

Pipeline Concepts	
Stages,Jobs,Steps (create a pipeline with stages, jobs and steps)	<ul style="list-style-type: none">• A pipeline is made up of one or more stages. A pipeline can deploy to one or more environments.• A stage is a way of organizing jobs in a pipeline and each stage can have one or more jobs.• Each job runs on one agent. A job can also be agentless.• Each agent runs a job that contains one or more steps.• A step can be a task or script and is the smallest building block of a pipeline.
Artifact	<ul style="list-style-type: none">• An artifact is a collection of files or packages published by a run.
Trigger Syntax	<pre># specific path build trigger: branches: include: - master - releases/* paths: include: - docs exclude: - docs/README.md</pre>
Task types & usage	<p>A task is the building block for defining automation in a pipeline. A task is simply a packaged script or procedure that has been abstracted with a set of inputs.</p> <pre>- task: string # reference to a task and version, e.g. "VSBuild@1" condition: expression # see below continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false' enabled: boolean # whether or not to run this step; defaults to 'true' timeoutInMinutes: number # how long to wait before timing out the task target: string # 'host' or the name of a container resource to target</pre>

Jobs	<p>The full syntax to specify a job is:</p> <pre>- job: string # name of the job, A-Z, a-z, 0-9, and underscore displayName: string # friendly name to display in the UI dependsOn: string [string] condition: string strategy: parallel: # parallel strategy matrix: # matrix strategy maxParallel: number # maximum number simultaneous matrix legs to run # note: `parallel` and `matrix` are mutually exclusive # you may specify one or the other; including both is an error # `maxParallel` is only valid with `matrix` continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false' pool: pool # agent pool workspace: clean: outputs resources all # what to clean up before the job runs container: containerReference # container to run this job inside timeoutInMinutes: number # how long to run the job before automatically cancelling cancelTimeoutInMinutes: number # how much time to give 'run always even if cancelled tasks' before killing them variables: { string: string } [variable variableReference] steps: [script bash pwsh powershell checkout task templateReference] services: { string: string container } # container resources to run as a service container uses: # Any resources (repos or pools) required by this job that are not already referenced repositories: [string] # Repository references to Azure Git repositories pools: [string] # Pool names, typically when using a matrix strategy for the job</pre>
Workspace	<p>When you run an agent pool job, it creates a workspace on the agent. The workspace is a directory in which it downloads the source, runs steps, and produces outputs. The workspace directory can be referenced in your job using the Agent.BuildDirectory variable. Under this, various subdirectories are created:</p> <p>Build.SourcesDirectory is where tasks download the application's source code. Build.ArtifactStagingDirectory is where tasks download artifacts needed for the pipeline or upload artifacts before they are published. Build.BinariesDirectory is where tasks write their outputs. Common.TestResultsDirectory is where tasks upload their test results.</p>
Deployment jobs	Add syntax screenshot

	<p>Deployment jobs provide the following benefits:</p> <p>Deployment history: You get the deployment history across pipelines, down to a specific resource and status of the deployments for auditing.</p> <p>Apply deployment strategy: You define how your application is rolled out. Once Rolling deploy Canary</p> <p>Descriptions of lifecycle hooks</p> <p>preDeploy : / deploy : routeTraffic : Used to run steps that serve the traffic to the updated version. postRouteTraffic : Used to run the steps after the traffic is routed. Typically, these tasks monitor the health of the updated version for defined intervals. on: failure or on: success : Used to run steps for rollback actions or clean-up.</p>																				
Library of assets	<p>A library is a collection of build and release assets for an Azure DevOps project. Assets defined in a library can be used in multiple build and release pipelines of the project. The Library tab can be accessed directly in Azure Pipelines.</p> <p>The library contains two types of assets: variable groups and secure files.</p>																				
Understand variable syntax	<table><tr><th>SYNTAX</th><th>EXAMPLE</th><th>WHEN IS IT PROCESSED?</th><th>WHERE DOES IT EXPAND IN A PIPELINE DEFINITION?</th><th>HOW DOES IT RENDER WHEN NOT FOUND?</th></tr><tr><td>macro</td><td><code>\$(var)</code></td><td>runtime before a task executes</td><td>value (right side)</td><td>prints <code>\$(var)</code></td></tr><tr><td>template expression</td><td><code>{{ variables.var }}</code></td><td>compile time</td><td>key or value (left or right side)</td><td>empty string</td></tr><tr><td>runtime expression</td><td><code>\$(variables.var)</code></td><td>runtime</td><td>value (right side)</td><td>empty string</td></tr></table>	SYNTAX	EXAMPLE	WHEN IS IT PROCESSED?	WHERE DOES IT EXPAND IN A PIPELINE DEFINITION?	HOW DOES IT RENDER WHEN NOT FOUND?	macro	<code>\$(var)</code>	runtime before a task executes	value (right side)	prints <code>\$(var)</code>	template expression	<code>{{ variables.var }}</code>	compile time	key or value (left or right side)	empty string	runtime expression	<code>\$(variables.var)</code>	runtime	value (right side)	empty string
SYNTAX	EXAMPLE	WHEN IS IT PROCESSED?	WHERE DOES IT EXPAND IN A PIPELINE DEFINITION?	HOW DOES IT RENDER WHEN NOT FOUND?																	
macro	<code>\$(var)</code>	runtime before a task executes	value (right side)	prints <code>\$(var)</code>																	
template expression	<code>{{ variables.var }}</code>	compile time	key or value (left or right side)	empty string																	
runtime expression	<code>\$(variables.var)</code>	runtime	value (right side)	empty string																	
predefined variables	System.AccessToken:																				

	<p>Agent variables</p> <p>Agent.BuildDirectory: /home/vsts/work/1</p> <p>Agent.HomeDirectory:</p> <p>Agent.Id / Agent.JobName / Agent.JobStatus</p> <p>Agent.MachineName / Agent.Name</p> <p>Agent.OS</p> <p>Build variables</p> <p>Build.ArtifactStagingDirectory: c:\agent_work\1\A</p> <p>Build.BuildId / Build.BuildNumber</p> <p>Build.BinariesDirectory: c:\agent_work\1\B</p> <p>Build.Reason: Manual / IndividualCI / PullRequest / ResourceTrigger</p> <p>Build.Repository.Name / Build.SourceBranch / Build.SourceBranchName</p> <p>Build.StagingDirectory: The local path on the agent where any artifacts are copied to before being pushed to their destination. For example: c:\agent_work\1\A</p> <p>Pipeline variables</p> <p>Pipeline.Workspace: Workspace directory for a particular pipeline. This variable has the same value as Agent.BuildDirectory.</p>
System variables	<p>System.AccessToken</p> <p>System.DefaultWorkingDirectory</p> <p>System.JobId</p> <p>System.JobName</p>

Use conditionals with parameters

YAML

```
parameters:
- name: configs
  type: string
  default: 'x86,x64'

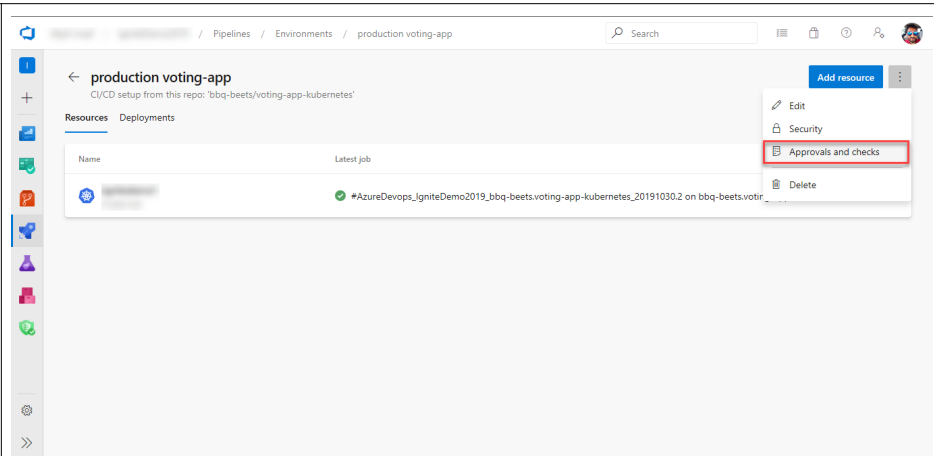
trigger: none

jobs:
- ${{ if contains(parameters.configs, 'x86') }}:
  - job: x86
    steps:
    - script: echo Building x86...
- ${{ if contains(parameters.configs, 'x64') }}:
  - job: x64
    steps:
    - script: echo Building x64...
- ${{ if contains(parameters.configs, 'arm') }}:
  - job: arm
    steps:
    - script: echo Building arm...
```

Release gates and approvals overview

By using gates, approvals, and manual intervention you can take full control of your releases to meet a wide range of deployment requirements. Typical scenarios where approvals, gates, and manual intervention are useful include the following.

	<table><tr><th>Scenario</th><th>Feature(s) to use</th></tr><tr><td>A user must manually validate the change request and approve the deployment to a certain stage.</td><td>Pre-deployment approvals</td></tr><tr><td>A user must manually sign out after deployment before the release is triggered to other stages.</td><td>Post-deployment approvals</td></tr><tr><td>A team wants to ensure there are no active issues in the work item or problem management system before deploying a build to a stage.</td><td>Pre-deployment gates</td></tr><tr><td>A team wants to ensure there are no reported incidents after deployment, before triggering a release.</td><td>Post-deployment gates</td></tr><tr><td>After deployment, a team wants to wait for a specified time before prompting users to sign out.</td><td>Post-deployment gates and post-deployment approvals</td></tr><tr><td>During deployment, a user must manually follow specific instructions and then resume the deployment.</td><td>Manual Intervention or Manual Validation</td></tr><tr><td>During deployment, a team wants to prompt users to enter a value for a parameter used by the deployment tasks, or allow users to edit the release.</td><td>Manual Intervention or Manual Validation</td></tr><tr><td>During deployment, a team wants to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment jobs.</td><td>Planned</td></tr></table>	Scenario	Feature(s) to use	A user must manually validate the change request and approve the deployment to a certain stage.	Pre-deployment approvals	A user must manually sign out after deployment before the release is triggered to other stages.	Post-deployment approvals	A team wants to ensure there are no active issues in the work item or problem management system before deploying a build to a stage.	Pre-deployment gates	A team wants to ensure there are no reported incidents after deployment, before triggering a release.	Post-deployment gates	After deployment, a team wants to wait for a specified time before prompting users to sign out.	Post-deployment gates and post-deployment approvals	During deployment, a user must manually follow specific instructions and then resume the deployment.	Manual Intervention or Manual Validation	During deployment, a team wants to prompt users to enter a value for a parameter used by the deployment tasks, or allow users to edit the release.	Manual Intervention or Manual Validation	During deployment, a team wants to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment jobs.	Planned
Scenario	Feature(s) to use																		
A user must manually validate the change request and approve the deployment to a certain stage.	Pre-deployment approvals																		
A user must manually sign out after deployment before the release is triggered to other stages.	Post-deployment approvals																		
A team wants to ensure there are no active issues in the work item or problem management system before deploying a build to a stage.	Pre-deployment gates																		
A team wants to ensure there are no reported incidents after deployment, before triggering a release.	Post-deployment gates																		
After deployment, a team wants to wait for a specified time before prompting users to sign out.	Post-deployment gates and post-deployment approvals																		
During deployment, a user must manually follow specific instructions and then resume the deployment.	Manual Intervention or Manual Validation																		
During deployment, a team wants to prompt users to enter a value for a parameter used by the deployment tasks, or allow users to edit the release.	Manual Intervention or Manual Validation																		
During deployment, a team wants to wait for monitoring or information portals to detect any active incidents, before continuing with other deployment jobs.	Planned																		
Define approvals and checks	<div><h3>Approvals</h3><p>You can manually control when a stage should run using approval checks. This is commonly used to control deployments to production environments.</p><ol style="list-style-type: none">1. In your Azure DevOps project, go to the resource (eg environment) that needs to be protected.2. Navigate to Approvals and Checks for the resource.</div>																		



3. Select Create, provide the approvers and an optional message, and select Create again to complete the addition of the manual approval check.

Branch control

Using the branch control check, you can ensure all the resources linked with the pipeline are built from the allowed branches and that the branches have protection enabled. This helps in controlling the release readiness and quality of deployments. In case multiple resources are linked with the pipeline, the source for all the resources is verified. If you have linked another pipeline, then the branch of the specific run being deployed is verified for protection.

Branch control



Display name *

Branch control

Allowed branches * ⓘ

refs/heads/master, refs/heads/releases/*

☒ Verify branch protection * ⓘ

☐ Ignore unknown protection status * ⓘ

Control options



Business hours

In case you want all deployments to your environment to happen in a specific time window only, then business hours check is the ideal solution.

Invoke Azure function

Invoke REST API

Query Azure Monitor Alerts

Required template

With the required template check, you can enforce pipelines to use a specific YAML template. When this check is in place, a pipeline will fail if it doesn't extend from the referenced template.

Release deployment control using gates

Gates allow automatic collection of health signals from external services, and then promote the release when all the signals are successful at the same time or stop the deployment on timeout.

The following gates are available by default:

- **Invoke Azure function:**
- **Query Azure monitor alerts:**
- **Invoke REST API:**
- **Query Work items:** Ensure the number of matching work items returned from a query is within a threshold. For more details, see [Work item query task](#).
- **Security and compliance assessment:** Assess Azure Policy compliance on resources within the scope of a given subscription and resource group, and optionally at a specific resource level. For more details, see [Security Compliance and Assessment task](#).

The evaluation options that apply to all the gates you've added are:

- **Time between re-evaluation of gates.** The time interval between successive evaluations of the gates. At each sampling interval, new requests are sent concurrently to each gate and the new results are evaluated. It is recommended that the sampling interval is greater than the longest typical response time of the configured gates to allow time for all responses to be received for evaluation.
- **Timeout after which gates fail.** The maximum evaluation period for all gates. The deployment will be rejected if the timeout is reached before all gates succeed during the same sampling interval.
- **Gates and approvals.** Select the required order of execution for gates and approvals if you have configured both. For pre-deployment conditions, the default is to prompt for manual (user) approvals first, then evaluate gates afterwards. This saves the system from evaluating the gate functions if the release is rejected by the user. For post-deployment conditions, the default is to evaluate gates and prompt for manual approvals only when all gates are successful. This ensures the approvers have all the information required to approve.

Switch based on platform	<div><div>YAML</div><div><div>Copy</div></div><pre>steps: # Linux - bash: export IPADDR=\$(ip addr grep 'state UP' -A2 tail -n1 awk '{print \$2}' cut -f1 -d'/') echo "##vso[task.setvariable variable=IP_ADDR]\$IPADDR" condition: eq(variables['Agent.OS'], 'Linux') displayName: Get IP on Linux # macOS - bash: export IPADDR=\$(ifconfig grep 'en0' -A3 grep inet tail -n1 awk '{print \$2}') echo "##vso[task.setvariable variable=IP_ADDR]\$IPADDR" condition: eq(variables['Agent.OS'], 'Darwin') displayName: Get IP on macOS # Windows - powershell: Set-Variable -Name IPADDR -Value ((Get-NetIPAddress ?{ \$_.AddressFamily -eq "IPv4" -and !(\$_.IPAddress -eq "0.0.0.0") } Select-Object -ExpandProperty IPAddress)[0]) Write-Host "##vso[task.setvariable variable=IP_ADDR]\$IPADDR" condition: eq(variables['Agent.OS'], 'Windows_NT') displayName: Get IP on Windows # now we use the value, no matter where we got it - script: echo The IP address is \$(IP_ADDR)</pre></div>
Pipeline caching	<div><div><h3>Maven</h3><p>Maven has a local repository where it stores downloads and built artifacts. To enable, set the <code>maven.repo.local</code> option to a path under <code>\$(Pipeline.Workspace)</code> and cache this folder.</p></div></div>

	<div><div>YAML</div><pre>variables: MAVEN_CACHE_FOLDER: \$(Pipeline.Workspace)/.m2/repository MAVEN_OPTS: '-Dmaven.repo.local=\$(MAVEN_CACHE_FOLDER)' steps: - task: Cache@2 inputs: key: 'maven "\$(Agent.OS)" **/pom.xml' restoreKeys: maven "\$(Agent.OS)" maven path: \$(MAVEN_CACHE_FOLDER) displayName: Cache Maven local repo - script: mvn install -B -e</pre></div>
Release artifacts and artifact sources	<p>A release is a collection of artifacts in your DevOps CI/CD processes. An artifact is a deployable component of your application. Azure Pipelines can deploy artifacts that are produced by a wide range of artifact sources, and stored in different types of artifact repositories.</p> <p>Artifacts are central to a number of features in Azure Pipelines. Some of the features that depend on the linking of artifacts to a release pipeline are:</p> <ul style="list-style-type: none">● Auto-trigger releases. You can configure new releases to be automatically created whenever a new version of an artifact is produced. For more information, see Continuous deployment triggers. Note that the ability to automatically create releases is available for only some artifact sources.● Trigger conditions. You can configure a release to be created automatically, or the deployment of a release to a stage to be triggered automatically, when only specific conditions on the artifacts are met. For example, you can configure releases to be automatically created only when a new build is produced from a certain branch.● Artifact versions. You can configure a release to automatically use a specific version of the build artifacts, to always use the latest version, or to allow you to specify the version when the release is created.● Artifact variables. Every artifact that is part of a release has metadata associated with it, exposed to tasks through variables. This metadata includes the version number of the artifact, the branch of code from which the artifact was produced (in the case of build or source code artifacts), the

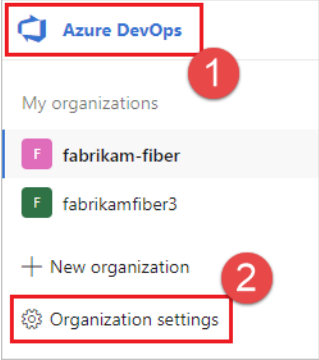
pipeline that produced the artifact (in the case of build artifacts), and more. This information is accessible in the deployment tasks. For more information, see [Artifact variables](#).

- **Work items and commits.** The work items or commits that are part of a release are computed from the versions of artifacts. For example, each build in Azure Pipelines is associated with a set of work items and commits. The work items or commits in a release are computed as the union of all work items and commits of all builds between the current release and the previous release. Note that Azure Pipelines is currently able to compute work items and commits for only certain artifact sources.
- **Artifact download.** Whenever a release is deployed to a stage, by default Azure Pipelines automatically downloads all the artifacts in that release to the [agent](#) where the deployment job runs. The procedure to download artifacts depends on the type of artifact. For example, Azure Pipelines artifacts are downloaded using an algorithm that downloads multiple files in parallel. Git artifacts are downloaded using Git library functionality. For more information, see [Artifact download](#).

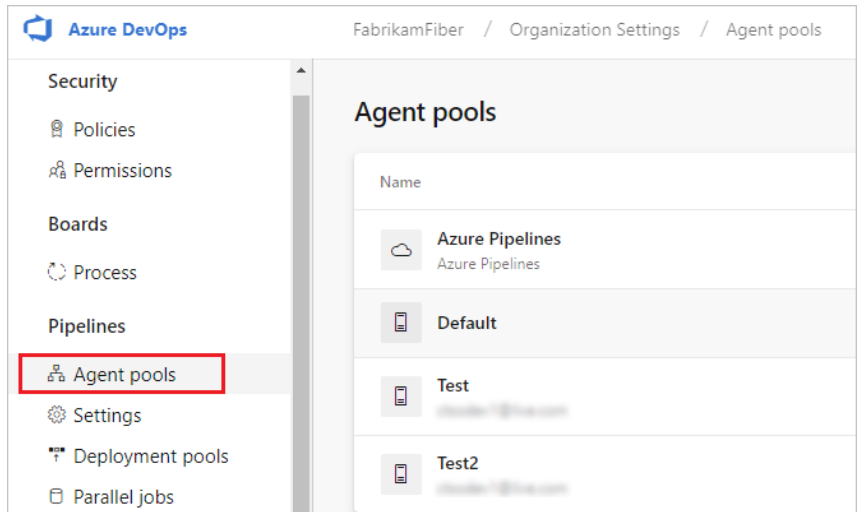
Artifact sources

There are several types of tools you might use in your application lifecycle process to produce or store artifacts. For example, you might use continuous integration systems such as Azure Pipelines, Jenkins, or TeamCity to produce artifacts.

	<div><div><div>All definitions > Fabrikam</div><div>PipelineTasksVariablesRefr</div><div><div>Artifacts + Add</div><div><div>SampleWebAppWithTestsBuild</div><div>Schedule not set</div></div></div></div><div><div>Add artifact</div><div>Source type</div><div><div>✓ Build</div><div>Azure Repos...</div><div>GitHub</div><div>TFVC</div></div><div>3 more artifact types</div><div>Project * ⓘ</div><div>Fabrikam</div><div>Source (Build definition) * ⓘ</div><div>Fabrikam.CI</div><div>Default version * ⓘ</div><div>Latest</div><div>Latest from build definition default branch with tags</div><div>Latest from specific branch with tags</div><div>Specific version</div><div>Specify at the time of release creation</div><div>Add</div></div></div>
Manage Agents and agent pools	<p>Azure Pipelines agents:</p> <p>Microsoft-hosted agents: maintenance and upgrade taken care by Microsoft.</p> <p>Self-hosted agents</p> <p>Azure virtual machine scale set agents</p> <p>Parallel jobs: Parallel jobs represent the number of jobs you can run at the same time in your organization. If your organization has a single parallel job, you can run a single job at a time in your organization, with any additional concurrent jobs being queued until the first job completes. To run two jobs at the same time, you need two parallel jobs.</p> <p>Capabilities: Every self-hosted agent has a set of capabilities that indicate what it can do. Capabilities are name-value pairs that are either automatically discovered by the agent software, in which case they are called system capabilities, or those that you define, in which case they are called user capabilities.</p>

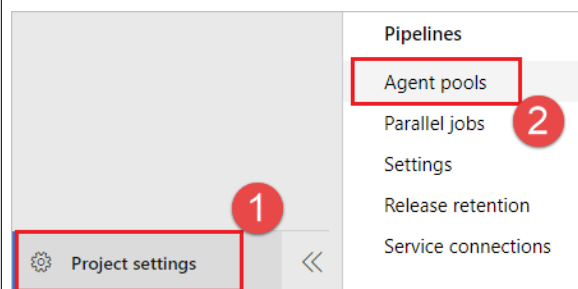
Create and manage agent pools	<p>An agent pool is a collection of agents. Instead of managing each agent individually, you organize agents into agent pools. When you configure an agent, it is registered with a single pool, and when you create a pipeline, you specify the pool in which the pipeline runs. When you run the pipeline, it runs on an agent from that pool that meets the demands of the pipeline.</p> <p>If you are an organization administrator, you create and manage agent pools from the agent pools tab in admin settings.</p> <p>1. Choose Azure DevOps, Organization settings.</p> 

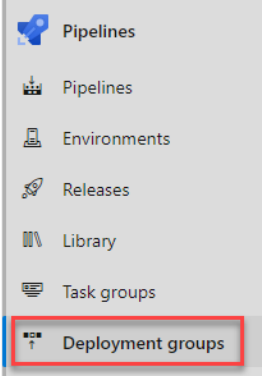
2. Choose Agent pools.



If you are a project team member, you create and manage agent queues from the agent pools tab in project settings.

Navigate to your project and choose Project settings, Agent pools.



Default agent pools	<p>The following agent pools are provided by default:</p> <ul style="list-style-type: none">• Default pool: Use it to register self-hosted agents that you've set up.• Azure Pipelines hosted pool with various Windows, Linux, and macOS images. For a complete list of the available images and their installed software, see Microsoft-hosted agents.
Provision deployment groups	<p>A deployment group is a logical set of deployment target machines that have agents installed on each one. Deployment groups represent the physical environments; for example, "Dev", "Test", or "Production" environments. In effect, a deployment group is just another grouping of agents, much like an agent pool.</p>  <p>The screenshot shows a vertical navigation menu with the following items: Pipelines (with a blue icon), Pipelines (with a server rack icon), Environments (with a laptop icon), Releases (with a rocket icon), Library (with a book icon), Task groups (with a document icon), and Deployment groups (with a server rack icon and a red rectangular highlight).</p>

	<div><div>Deployment groups > DeploymentGroupsDemo</div><div>DetailsTargetsSaveShareSecurityHelp</div><div><div>Deployment group name</div><div>DeploymentGroupsDemo</div></div><div><div>Type of target to register</div><div>Windows</div><div>Registration script (PowerShell)</div></div><div><div>Description</div><div></div></div><div><div>Deployment pool</div><div>JavaScript-Docker-DeploymentGroupsDemo</div><div>Manage</div></div><div><div>1.</div></div><div><div><div>\$?ErrorActionPreference="Stop";If (-NOT ([Security.Principal.WindowsPrincipal][Security.Principal.WindowsIdentity]::GetCurrent()).IsInRole([Security.Principal.WindowsBuiltInRole] "Administrator")){ throw "Run command in an administrator PowerShell prompt";If (\$PSVersionTable.PSVersion -lt (New-Object System.Version("3.0"))){ throw "The minimum version of Windows PowerShell that is required by the script (3.0) does not match the currently running version of Windows PowerShell." };If (-NOT (Test-Path \$env:SystemDrive\azagent)){mkdir \$env:SystemDrive\azagent}; cd \$env:SystemDrive\azagent; for (\$i=1; \$i -lt 100; \$i++){\$destFolder="A"\$i.ToString();If (NOT (Test-Path \$destFolder)){mkdir \$destFolder;cd \$destFolder;break}}; \$agentZip="\$PMAgent.zip";\$defaultProxy=[System.Net.WebRequest]::DefaultWebProxy;\$securityProtocol=@();\$securityProtocol+=[Net.ServicePointManager]::SecurityProtocol;\$securityProtocol+=[Net.SecurityProtocolType]::Tls12;[Net.ServicePointManager]::SecurityProtocol=\$securityProtocol;\$webClient=New-Object Net.WebClient; \$url="https://vstsagentpackage.azureedge.net/agent/2.188.1/vsts-agent-win-x64-2.188.1.zip";If (\$defaultProxy -and (not \$defaultProxy.IsBypassed(\$url))){\$webClient.Proxy= New-Object Net.WebProxy(\$defaultProxy.GetProxy(\$url).OriginalString, \$true)};\$webClient.DownloadFile(\$url, \$agentZip);Add-Type -AssemblyName System.IO.Compression.FileSystem;[System.IO.Compression.ZipFile]::ExtractToDirectory(\$agentZip, "PMA");.config.cmd --deploymentgroup --deploymentgroupname "DeploymentGroupsDemo" --agent \$env:COMPUSERHOME --runaservice --work ".\work" -url "https://dev.azure.com/ram10sf/DevOps/" --projectname "JavaScript-Docker"; Remove-Item \$agentZip;</div><div><div><input type="checkbox"/> Use a personal access token in the script for authentication</div><div><input type="checkbox"/> Run from an administrator PowerShell command prompt</div></div><div><div>Copy script to the clipboard</div></div></div></div></div>
https://docs.google.com/document/d/1slyrNhYL9RXtL-4KiWc7GakHX8YSSohL5OMvc-nWEo/ed-it#heading=h.d525rny21sfp	

Azure Pipeline Table

Stages, Jobs, Steps	<ul style="list-style-type: none">• Pipeline consists of one or more stages. Stages run sequentially.• Stage is a way of organizing Jobs. Jobs run parallelly on one or more agents.• Each agent runs a job that contains one or more steps.• A step can be a job or script that is the smallest building block of the pipeline. It can be a task or script.
---------------------	---

Stage Syntax	
Job Syntax	<pre>- job: string # name of the job, A-Z, a-z, 0-9, and underscore displayName: string # friendly name to display in the UI dependsOn: string [string] condition: string strategy: parallel: # parallel strategy matrix: # matrix strategy maxParallel: number # maximum number simultaneous matrix legs to run # note: `parallel` and `matrix` are mutually exclusive # you may specify one or the other; including both is an error # `maxParallel` is only valid with `matrix` continueOnError: boolean # 'true' if future jobs should run even if this job fails; defaults to 'false' pool: pool # agent pool workspace: clean: outputs resources all # what to clean up before the job runs container: containerReference # container to run this job inside timeoutInMinutes: number # how long to run the job before automatically cancelling cancelTimeoutInMinutes: number # how much time to give 'run always even if cancelled tasks' before killing them variables: { string: string } [variable variableReference] steps: [script bash pwsh powershell checkout task templateReference] services: { string: string container } # container resources to run as a service container</pre>
Step Syntax	<pre>- task: string # reference to a task and version, e.g. "VSBUILD@1" condition: expression # see below continueOnError: boolean # 'true' if future steps should run even if this step fails; defaults to 'false' enabled: boolean # whether or not to run this step; defaults to 'true' timeoutInMinutes: number # how long to wait before timing out the task target: string # 'host' or the name of a container resource to target</pre>
Artifact	<ul style="list-style-type: none">• An artifact is a collection of files or packages published by a run. <p>Upload build artifact</p>

	<div><pre>- task: PublishBuildArtifacts@1 inputs: pathToPublish: '\$(System.DefaultWorkingDirectory)' artifactName: WebSite</pre></div> <p>Download artifact</p> <div><pre>- task: DownloadBuildArtifacts@0 displayName: 'Download Build Artifacts' inputs: artifactName: WebSite downloadPath: \$(System.DefaultWorkingDirectory)</pre></div> <p>Build artifact vs Pipeline artifact</p>
Triggers	<div><pre># specific path build trigger: branches: include: - master - releases/* paths: include: - docs exclude: - docs/README.md</pre></div> <p>Pr triggers</p>

	<pre># specific path pr: branches: include: - main - releases/* paths: include: - docs exclude: - docs/README.md</pre>						
JOBS							
Jobs syntax							
Jobs strategy	<pre>strategy: parallel: # parallel strategy matrix: # matrix strategy maxParallel: number # maximum number simultaneous matrix legs to run</pre> <table><tr><th>Implementation</th><th>Description</th></tr><tr><td><code>strategy: matrix, maxParallel</code></td><td>Matrix job strategy.</td></tr><tr><td><code>strategy: parallel</code></td><td>Parallel job strategy.</td></tr></table>	Implementation	Description	<code>strategy: matrix, maxParallel</code>	Matrix job strategy.	<code>strategy: parallel</code>	Parallel job strategy.
Implementation	Description						
<code>strategy: matrix, maxParallel</code>	Matrix job strategy.						
<code>strategy: parallel</code>	Parallel job strategy.						

	<div><div>YAML</div><pre>jobs: - job: Build strategy: matrix: Python35: PYTHON_VERSION: '3.5' Python36: PYTHON_VERSION: '3.6' Python37: PYTHON_VERSION: '3.7' maxParallel: 2</pre></div> <p>This matrix creates three jobs: "Build Python35," "Build Python36," and "Build Python37." Within each job, a variable named PYTHON_VERSION is available. In "Build Python35," the variable is set to "3.5". It's likewise set to "3.6" in "Build Python36." Only two jobs run simultaneously.</p> <p>strategy: parallel:</p> <div><div>YAML</div><pre>strategy: parallel: string # Run the job this many times.</pre></div> <p>The parallel job strategy specifies how many duplicates of a job should run. parallel string. Run the job this many times.</p>
Types of Jobs: Agent pool jobs run on an agent in an agent pool. Server jobs run on the Azure DevOps Server. Container jobs run in a container on an agent in an agent pool.	Agent Pool

	<div><pre>pool: name: myPrivateAgents demands: - agent.os -equals Darwin - anotherCapability -equals somethingElse steps: - script: echo hello world</pre></div> <p>Server jobs run on the Azure DevOps Server. (Delay task, Invoke Azure function, Manual validation, Query tasks)</p> <div><pre>jobs: - job: string pool: server</pre></div> <p>Container jobs:</p>
Dependencies	<div><pre>jobs: - job: A steps: - script: exit 1 - job: B dependsOn: A condition: failed() steps: - script: echo this will run when A fails - job: C dependsOn: - A - B condition: succeeded('B') steps: - script: echo this will run when B runs and succeeds</pre></div>

Container Job

You can specify container: imagename:tag in Job.

```
pool:
  vmImage: 'ubuntu-18.04'

container: ubuntu:18.04

steps:
- script: printenv
```

You can create a reusable container definition by declaring it in resources:

```
resources:
  containers:
  - container: u16
    image: ubuntu:16.04

jobs:
- job: runincontainer
  pool:
    vmImage: 'ubuntu-18.04'
  container: u16
  steps:
  - script: printenv
```

Deployment Job

```
- deployment: string      # instead of job keyword, use deployment keyword
pool:
  name: string
  demands: string | [ string ]
environment: string
strategy:
  runOnce:
    deploy:
      steps:
        - script: echo Hi!
```

Deployment Jobs Strategy:

RunOnce deployment strategy:
runOnce is the simplest deployment strategy wherein all the lifecycle hooks are executed once.

Rolling deployment strategy: Only supported on VM resources.
rolling:
maxParallel:

Canary deployment strategy:
The canary strategy for AKS will first deploy the changes with 10-percent pods, followed by 20 percent, while monitoring the health during postRouteTraffic. If all goes well, it will promote to 100 percent.

- **Increments:**
predeploy:

Lifecycle Hooks:


```
strategy:
  runOnce:
    preDeploy:
      pool: [ server | pool ] # See pool schema.
      steps:
        - script: [ script | bash | pwsh | powershell | checkout | task | templateReference ]
    deploy:
      pool: [ server | pool ] # See pool schema.
      steps:
        ...
    routeTraffic:
      pool: [ server | pool ]
      steps:
        ...
    postRouteTraffic:
      pool: [ server | pool ]
      steps:
        ...
    on:
      failure:
        pool: [ server | pool ]
        steps:
          ...
      success:
        pool: [ server | pool ]
        steps:
          ...
```

preDeploy / deploy
routeTraffic / postRouteTraffic
on: failure or on: success: Used to run steps for rollback actions or clean-up.

STAGES

Syntax	<pre>stages: - stage: string # name of the stage, A-Z, a-z, 0-9, and underscore displayName: string # friendly name to display in the UI dependsOn: string [string] condition: string pool: string pool variables: { string: string } [variable variableReference] jobs: [job templateReference]</pre>
Dependencies	<pre>stages: - stage: A # stage B runs if A fails - stage: B condition: failed() # stage C runs if B succeeds - stage: C dependsOn: - A - B condition: succeeded('B')</pre>

Conditions

```
variables:
  isMain: ${eq(variables['Build.SourceBranch'], 'refs/heads/main')}

stages:
- stage: A
  jobs:
  - job: A1
    steps:
    - script: echo Hello Stage A!

- stage: B
  condition: and(succeeded(), eq(variables.isMain, 'true'))
  jobs:
  - job: B1
    steps:
    - script: echo Hello Stage B!
    - script: echo ${isMain}
```

Custom condition	<div>Run for the main branch, if succeeding</div> <div><pre>and(succeeded(), eq(variables['Build.SourceBranch'], 'refs/heads/main'))</pre></div> <div>Run if the branch is not main, if succeeding</div> <div><pre>and(succeeded(), ne(variables['Build.SourceBranch'], 'refs/heads/main'))</pre></div> <div>Run for user topic branches, if succeeding</div> <div><pre>and(succeeded(), startsWith(variables['Build.SourceBranch'], 'refs/heads/users/'))</pre></div> <div>Run for continuous integration (CI) builds if succeeding</div> <div><pre>and(succeeded(), in(variables['Build.Reason'], 'IndividualCI', 'BatchedCI'))</pre></div>
Variables	
Variable syntax	<p>Template expression variables (<code>\${{ variables.var }}</code>) get processed at compile time. (Choose when)</p> <p>Macro syntax variables (<code>\$(var)</code>) get processed during runtime before a task runs. (use when providing input for a task.)</p> <p>Runtime expressions (<code>\$(variables.var)</code>) also get processed during runtime but were designed for use with conditions and expressions. (Choose when working with conditions and expression.)</p>
Secret variables	You need to set secret variables in the pipeline settings UI for your pipeline. Unlike a normal variable, secret variables are not automatically decrypted into environment variables for scripts. You need to explicitly map secret variables.
Share variables across pipelines	To share variables across multiple pipelines in your project, use the web interface. Under the Library, use variable groups.

Output variables from tasks	<p>Some tasks define output variables, which you can consume in downstream steps, jobs, and stages.</p> <p>Task.setvariables Create output variable</p> <pre>echo `##vso[task.setvariable variable=doThing]Yes`</pre> <p>Other available properties: Issecret Isoutput Isreadonly</p> <p>Use output variable in subsequent steps:</p> <div><pre>steps: # This step creates a new pipeline variable: doThing. This variable will be available to subsequent steps. - bash: echo `##vso[task.setvariable variable=doThing]Yes` displayName: Step 1 # This step is able to use doThing, so it uses it in its condition - script: # You can access the variable from Step 1 as an environment variable. echo "Value of doThing (as DOTHING env var): \$DOTHING." displayName: Step 2 condition: and(succeeded(), eq(variables['doThing'], 'Yes')) # or and(succeeded(), eq(variables.doThing, 'Yes'))</pre></div> <p>Use output variable in subsequent jobs:</p> <pre>dependencies.JobName.outputs['taskname.varname']</pre> <p>Use output variable in subsequent stages:</p> <pre>stageDependencies.stageName.jobName.ouputs['taskName.varName']</pre>

Templates

Types of templates:

- Stage Template
- Job Template
- Step Template

Step Template

```
# File: templates/include-npm-steps.yml
```

```
steps:  
- script: npm install  
- script: yarn install  
- script: npm run compile
```

```
# File: azure-pipelines.yml
```

```
jobs:  
- job: Linux  
  pool:  
    vmImage: 'ubuntu-latest'  
  steps:  
    - template: templates/include-npm-steps.yml # Template reference  
- job: Windows  
  pool:  
    vmImage: 'windows-latest'  
  steps:  
    - template: templates/include-npm-steps.yml # Template reference
```

Reuse Jobs

```

-----
# File: template/npm-job.yaml
jobs:
- job: ${ parameters.name }
  pool:
    vmImage: ${ parameters.vmImage }
  steps:
  - script: npm install
    script: npm test

-----
jobs:
- template: template/npm-job.yaml
  parameters:
    name: 'windows'
    vmImage: 'windows-latest'

```

Stage reuse:

```

# File: templates/stages1.yml
stages:
- stage: Angular
  jobs:
  - job: angularinstall
    steps:
    - script: npm install angular

```

```

-----
stages:
- template: templates/stage1.yml

```

Reuse variables

```
# File: vars.yml
variables:
  favoriteVeggie: 'brussels sprouts'
```

```
# File: azure-pipelines.yml

variables:
- template: vars.yml # Template reference
```



```
# File: steps/build.yml

parameters:
- name: 'toolset'
  default: msbuild
  type: string
  values:
  - msbuild
  - dotnet

steps:
# msbuild
- ${{ if eq(parameters.toolset, 'msbuild') }}:
  - task: msbuild@1
  - task: vstest@2

# dotnet
- ${{ if eq(parameters.toolset, 'dotnet') }}:
  - task: dotnet@1
    inputs:
      command: build
  - task: dotnet@1
    inputs:
      command: test
```

Use templates	<pre># Repo: Contoso/LinuxProduct # File: azure-pipelines.yml resources: repositories: - repository: templates type: github name: Contoso/BuildTemplates jobs: - template: common.yml@templates # Template reference</pre>