

**Agenda: Continuous Deployment using Azure Pipelines**

- What is Continuous Delivery
- Connecting to Azure Subscription
- Deploying App to App Service using Designer
- Multi-State Pipeline
- Approvals and Gates
- Deploying App to Virtual Machine
- Deploying App to App Service using YAML
- Add the deployment State to the pipeline
- Deploy Apps to Specific Environment
- Deploy Azure Functions

**What is Continuous Delivery?**

**Continuous Delivery** is a software development discipline where you build software in such a way that the software can be released to production at any time

CD by itself is a set of processes, tools, and techniques that enable **rapid, reliable, and continuous delivery** of software. So, CD isn't only about setting up a pipeline, although that part is important.

**Companies need to become**

- **Better:** Quality needs to be high. Applications need to be secure. Code needs to be maintainable
- **Faster:** Customers demand features. They want it fast, otherwise they go to the competitor
- **Cheaper:** Competition is fierce and we are competing with the neighbor next door

**The Eight Principles of Continuous Delivery:**

1. We have a reliable and repeatable process for releasing and deploying software.
2. We automate as much as possible.
3. We don't put off doing something that's difficult or painful. Instead, we do it more often so that we figure out how to make it routine.
4. We keep everything in source control.
5. We all agree that *done* means *released*.
6. We build quality into the process. Quality is never an after thought.
7. We're all responsible for the release process. We no longer work in silos.
8. We always try to improve.

CD helps software teams deliver reliable software updates to their customers at a rapid cadence. CD also helps ensure that both customers and stakeholders have the latest features and fixes quickly.

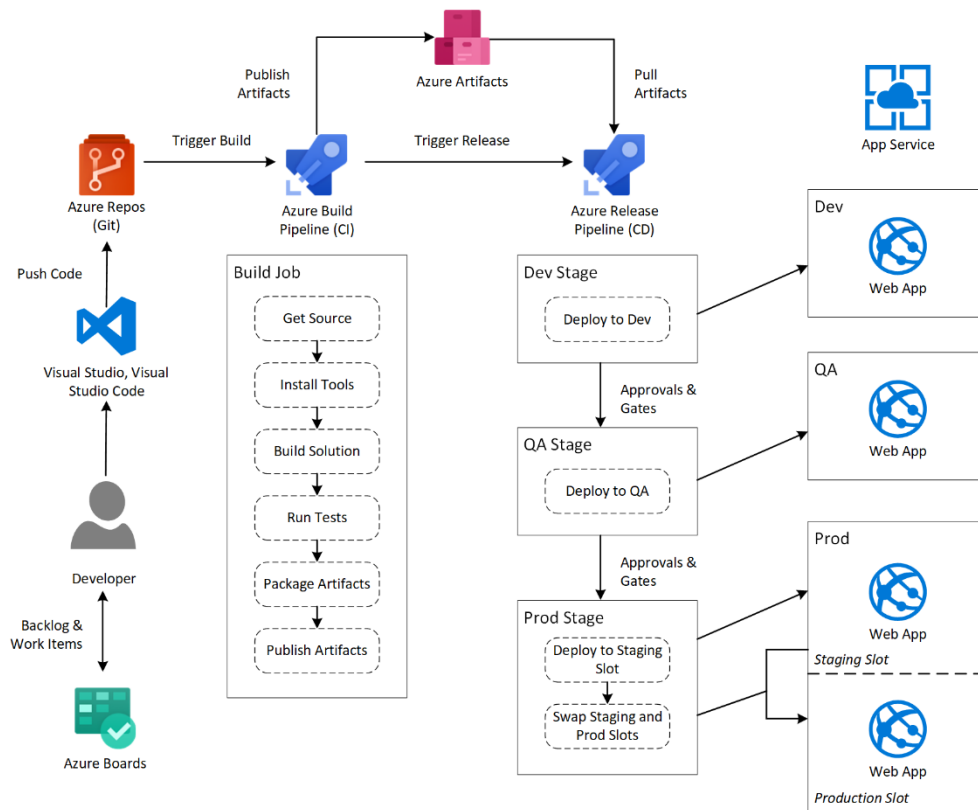
When you **right-click publish** your application, there's no guarantee that the code was properly tested or will behave as expected under real-world usage. Its good for individual but not for team of developers.

**Azure Compute Options:**

- Azure App Service - PaaS
- Virtual machines & VM Scalesets - IaaS
- Serverless computing
- Containers

**Supported build source repositories**

| Repository type          | Azure Pipelines (YAML) | Azure Pipelines (classic editor) |
|--------------------------|------------------------|----------------------------------|
| Azure Repos Git          | Yes                    | Yes                              |
| Azure Repos TFVC         | No                     | Yes                              |
| Bitbucket Cloud          | Yes                    | Yes                              |
| Other Git (generic)      | No                     | Yes                              |
| GitHub                   | Yes                    | Yes                              |
| GitHub Enterprise Server | Yes                    | Yes                              |
| Subversion               | No                     | Yes                              |



### Setup App Services and Connection to Azure Subscription in ADO

#### Create a Web App in Azure Portal

- Login to Azure Portal, <https://portal.azure.com/>
- Azure Portal → More Services → Web App → + Add
- Select Web Apps → Create
- Name = "DssDemoWebApp", Subscription = "Free Trail" Resource Group="DemoRG", App Service plan/Location=Create New Plan (Name=Standard\_Plan, Location=Central US, Pricing tier=S1 Standard.
- Application Insights=Off
- Create

#### Follow the above steps and create the below AppServices

- Create an App Service = **DssDemoApp-Dev**
- Create an App Service = **DssDemoApp-QA**
- Create an App Service = **DssDemoApp / Slot = Staging**

#### How does Azure Pipelines connect to Azure?

To deploy your app to an Azure resource, such as a virtual machine or App Service, you need a *service connection*.

**Option1: Create an Azure Resource Manager service connection using **automated** security**

1. Project Settings → (Pipeline Section) → **Service Connections** → Create a Service Connection → **Azure Resource Manager** → Next
2. Select **Service Principal Authentication** and provide the required details (Connection name, Scope level = Subscription, Subscription = <Select>, Resource Group = <Select> **(Optional)** → Check **Allow all pipelines to use this connection** → OK

**Option2: Create an Azure Resource Manager service connection **Manually****

**Create Service Principal and assign Contributor role to it for a subscription.**

1. Login to Azure Portal as an Global Administrator and Subscription Owner.
2. To Create a Service Principal: Azure Active Directory → App Registrations → **+ New registration**
  - a. Name = **AzureDevOpsServicePrincipal**.
  - b. Select Accounts in this organizational directory only (Microsoft)
  - c. Register.
3. The Service Principal should be assigned owner rights for the Subscription
  - a. Search and goto Azure Subscription
  - b. Access Control (IAM) → Add role Assignment
  - c. Role = **Contributor**, Select Service Principal (**AzureDevOpsServicePrincipal**) → Assign.
4. Copy Application (Client) ID and Secret
  - a. To go AzureDevOpsServicePrincipal → Certificates and secrets → **+ New Client secret** → Copy the secret
  - b. Overview Tab and Copy Application ID and Tenant ID
5. Go to Subscription and Copy Subscription ID and Name.

Client ID = 9cccb41d-a0fa-4304-9ea9-999402037270

Secret = Az\_aWfo~fwi-91evSos-eu5qJK1.M2Cs.e

Tenant ID = 3217245f-90ec-4ef7-b6c1-909be73fa1eb

Subscription ID = 51081bf2-da0d-4998-9462-b59b512f8690

Subscription Name = Visual Studio Enterprise – SS2

**Create a DevOps Service Connection to Azure Subscription:**

6. **Switch to Azure DevOps** → Project Settings → (Pipeline Section) → **Service Connections** → Create a Service Connection → **Azure Resource Manager** → Next
7. Service Principal (**Manual**) → Next
8. Provide the values copied earlier.
9. Verify and Create a Service Connection.

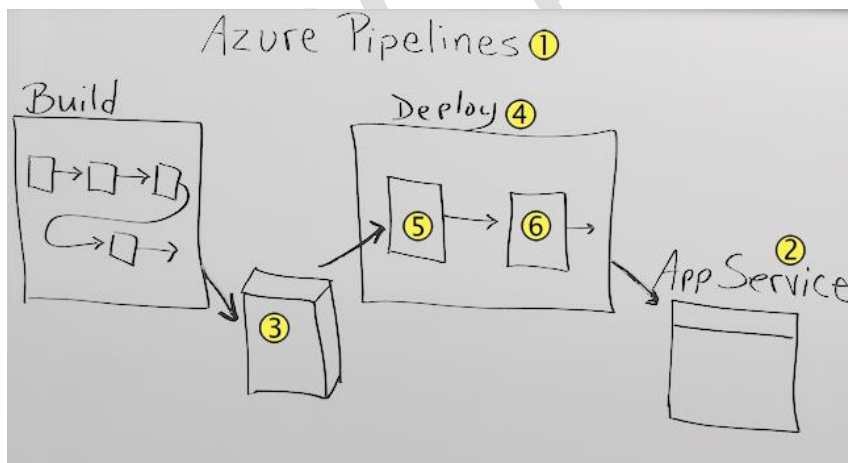
### Option3: Create an Azure Resource Manager service connection to a VM with a managed service identity

You can configure Azure Virtual Machines (VM)-based agents with an Azure Managed Service Identity in Azure Active Directory (Azure AD). This lets you use the system assigned identity (Service Principal) to grant the Azure VM-based agents access to any Azure resource that supports Azure AD, such as Key Vault, instead of persisting credentials in Azure DevOps for the connection.

Ensure that the VM (agent) has the appropriate permissions. For example, if your code needs to call Azure Resource Manager, assign the VM the appropriate role using Role-Based Access Control (RBAC) in Azure AD.

### Multi-Stage Pipeline

A *multistage pipeline* enables you to define distinct phases that your change passes through as it's promoted through the pipeline. Each stage defines the agent, variable, and steps required to carry out that phase of the pipeline. In this module, you define one stage to perform the build. You define a second stage to deploy the web application to App Service.



We use **Azure Pipelines(1)** to deploy to **App Service(2)**. To do that, we take the **build artifact(3)** as the input to the **deployment stage(4)**. The tasks in the deployment stage download the **artifact(5)** and use a service connection to **deploy(6)** the artifact to App Service.

### Step2: Add Release Stage (QA)

1. Pipelines → **Releases** → +New → New release pipeline → Select Template **Azure App Service Deployment** → Apply
2. Change Stage name = **Dev Stage**
3. View Stage tasks: Click on **1 job, task**, Select Azure Subscription = <Subscription created before>, App type = Web App on Linux, App service name = **Dssdemoapp**
4. Check **Deploy to App Service**, App Service Name = **Dev**

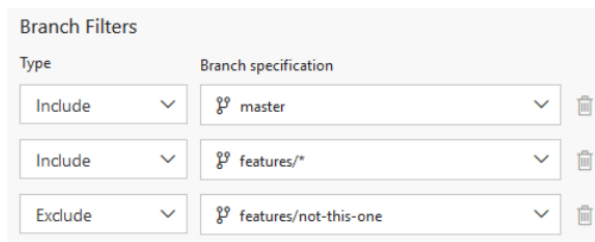
#### Optional: Enable Continuous deployment triggers:

Pipelines → **Releases** → Select Release Pipeline → Edit → In **Artifacts Section** click on Icon Continuous deployment trigger

In Continuous deployment trigger → **Enabled**

For example:

You want your build to be triggered by changes in master and most, but not all, of your feature branches.



| Type    | Branch specification  |
|---------|-----------------------|
| Include | master                |
| Include | features/*            |
| Exclude | features/not-this-one |

#### Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

5. Choose **+ Release** and select **Create a release**.
6. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
7. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
8. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
9. After the release is complete, navigate to your app and verify its contents.

#### Step 3: Adding Another Stage (QA)

10. Edit the Release Pipeline
11. Create a copy of existing Dev stage
12. Change Name to **QA Stage** → Go to Tasks tab → Check **Deploy to App Service** = **QA**

**Create a release to deploy your app.**

13. Pipelines → **Releases** → Select the Pipeline → Click **Create release** → Create
14. Choose the release link in the information bar message. For example: "Release **Release-2** has been created".
15. After the release is complete, navigate to your app and verify its contents.

Note: Choose the **Pre-deployment conditions** icon for the **UAT Stage** and note that: Select trigger = **After stage**, and Stages = **Dev Stage** in the **Stages** drop-down list

**Step 4: Adding Another Stage (Production).**

16. Edit the Release Pipeline
17. Create a **clone** of existing UAT Stage
18. Change Name to **Production**
19. Go to Tasks tab → Select task **Deploy Azure App Service** → Check **Deploy to slot** → Select Resource Group Name and Slot Name = **PreProd**
20. Add a task → Search = Azure App Service manage → Add
21. Select Azure Subscription, **Action** = **Swap Slot**, App Service name = "DssDemoApp", Resource Group="DemoRG", Source Slot = **PreProd** and check **Swap with Production**
22. Save

**Create a release to deploy your app.**

23. Pipelines → **Releases** → Select the Pipeline → Click **Create release** → Create
24. View Logs and test the deployment

**Deploy ASP.NET Core App to a Windows Virtual Machine using Deployment Group****Step1: Install the Prerequisites to run .NET Core Web App**

1. Create an Azure VM (Windows OS)

On your VM, open an **Administrator: Windows PowerShell** console. Install IIS and the required .NET features:

**2. Install IIS**

[Install-WindowsFeature Web-Server, Web-Asp-Net45, NET-Framework-Features](#)

OR

Server Manager → Add roles and features → Next ... → Select Webserver (IIS) → OK.

3. Install the .NET Core Windows Server Hosting bundle:

```
Invoke-WebRequest https://download.visualstudio.microsoft.com/download/pr/24847c36-9f3a-40c1-8e3f-4389d954086d/0e8ae4f4a8e604a6575702819334d703/dotnet-hosting-5.0.6-win.exe -outfile
```

```
$env:temp\DotNetCore.WindowsHosting.exe
```

```
Start-Process $env:temp\DotNetCore.WindowsHosting.exe -ArgumentList '/quiet' -Wait
```

OR

Use Browser:

- a) In browser, navigate to the [.NET download archives](https://dotnet.microsoft.com/download/dotnet-core/3.1)  
(https://dotnet.microsoft.com/download/dotnet-core/3.1)
- b) Under **.NET Core**, select the .NET Core version.
- c) Download the installer using the **Hosting Bundle** link

4. Restart the web server so that system PATH updates take effect

```
net stop was /y
```

```
net start w3svc
```

## Step2: Create a deployment group in DevOps Project

- Deployment groups in Azure Pipelines make it easier to organize the servers that you want to use to host your app.
- A deployment group is a collection of machines with an **Azure Pipelines agent** on each of them. Each machine interacts with Azure Pipelines to coordinate deployment of your app.
- Deployment groups represent the physical environments; for example, "Dev", "Test", "UAT", and "Production".
- In effect, a deployment group is just another grouping of agents, much like an agent pool.

You can install the agent in any one of these ways:

1. [Run the script](#) that is generated automatically when you create a deployment group.
2. [Install the Azure Pipelines Agent Azure VM extension](#) on each of the VMs.
3. [Use the Azure Resource Group Deployment task](#) in your release pipeline (ARM Templates)

### Example with Option1: Run the Script

1. Azure Pipelines → **Deployment groups**.
2. Click **Add Deployment group**.
3. Name = *myIIS* → **Create**.
4. In the **Register machine** section, Select **Windows**, Check **Use a personal access token in the script for authentication** and click on **Copy script to clipboard**.

Note: The script that you've copied to your clipboard will download and configure an agent on the VM so that it can receive new web deployment packages and apply them to IIS.



5. On your VM, in an **Administrator PowerShell** console, paste and **run the script**.
6. When you're prompted to configure tags for the agent, press Enter (you don't need any tags).
7. When you're prompted for the user account, press Enter to accept the defaults.
8. When the script is done, it displays the message *Service vstsagent.account.computername started successfully*.
9. Azure Portal → On the **Deployment groups** page in Azure Pipelines, open the *myIIS* deployment group. On the **Targets** tab, verify that your VM is listed.

### Step 3: Create a release pipeline to deploy your app

Your CD release pipeline picks up the artifacts published by your CI build and then deploys them to your IIS servers.

1. Open the **Releases** tab of **Azure Pipelines**, open the + drop-down in the list of release pipelines, and choose **Create release pipeline**.
2. Select the **IIS Website Deployment** template and choose **Apply**.
3. Choose the + **Add** link and select your build artifact.
4. Choose the **Continuous deployment** icon in the **Artifacts** section, check that the continuous deployment trigger is enabled, and add a filter to include the **master** branch.
5. Open the **Tasks** tab and select the **IIS Deployment** job. For the **Deployment Group**, select the deployment group you created earlier (such as *myIIS*).
6. Save the release pipeline.

### Step 4: Create a release to deploy your app

You're now ready to create a release, which means to run the release pipeline with the artifacts produced by a specific build. This will result in deploying the build:

1. Choose + **Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In the pipeline view, choose the status link in the stages of the pipeline to see the logs and agent output.
5. After the release is complete, navigate to your app and verify its contents.

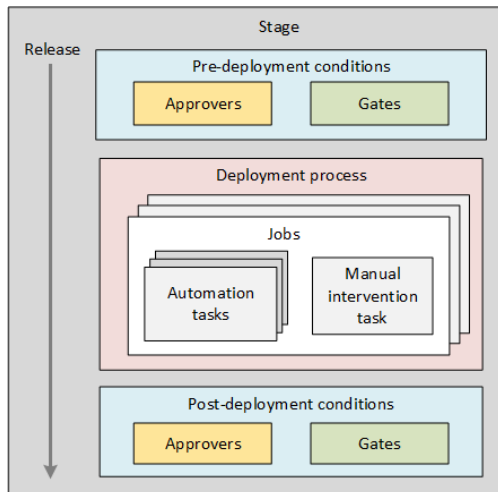
### VM with Linux OS

<https://docs.microsoft.com/en-us/azure/devops/pipelines/apps/cd/deploy-linuxvm-deploygroups>

### Approvals and Gates (In Classic Pipeline only)

**Approvals** and **gates** give you additional control over the start and completion of the deployment pipeline.

Each stage in a release pipeline can be configured with pre-deployment and post-deployment conditions that can include waiting for users to manually approve or reject deployments, and checking with other automated systems until specific conditions are verified. In addition, you can configure a manual intervention to pause the deployment pipeline and prompt users to carry out manual tasks, then resume or reject the deployment.



#### Step 6: Add approvals within a release pipeline for Production Stage

25. Pipelines → Releases → Select Release Pipeline → Edit
26. Click **Pre-deployment conditions** icon in the **Production** section to open the conditions panel.
27. Enable **Pre-deployment approvers** section, Select Approvers
28. Save → Comment = Approvers added → OK

#### Step 7: Working with 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 .
- You can use gates to ensure that the release meets a wide range or criteria, without requiring user intervention.
- Typically, gates are used in connection with incident management, problem management, change management, monitoring, and external approval systems.

#### Create a Work Items Query

29. Azure Boards → Queries → +New Query →
30. WorkItemType = Bugs and Status <> Closed and Status <> Resolved
31. Save Query → Query Name = ActiveBugs, Folder = **Shared**

**Create a Gate**

32. Pipelines → Releases → Select Release Pipeline → Edit

33. Click **Pre-deployment conditions** icon in the **Production** Stage to open the conditions panel

34. **Enable gates** by using the switch control in the Gates section.

To allow gate functions to initialize and stabilize (it may take some time for them to begin returning accurate results), you configure a delay before the results are evaluated and used to determine if the deployment should be approved or rejected.

35. Set **The delay before evaluation** = 1 Minute

36. Choose **+ Add** and select the **Query Work Items** gate.

37. In Deployment gates window: Select **Query = Active Bugs, Upper threshold = 0**

38. **Expand Evaluation options**, re-evaluation = 15 minutes . . . **Check Before gates, ask for approvals\***

39. Save your release pipeline.

**\* Gates and approvals Ordering.** 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 for a sign-off.

**Step 8: Configure a manual intervention (Job)**

Sometimes, you may need to introduce manual intervention into a release pipeline. For example, there may be tasks that cannot be accomplished automatically such as confirming network conditions are appropriate, or that specific hardware or software is in place, before you approve a deployment. You can do this by using the Manual Intervention task in your pipeline.

40. In the release pipeline editor, open the **Tasks** editor for the **QA** stage.

41. Choose the ellipses (...) in the **QA** deployment pipeline bar and then choose **Add agentless job**.

42. Drag and drop the new agentless job to the start of the QA process, before the existing agent job.

43. Choose **+** in the **Agentless job** bar and add a **Manual Intervention** task to the job.

44. Configure the task by entering a message to display when it executes and pauses the release pipeline.

1. Display name = "Manual Intervention"
2. Instructions = "Ensure QA database updates have been completed before continuing deployment"

45. Save the release pipeline

**Note:** [Error regarding query work item gate in azure pipeline - Stack Overflow](#)

**Create Release and view the logs for approvals**

46. Create **New Release** and Open Summary
47. You'll see the live status for each step in the release pipeline. It indicates that a **manual intervention is pending**
48. Choose the **Resume** link.
49. You see the intervention message, and can choose to resume or reject the deployment. Enter some text response to the intervention and choose **Resume**.
50. Go back to the pipeline view of the release. After deployment to the QA stage succeeds, you see the pre-deployment approval pending message for the **Production** environment.
51. Enter your approval message and choose **Approve** to continue the deployment.
52. Go back to the pipeline view of the release. Now you see that the **gates are being processed** before the release continues.
53. After the gate evaluation has successfully completed, the deployment occurs for the Production stage.

**Note:** For pre-deployment conditions, the default is to prompt for manual (user) approvals first, then evaluate gates afterwards.