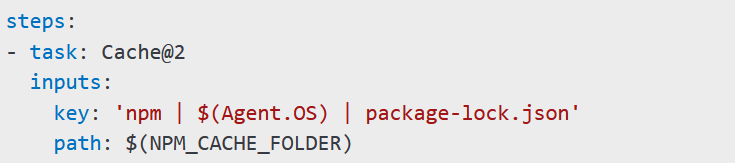
**2) Approvals:** Approvals enforce manual validation before proceeding.



**3) Artifacts:** Artifacts store pipeline outputs for later use.

****

**4) Caching:** Caching stores dependencies to speed up builds.

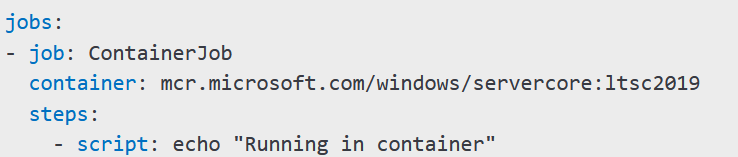
****

**Note:** Suppose, which we need to deploy by 2 pipelines in selfhosted agent, so caching can help here.

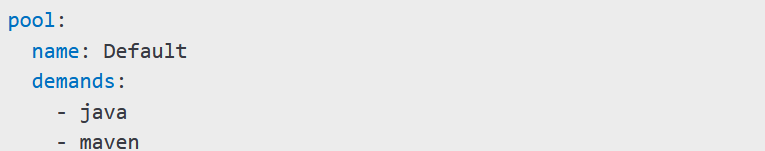
**5) Conditions** Condition control job execution.

****

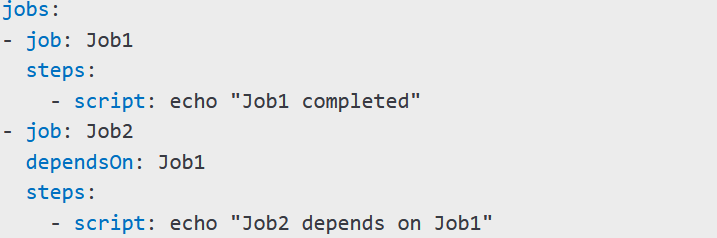
**6) Container Jobs** Run jobs inside containers.



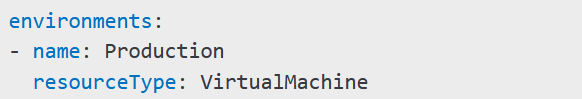
**7) Demands:** Ensure agents meet specific requirements.



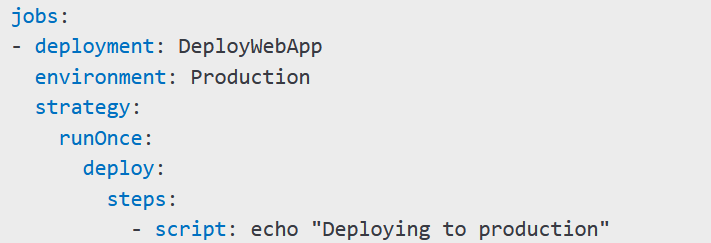
**8) Dependencies:** Define dependencies between jobs.

****

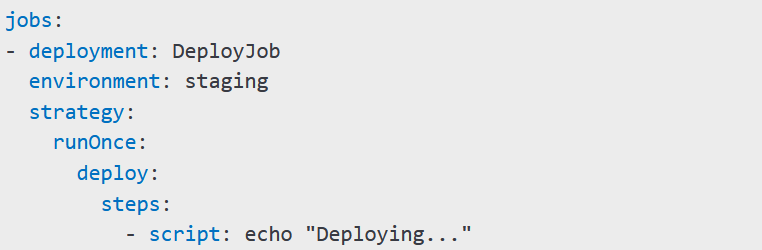
**9) Deployment Groups:** Group machines for deployment.



**10) Deployment Group Jobs:** Deploy to a group of machines.

****

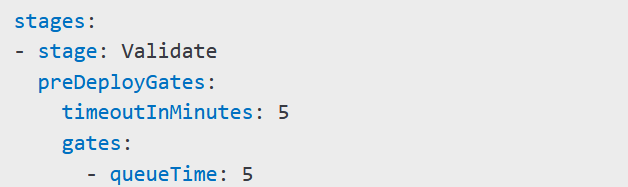
**11) Deployment Jobs:** Define deployment-specific jobs.

****

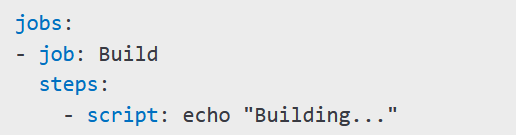
**12) Environment:** Define execution contexts.

****

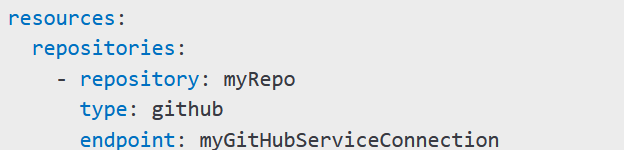
**13) Gates:** Gates enforce quality checks before proceeding.



**14) Job**: A job is a collection of steps.

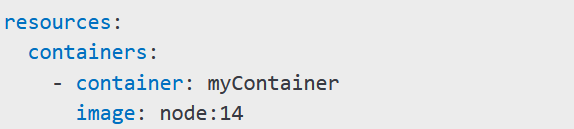


**15) Service Connections:** Connect to external services.

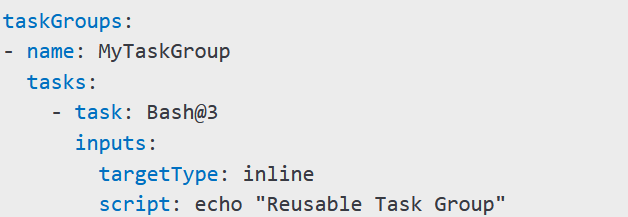


**16) Service Containers:** Use containers in pipelines.

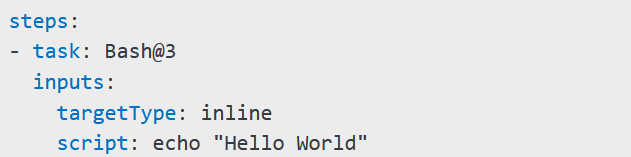
**17) Stages:** Stages group jobs together.



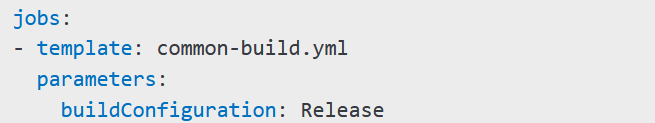
**18) Task Groups:** Reusable sets of tasks.



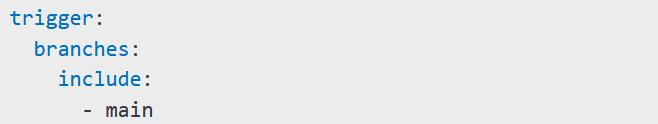
**19) Tasks:** Atomic units of work in a pipeline.

****

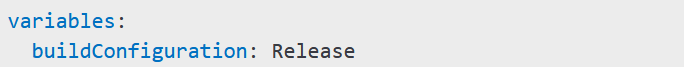
**20) Templates:** Reusable pipeline configurations.

****

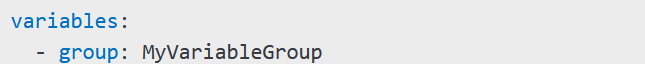
**21) Triggers:** Automate pipeline execution.

****

**22) Variables:** Store values for reuse.

****

**23) Variable Groups:** Store and manage variables centrally.

****