Agenda: Working with AKS

- Creating an AKS Cluster
- Writing Deployment and Service YAML files
- Deploying the Application using Kubectl
- Building a CI and CD Pipeline for Deploying to Kubernetes Cluster.

Azure Kubernetes Service (AKS)

- AKS makes it quick and easy to deploy and manage containerized applications without container orchestration expertise.
- AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure.
- As a hosted Kubernetes service, Azure handles critical tasks like provisioning, upgrading, scaling, health monitoring and maintenance for you.
- In addition, the service is free, you only pay for the agent nodes within your clusters, not for the masters.

Steps Required for Running an Application in Kubernetes

1. Package your Application in to one or more containers/images
2. Push those images to an image registry like Docker Hub or ACR
3. Post App Descriptor (YAML) to the Kubernetes API Server
    a. Scheduler schedules containers on available Worker Nodes
    b. Kubelet instructs Nodes to download Container Images
    c. Kubelet instructs Nodes to run the Containers

Walkthrough:

Step 1: Develop and Test the application locally

Step2: Deploy the docker images in ACR

Step3: Create Kubernetes Cluster

Step4: Run the application developed in step 1

Step 1: Develop and Test the application locally

1. Execute the following commands to create an ASP.NET Core MVC project.

```
D:\>md HelloWebApp
D:\>cd HelloWebApp
D:\HelloWebApp:\> dotnet new mvc
D:\HelloWebApp:\> Code .
```

2. Add the following docker file to the project

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["HelloWebApp.csproj", ""]
RUN dotnet restore "./HelloWebApp.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "HelloWebApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "HelloWebApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build the docker image

   D:\HelloWebApp>**docker build** -t sandeepsoni/hellowebapp:v1 .

4. Test the image locally

   D:\HelloWebApp>**docker run** -p "8080:80" sandeepsoni/hellowebapp:v1

   Visit: http://localhost:8080

Step2: Create Container Registry and Push the Image

5. Login to Azure

```
az login --tenant "sandeepsonideccansoft.onmicrosoft.com"
az account set --subscription "<subscription guid>"
```

6. Create Resource Group

```
az group create --name DemoKRG --location eastus
```

7. Create Azure Container Registry

```
az acr create --resource-group DemoKRG --name dssregistry --sku Standard --location eastus
```

8. Azure Portal → Goto ACR → Access Keys → Under **Admin user**, select **Enable**. Take note of the following values:
    - Login server
    - Username
    - Password
9. Local Machine: Login to ACR

```
docker login --username dssregistry --password oZ2+N4QW+DmhwDvLe5pDFF/9oMB0PS/r
dssregistry.azurecr.io
```

10. Tag Local Images

```
docker image tag sandeepsoni/hellowebapp:v1 dssregistry.azurecr.io/hellowebapp:v1
```

11. Push images to ACR

```
docker push dssregistry.azurecr.io/hellowebapp:v1
```

Step3: Create Service Principal and Assign AcrPull Role over the Container Registry

12. Create Service Principal in Azure Active Directory.

Azure Portal → Azure Active Directory → App Registration → + Register App

Name = **KubernetesServicePrincipal,**

OK.

Note the service principal ClientID and create a new Secret.

OR

az ad sp create-for-rbac -n KubernetesServicePrincipal --skip-assignment

output

```
{
  "appId": "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4",
  "displayName": "azure-cli-2018-09-18-10-52-15",
  "name": "http://azure-cli-2018-09-18-10-52-15",
  "password": "ffb12981-f5f1-44d4-8fa1-e3b0aa609936",
  "tenant": "82d8af3b-d3f9-465c-b724-0fb186cc28c7"
}
```

13. Assign Role **AcrPull** to Service Principal to read images from Azure Container Registry

    Azure Portal → Go to Azure Contaner Registry → Access Control (IAM) → Add Role Assignment

    Role = **AcrPull**

    Service Principal = KubernetesServicePrincipal

    Assign.

OR

1. Get the ACR Id

az acr show --resource-group DemoKRG --name dssregistry --query "id" --output tsv

output:

/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry

2. Assign contributor Role to Service Principal in Azure AD

az role assignment create --assignee "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --scope "/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry" **--role AcrPull**

Step3: Create Azure Kubernetes (AKS) Cluster

14. Create **Azure Kubernetes Service** with associated **Service Principal**

CLI:

az aks create --resource-group DemoKRG --name **dssdemocluster** --node-count 1 --generate-ssh-keys --service-principal "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --client-secret "ffb12981-f5f1-44d4-8fa1-e3b0aa609936"

OR

PowerShell:

$user = "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4"

$password = ConvertTo-SecureString -String "ffb12981-f5f1-44d4-8fa1-e3b0aa609936" -AsPlainText -Force

$Credential = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $user, $password

New-AzAksCluster -ResourceGroupName DemoKRG -Name dssdemocluster -NodeCount 2 -GenerateSshKey -ServicePrincipalIdAndSecret $credential

Note: Delete key related files in this folder if existing: C:\Users\<Current User>\.ssh

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

15. To connect to Kubernetes Cluster from local machine

az aks get-credentials --resource-group DemoKRG --name **dssdemocluster**

16. **Configure Kubernetes to use your ACR** When creating deployments, replicasets or pods, kubernetes will try to use docker images already stored locally or pull them from the public docker hub. To change this, you need to specify the custom docker registry as part of your k8s object configuration (yaml or json).

Instead of specifying this directly in your configuration, we'll use the concept of k8s **secrets**. You decouple the k8s object from the registry configuration by just referencing the secret by it's name. But first, let's create a new k8s secret.

Note: It's optional for Azure Container Registry because the Service Principal used while creating the AKS cluster is already assigned AcrPull permission over the Azure Container Registry. But if Docker Hub Private image is used then it's mandatory to use secret for pulling image.

kubectl create secret docker-registry **kubernetes-principle-secret** --docker-server dssregistry**.azurecr.io** --docker-username="<Service Principal CLIENTID>" --docker-password "<Service Pricipal Secret>"

Kubernetes manifest files define a desired state for a cluster, including what container images should be running.

17. Create a file DeploymentAndService.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellowebapp-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hellowebapp
  template:
    metadata:
      labels:
        app: hellowebapp
    spec:
      containers:
      - name: hellowebapp
        image: dssregistry2022.azurecr.io/hellowebapp:v1
        imagePullPolicy: Always
        ports:
        - containerPort: 80
        - containerPort: 443
      imagePullSecrets:
      - name: kubernetes-principle-secret

---
```

```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

18. Use the kubectl create command to run the application.

```
kubectl apply -f deployment.yaml
```

Test the application

19. To monitor progress, use the the below command.

    kubectl get service azure-vote-front –watch

    Once the *EXTERNAL-IP* address has changed from *pending* to an *IP address*, use CTRL-C to stop the kubectl watch process.

20. Open a web browser to the **external IP address** of your service.

Publish to AKS Cluster using Kubernetes Manifest Task

Step 1: Add all the Kubernetes Deployments to the folder **HelloWorldApp.AKSYamls**

Eg:

HelloWorldApp.Deployments\Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworldapp
  labels:
    app: helloworldapp
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: helloworldapp
  template:
    metadata:
      labels:
        app: helloworldapp
    spec:
      containers:
      - name: helloworldapp
        image: helloworldapp #This is overridden by pipeline and should not have tag name mentioned.
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

HelloWorldApp.Deployments\Service.yaml

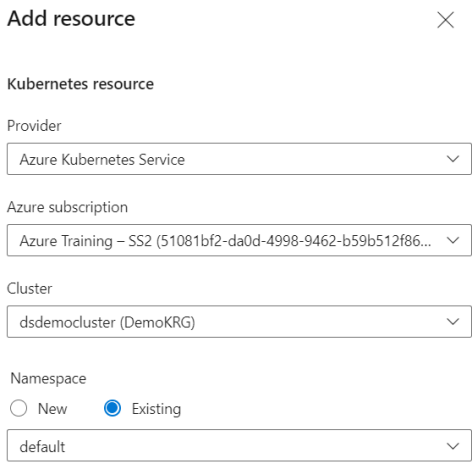
```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

Step 2: Create the Service Connections

1. Azure Container Registery (Name=DssAzureContainerRegistry)
    a. DevOps Project → Project Properties → Service Connection → New service connection → search Docker Registry → Next

Step 3: Create Environment

1. Pipelines → Environment → New Environment →
2. Environment Name = HelloWorldApp, Select Kubernetes → Create
3. Select Environment = **HelloWorldApp** → Add resource → Provider = Azure Kubernetes Service, Select Subscription and AKS Cluster and Namespace = default → Create



Step 4: Create a New YAML Pipeline as below

```
trigger:
- none

variables:
 # Container registry service connection established during pipeline creation
 dockerRegistryServiceConnection: 'DssAzureContainerRegistry'
 imageRepository: 'helloworldapp'
 containerRegistry: 'dssdemoregistry.azurecr.io'
 dockerfilePath: '**/Dockerfile'
 tag: '$(Build.BuildId)'
 imagePullSecret: 'dssregistry4568950f942-auth'

 # Agent VM image name
 vmImageName: 'ubuntu-latest'

stages:
- stage: Build
 displayName: Build stage
 jobs:
 - job: Build
  displayName: Build
  pool:
   vmImage: $(vmImageName)
  steps:
  - task: Docker@2
```

```
    displayName: Build and push an image to container registry
    inputs:
      command: buildAndPush
      repository: $(imageRepository)
      dockerfile: $(dockerfilePath)
      buildContext: '.'
      containerRegistry: $(dockerRegistryServiceConnection)
      tags: |
        $(tag)

  - task: CopyFiles@2
    inputs:
      SourceFolder: 'HelloWorldApp.AKSYamls'
      Contents: '**'
      TargetFolder: '$(Build.ArtifactStagingDirectory)'

  - task: PublishPipelineArtifact@1
    inputs:
      targetPath: '$(Build.ArtifactStagingDirectory)'
      artifact: 'drop'
      publishLocation: 'pipeline'

  # - upload: manifests
  #   artifact: manifests

- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: Deploy
    pool:
      vmImage: $(vmImageName)
    environment: 'HelloWorldApp.default'
    strategy:
      runOnce:
```

```
deploy:
 steps:
 - task: KubernetesManifest@0
  displayName: Create imagePullSecret
  inputs:
   action: createSecret
   secretName: $(imagePullSecret)
   dockerRegistryEndpoint: $(dockerRegistryServiceConnection)

 - task: KubernetesManifest@0
  displayName: Deploy to Kubernetes cluster
  inputs:
   action: deploy
   manifests: |
    $(Pipeline.Workspace)/drop/Deployment.yaml
    $(Pipeline.Workspace)/drop/Service.yaml
   imagePullSecrets: |
    $(imagePullSecret)
   containers: |
    $(containerRegistry)/$(imageRepository):$(tag)
```

Step 5: Test the Deployment

1. **Go to Kubernetes Cluster → Services and ingresses → Click on External IP of the Service**

Name	Namespace	Status	Type	Cluster IP	External IP	Ports
kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP
kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,5
metrics-server	kube-system	Ok	ClusterIP	10.0.43.73		443/TCP
hellowebapp	default	Ok	LoadBalancer	10.0.221.211	20.121.252.205	80:31500/

Publish to Azure Kubernetes Service using Kubernetes Task

1. Add the YML file to manifests folder of the Azure Repository

manifests/Deployment.yaml

Edit the deployment.yaml file update the containers section as below

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: helloworldapp
  labels:
    app: helloworldapp
spec:
  replicas: 1
  selector:
     matchLabels:
       app: helloworldapp
  template:
    metadata:
      labels:
        app: helloworldapp
    spec:
      containers:
      - name: helloworldapp
        image: #{ImageName}#
        imagePullPolicy: Always
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: kubernetes-principle-secret
```

Manifests/Service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

2. Create following Service Connections

a. Azure Subscription (Name = AzureServiceConnection-SS2)

i. DevOps Project → Project Properties → Service Connection → New service connection → Azure Resouce Manager → Next

b. Azure Container Registery (Name=DssAzureContainerRegistry)

i. DevOps Project → Project Properties → Service Connection → New service connection → search Docker Registry → Next

c. Azure Kubernetes Cluster (Name=KubernetesServiceConnection)

i. DevOps Project → Project Properties → Service Connection → New service connection → search Kubernetes → Next

ii. Authentication Method = Azure Subscription

iii. Name=default

3. Create a YAML Azure DevOps Pipeline

```
trigger:
- none

variables:
 vmImageName: 'ubuntu-latest'
 azureResourceGroup: "Demo-rg"
 azureSubscriptionEndPoint: 'Azure - SS1'
 dockerRegistryServiceConnection: 'DockerRegistryConnection'
 kubernetesServiceConnection: 'KubernetesServiceConnection'
 dockerfilePath: 'HelloWorldApp.Web/Dockerfile'
 containerRegistry: dssdemoregistry.azurecr.io
 imageRepository: 'helloworldapp'
 tag: $(Build.BuildId)
 secretName: 'kubernetes-principle-secret'
 kubernetesCluster: 'dssdemocluster'
 ImageName: '$(containerRegistry)/$(imageRepository):$(tag)'

stages:
- stage: Build
 displayName: Build and push stage
 jobs:
 - job: Build
```

```
displayName: Build
pool:
 vmImage: $(vmImageName)
steps:
- task: Docker@2
 displayName: Build and push an image to container registry
 inputs:
  containerRegistry: '$(dockerRegistryServiceConnection)'
  repository: '$(imageRepository)'
  command: 'buildAndPush'
  Dockerfile: '$(dockerfilePath)'
  buildContext: '.'
  tags: '$(tag)'

- task: replacetokens@4
 inputs:
  targetFiles: '**/deployment.yml'
  encoding: 'auto'
  tokenPattern: 'default'
  writeBOM: true
  actionOnMissing: 'warn'
  keepToken: false
  actionOnNoFiles: 'continue'
  enableTransforms: false
  useLegacyPattern: false
  enableTelemetry: true

- task: replacetokens@3
 inputs:
  targetFiles: '**/deployment.yaml'
  encoding: 'auto'
  writeBOM: true
  actionOnMissing: 'fail'
  keepToken: false
  tokenPrefix: '~~'
  tokenSuffix: '~~'
```

```
      useLegacyPattern: false
      enableTransforms: false
      enableTelemetry: true

  - task: CopyFiles@2
    inputs:
      SourceFolder: 'manifests'
      Contents: '*.yml'
      TargetFolder: '$(build.artifactstagingdirectory)'

  - task: PublishPipelineArtifact@1
    inputs:
      targetPath: '$(build.artifactstagingdirectory)'
      artifact: 'drop'
      publishLocation: 'pipeline'

- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: Deploy job
    pool:
      vmImage: $(vmImageName)
    environment: 'deccansoft'
    strategy:
      runOnce:
        deploy:
          steps:
          - task: DownloadPipelineArtifact@2
            inputs:
              artifactName: 'drop'
              downloadPath: '$(System.ArtifactsDirectory)'

          - task: Kubernetes@1
            displayName: kubectl apply
```

```
    inputs:
      connectionType: 'Azure Resource Manager'
      azureSubscriptionEndpoint: $(azureSubscriptionEndPoint)
      azureResourceGroup: $(azureResourceGroup)
      kubernetesCluster: $(kubernetesCluster)
      command: 'apply'
      useConfigurationFile: true
      configuration: '$(System.ArtifactsDirectory)/deployment.yml'
      secretType: 'dockerRegistry'
      containerRegistryType: 'Azure Container Registry'
      azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndPoint)
      azureContainerRegistry: $(containerRegistry)
      secretName: $(secretName)

    - task: Kubernetes@1
      displayName: kubectl apply
      inputs:
        connectionType: 'Azure Resource Manager'
        azureSubscriptionEndpoint: $(azureSubscriptionEndPoint)
        azureResourceGroup: $(azureResourceGroup)
        kubernetesCluster: $(kubernetesCluster)
        command: 'apply'
        useConfigurationFile: true
        configuration: '$(System.ArtifactsDirectory)/service.yml'
        secretType: 'dockerRegistry'
        containerRegistryType: 'Azure Container Registry'
        azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndPoint)
        azureContainerRegistry: $(containerRegistry)
        secretName: $(secretName)
```

Note: YAML File names are case-sentive.

Create a Service Connection using Service Account for ANY Kubernetes Cluster

DevOps Project → Project Properties → Service Connection → New service connection → search Kubernetes → Next

Account Type = Service Account

1. Connect to the Kubernetes Cluster

   a. az aks get-credentials --resource-group DemoKRG --name dssaprdemocluster1

2. Server URL = Output of
   a. kubectl config view --minify -o jsonpath={.clusters[0].cluster.server}
3. Secret = Output of
   a. kubectl get serviceAccounts default -n default -o=jsonpath={.secrets[*].name}
   b. kubectl get secret <output from step3> -n default -o json
4. Service Connection Name = "Kubernetes Connection"
5. Save

Step 3 : Create a ClusterRoleBinding which gives the role **cluster-admin** to the ServiceAccount **default in default namespace (default.default)**.

```
kubectl create clusterrolebinding serviceaccounts-cluster-admin -n kube-system --clusterrole=cluster-admin --serviceaccount=default:default
```

Deploying a multi-container application to Azure Kubernetes Services

https://azuredevopslabs.com/labs/vstsextend/kubernetes/

Agenda: Working with AKS

- Creating an AKS Cluster
- Writing Deployment and Service YAML files
- Deploying the Application using Kubectl
- Building a CI and CD Pipeline for Deploying to Kubernetes Cluster.

Azure Kubernetes Service (AKS)

- AKS makes it quick and easy to deploy and manage containerized applications without container orchestration expertise.
- AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure.
- As a hosted Kubernetes service, Azure handles critical tasks like provisioning, upgrading, scaling, health monitoring and maintenance for you.
- In addition, the service is free, you only pay for the agent nodes within your clusters, not for the masters.

Steps Required for Running an Application in Kubernetes

1. Package your Application in to one or more containers/images
2. Push those images to an image registry like Docker Hub or ACR
3. Post App Descriptor (YAML) to the Kubernetes API Server
    a. Scheduler schedules containers on available Worker Nodes
    b. Kubelet instructs Nodes to download Container Images
    c. Kubelet instructs Nodes to run the Containers

Walkthrough:

Step 1: Develop and Test the application locally

Step2: Deploy the docker images in ACR

Step3: Create Kubernetes Cluster

Step4: Run the application developed in step 1

Step 1: Develop and Test the application locally

1. Execute the following commands to create an ASP.NET Core MVC project.

```
D:\>md HelloWebApp
D:\>cd HelloWebApp
D:\HelloWebApp:\> dotnet new mvc
D:\HelloWebApp:\> Code .
```

2. Add the following docker file to the project

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["HelloWebApp.csproj", ""]
RUN dotnet restore "./HelloWebApp.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "HelloWebApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "HelloWebApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build the docker image

   D:\HelloWebApp>**docker build** -t sandeepsoni/hellowebapp:v1 .

4. Test the image locally

   D:\HelloWebApp>**docker run** -p "8080:80" sandeepsoni/hellowebapp:v1

   Visit: http://localhost:8080

Step2: Create Container Registry and Push the Image

5. Login to Azure

```
az login --tenant "sandeepsonideccansoft.onmicrosoft.com"
az account set --subscription "<subscription guid>"
```

6. Create Resource Group

```
az group create --name DemoKRG --location eastus
```

7. Create Azure Container Registry

```
az acr create --resource-group DemoKRG --name dssregistry --sku Standard --location eastus
```

8. Azure Portal → Goto ACR → Access Keys → Under **Admin user**, select **Enable**. Take note of the following values:
    - Login server
    - Username
    - Password
9. Local Machine: Login to ACR

```
docker login --username dssregistry --password oZ2+N4QW+DmhwDvLe5pDFF/9oMB0PS/r
dssregistry.azurecr.io
```

10. Tag Local Images

```
docker image tag sandeepsoni/hellowebapp:v1 dssregistry.azurecr.io/hellowebapp:v1
```

11. Push images to ACR

```
docker push dssregistry.azurecr.io/hellowebapp:v1
```

Step3: Create Service Principal and Assign AcrPull Role over the Container Registry

12. Create Service Principal in Azure Active Directory.

Azure Portal → Azure Active Directory → App Registration → + Register App

Name = **KubernetesServicePrincipal,**

OK.

Note the service principal ClientID and create a new Secret.

OR

az ad sp create-for-rbac -n KubernetesServicePrincipal --skip-assignment

output

```
{
  "appId": "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4",
  "displayName": "azure-cli-2018-09-18-10-52-15",
  "name": "http://azure-cli-2018-09-18-10-52-15",
  "password": "ffb12981-f5f1-44d4-8fa1-e3b0aa609936",
  "tenant": "82d8af3b-d3f9-465c-b724-0fb186cc28c7"
}
```

13. Assign Role **AcrPull** to Service Principal to read images from Azure Container Registry

    Azure Portal → Go to Azure Contaner Registry → Access Control (IAM) → Add Role Assignment

    Role = **AcrPull**

    Service Principal = KubernetesServicePrincipal

    Assign.

OR

1. Get the ACR Id

az acr show --resource-group DemoKRG --name dssregistry --query "id" --output tsv

output:

/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry

2. Assign contributor Role to Service Principal in Azure AD

az role assignment create --assignee "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --scope "/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry" **--role AcrPull**

Step3: Create Azure Kubernetes (AKS) Cluster

14. Create **Azure Kubernetes Service** with associated **Service Principal**

CLI:

az aks create --resource-group DemoKRG --name **dssdemocluster** --node-count 1 --generate-ssh-keys --service-principal "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --client-secret "ffb12981-f5f1-44d4-8fa1-e3b0aa609936"

OR

PowerShell:

$user = "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4"

$password = ConvertTo-SecureString -String "ffb12981-f5f1-44d4-8fa1-e3b0aa609936" -AsPlainText -Force

$Credential = New-Object -TypeName System.Management.Automation.PSCredential -ArgumentList $user, $password

New-AzAksCluster -ResourceGroupName DemoKRG -Name dssdemocluster -NodeCount 2 -GenerateSshKey -ServicePrincipalIdAndSecret $credential

Note: Delete key related files in this folder if existing: C:\Users\<Current User>\.ssh

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

15. To connect to Kubernetes Cluster from local machine

az aks get-credentials --resource-group DemoKRG --name **dssdemocluster**

16. **Configure Kubernetes to use your ACR** When creating deployments, replicasets or pods, kubernetes will try to use docker images already stored locally or pull them from the public docker hub. To change this, you need to specify the custom docker registry as part of your k8s object configuration (yaml or json).

Instead of specifying this directly in your configuration, we'll use the concept of k8s **secrets**. You decouple the k8s object from the registry configuration by just referencing the secret by it's name. But first, let's create a new k8s secret.

Note: It's optional for Azure Container Registry because the Service Principal used while creating the AKS cluster is already assigned AcrPull permission over the Azure Container Registry. But if Docker Hub Private image is used then it's mandatory to use secret for pulling image.

kubectl create secret docker-registry **kubernetes-principle-secret** --docker-server dssregistry**.azurecr.io** --docker-username="<Service Principal CLIENTID>" --docker-password "<Service Pricipal Secret>"

Kubernetes manifest files define a desired state for a cluster, including what container images should be running.

17. Create a file DeploymentAndService.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hellowebapp-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: hellowebapp
  template:
    metadata:
      labels:
        app: hellowebapp
    spec:
      containers:
      - name: hellowebapp
        image: dssregistry2022.azurecr.io/hellowebapp:v1
        imagePullPolicy: Always
        ports:
        - containerPort: 80
        - containerPort: 443
      imagePullSecrets:
      - name: kubernetes-principle-secret

---
```

```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

18. Use the kubectl create command to run the application.

```
kubectl apply -f deployment.yaml
```

Test the application

19. To monitor progress, use the the below command.

    kubectl get service azure-vote-front –watch

    Once the *EXTERNAL-IP* address has changed from *pending* to an *IP address*, use CTRL-C to stop the kubectl watch process.

20. Open a web browser to the **external IP address** of your service.

Publish to AKS Cluster using Kubernetes Manifest Task

Step 1: Add all the Kubernetes Deployments to the folder **HelloWorldApp.AKSYamls**

Eg:

HelloWorldApp.Deployments\Deployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: helloworldapp
  labels:
    app: helloworldapp
spec:
  replicas: 1
  selector:
```

```
    matchLabels:
      app: helloworldapp
  template:
    metadata:
      labels:
        app: helloworldapp
    spec:
      containers:
      - name: helloworldapp
        image: helloworldapp #This is overridden by pipeline and should not have tag name mentioned.
        imagePullPolicy: Always
        ports:
        - containerPort: 80
```

HelloWorldApp.Deployments\Service.yaml

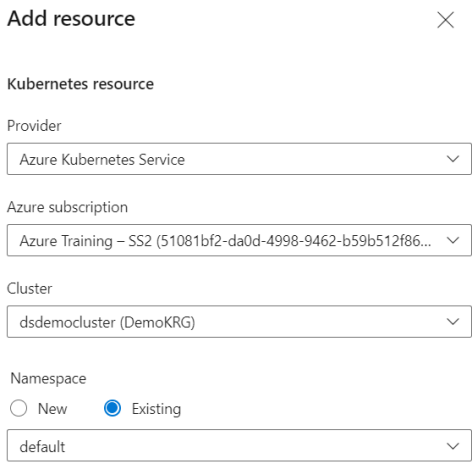
```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

Step 2: Create the Service Connections

1. Azure Container Registery (Name=DssAzureContainerRegistry)
    a. DevOps Project → Project Properties → Service Connection → New service connection → search Docker Registry → Next

Step 3: Create Environment

1. Pipelines → Environment → New Environment →
2. Environment Name = HelloWorldApp, Select Kubernetes → Create
3. Select Environment = **HelloWorldApp** → Add resource → Provider = Azure Kubernetes Service, Select Subscription and AKS Cluster and Namespace = default → Create



Step 4: Create a New YAML Pipeline as below

```
trigger:
- none

variables:
 # Container registry service connection established during pipeline creation
 dockerRegistryServiceConnection: 'DssAzureContainerRegistry'
 imageRepository: 'helloworldapp'
 containerRegistry: 'dssdemoregistry.azurecr.io'
 dockerfilePath: '**/Dockerfile'
 tag: '$(Build.BuildId)'
 imagePullSecret: 'dssregistry4568950f942-auth'

 # Agent VM image name
 vmImageName: 'ubuntu-latest'

stages:
- stage: Build
 displayName: Build stage
 jobs:
 - job: Build
  displayName: Build
  pool:
   vmImage: $(vmImageName)
  steps:
  - task: Docker@2
```

```
    displayName: Build and push an image to container registry
    inputs:
      command: buildAndPush
      repository: $(imageRepository)
      dockerfile: $(dockerfilePath)
      buildContext: '.'
      containerRegistry: $(dockerRegistryServiceConnection)
      tags: |
        $(tag)

  - task: CopyFiles@2
    inputs:
      SourceFolder: 'HelloWorldApp.AKSYamls'
      Contents: '**'
      TargetFolder: '$(Build.ArtifactStagingDirectory)'

  - task: PublishPipelineArtifact@1
    inputs:
      targetPath: '$(Build.ArtifactStagingDirectory)'
      artifact: 'drop'
      publishLocation: 'pipeline'

  # - upload: manifests
  #   artifact: manifests

- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: Deploy
    pool:
      vmImage: $(vmImageName)
    environment: 'HelloWorldApp.default'
    strategy:
      runOnce:
```

```
deploy:
 steps:
 - task: KubernetesManifest@0
  displayName: Create imagePullSecret
  inputs:
   action: createSecret
   secretName: $(imagePullSecret)
   dockerRegistryEndpoint: $(dockerRegistryServiceConnection)

 - task: KubernetesManifest@0
  displayName: Deploy to Kubernetes cluster
  inputs:
   action: deploy
   manifests: |
    $(Pipeline.Workspace)/drop/Deployment.yaml
    $(Pipeline.Workspace)/drop/Service.yaml
   imagePullSecrets: |
    $(imagePullSecret)
   containers: |
    $(containerRegistry)/$(imageRepository):$(tag)
```

Step 5: Test the Deployment

1. **Go to Kubernetes Cluster → Services and ingresses → Click on External IP of the Service**

Name	Namespace	Status	Type	Cluster IP	External IP	Ports
kubernetes	default	Ok	ClusterIP	10.0.0.1		443/TCP
kube-dns	kube-system	Ok	ClusterIP	10.0.0.10		53/UDP,5
metrics-server	kube-system	Ok	ClusterIP	10.0.43.73		443/TCP
hellowebapp	default	Ok	LoadBalancer	10.0.221.211	20.121.252.205	80:31500/

Publish to Azure Kubernetes Service using Kubernetes Task

1. Add the YML file to manifests folder of the Azure Repository

manifests/Deployment.yaml

Edit the deployment.yaml file update the containers section as below

```
apiVersion: apps/v1
kind: Deployment
```

```
metadata:
  name: helloworldapp
  labels:
    app: helloworldapp
spec:
  replicas: 1
  selector:
     matchLabels:
       app: helloworldapp
  template:
    metadata:
      labels:
        app: helloworldapp
    spec:
      containers:
      - name: helloworldapp
        image: #{ImageName}#
        imagePullPolicy: Always
        ports:
        - containerPort: 80
      imagePullSecrets:
      - name: kubernetes-principle-secret
```

Manifests/Service.yml

```
apiVersion: v1
kind: Service
metadata:
  name: hellowebapp
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: hellowebapp
```

2. Create following Service Connections

a. Azure Subscription (Name = AzureServiceConnection-SS2)

i. DevOps Project → Project Properties → Service Connection → New service connection → Azure Resouce Manager → Next

b. Azure Container Registery (Name=DssAzureContainerRegistry)

i. DevOps Project → Project Properties → Service Connection → New service connection → search Docker Registry → Next

c. Azure Kubernetes Cluster (Name=KubernetesServiceConnection)

i. DevOps Project → Project Properties → Service Connection → New service connection → search Kubernetes → Next

ii. Authentication Method = Azure Subscription

iii. Name=default

3. Create a YAML Azure DevOps Pipeline

```
trigger:
- none

variables:
 vmImageName: 'ubuntu-latest'
 azureResourceGroup: "Demo-rg"
 azureSubscriptionEndPoint: 'Azure - SS1'
 dockerRegistryServiceConnection: 'DockerRegistryConnection'
 kubernetesServiceConnection: 'KubernetesServiceConnection'
 dockerfilePath: 'HelloWorldApp.Web/Dockerfile'
 containerRegistry: dssdemoregistry.azurecr.io
 imageRepository: 'helloworldapp'
 tag: $(Build.BuildId)
 secretName: 'kubernetes-principle-secret'
 kubernetesCluster: 'dssdemocluster'
 ImageName: '$(containerRegistry)/$(imageRepository):$(tag)'

stages:
- stage: Build
 displayName: Build and push stage
 jobs:
 - job: Build
```

```
displayName: Build
pool:
 vmImage: $(vmImageName)
steps:
- task: Docker@2
 displayName: Build and push an image to container registry
 inputs:
  containerRegistry: '$(dockerRegistryServiceConnection)'
  repository: '$(imageRepository)'
  command: 'buildAndPush'
  Dockerfile: '$(dockerfilePath)'
  buildContext: '.'
  tags: '$(tag)'

- task: replacetokens@4
 inputs:
  targetFiles: '**/deployment.yml'
  encoding: 'auto'
  tokenPattern: 'default'
  writeBOM: true
  actionOnMissing: 'warn'
  keepToken: false
  actionOnNoFiles: 'continue'
  enableTransforms: false
  useLegacyPattern: false
  enableTelemetry: true

- task: replacetokens@3
 inputs:
  targetFiles: '**/deployment.yaml'
  encoding: 'auto'
  writeBOM: true
  actionOnMissing: 'fail'
  keepToken: false
  tokenPrefix: '~~'
  tokenSuffix: '~~'
```

```
      useLegacyPattern: false
      enableTransforms: false
      enableTelemetry: true

  - task: CopyFiles@2
    inputs:
      SourceFolder: 'manifests'
      Contents: '*.yml'
      TargetFolder: '$(build.artifactstagingdirectory)'

  - task: PublishPipelineArtifact@1
    inputs:
      targetPath: '$(build.artifactstagingdirectory)'
      artifact: 'drop'
      publishLocation: 'pipeline'

- stage: Deploy
  displayName: Deploy stage
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: Deploy job
    pool:
      vmImage: $(vmImageName)
    environment: 'deccansoft'
    strategy:
      runOnce:
        deploy:
          steps:
          - task: DownloadPipelineArtifact@2
            inputs:
              artifactName: 'drop'
              downloadPath: '$(System.ArtifactsDirectory)'

          - task: Kubernetes@1
            displayName: kubectl apply
```

```
    inputs:
      connectionType: 'Azure Resource Manager'
      azureSubscriptionEndpoint: $(azureSubscriptionEndPoint)
      azureResourceGroup: $(azureResourceGroup)
      kubernetesCluster: $(kubernetesCluster)
      command: 'apply'
      useConfigurationFile: true
      configuration: '$(System.ArtifactsDirectory)/deployment.yml'
      secretType: 'dockerRegistry'
      containerRegistryType: 'Azure Container Registry'
      azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndPoint)
      azureContainerRegistry: $(containerRegistry)
      secretName: $(secretName)

    - task: Kubernetes@1
      displayName: kubectl apply
      inputs:
        connectionType: 'Azure Resource Manager'
        azureSubscriptionEndpoint: $(azureSubscriptionEndPoint)
        azureResourceGroup: $(azureResourceGroup)
        kubernetesCluster: $(kubernetesCluster)
        command: 'apply'
        useConfigurationFile: true
        configuration: '$(System.ArtifactsDirectory)/service.yml'
        secretType: 'dockerRegistry'
        containerRegistryType: 'Azure Container Registry'
        azureSubscriptionEndpointForSecrets: $(azureSubscriptionEndPoint)
        azureContainerRegistry: $(containerRegistry)
        secretName: $(secretName)
```

Note: YAML File names are case-sentive.

Create a Service Connection using Service Account for ANY Kubernetes Cluster

DevOps Project → Project Properties → Service Connection → New service connection → search Kubernetes → Next

Account Type = Service Account

1. Connect to the Kubernetes Cluster

   a. az aks get-credentials --resource-group DemoKRG --name dssaprdemocluster1

2. Server URL = Output of
   a. kubectl config view --minify -o jsonpath={.clusters[0].cluster.server}
3. Secret = Output of
   a. kubectl get serviceAccounts default -n default -o=jsonpath={.secrets[*].name}
   b. kubectl get secret <output from step3> -n default -o json
4. Service Connection Name = "Kubernetes Connection"
5. Save

Step 3 : Create a ClusterRoleBinding which gives the role **cluster-admin** to the ServiceAccount **default in default namespace (default.default)**.

```
kubectl create clusterrolebinding serviceaccounts-cluster-admin -n kube-system --clusterrole=cluster-admin --serviceaccount=default:default
```

Deploying a multi-container application to Azure Kubernetes Services

https://azuredevopslabs.com/labs/vstsextend/kubernetes/

Create an Image Manually

Step 1: Update the Code

1. Create a New Project – HelloWebApp.Web

   dotnet new mvc -n HelloWebApp.Web

2. Add dockerfile as below Dockerfile

```
FROM mcr.microsoft.com/dotnet/sdk:5.0 AS build
WORKDIR /src
# copy csproj and restore as distinct layers
COPY *.csproj ./
RUN dotnet restore
# copy everything else and build app
COPY . ./
RUN dotnet build -c Release
RUN dotnet publish -c Release -o /out

FROM mcr.microsoft.com/dotnet/aspnet:5.0 AS runtime
WORKDIR /app
COPY --from=build /out ./
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build images

   docker build -t sandeepsoni/hellowebapp:v1 .

4. Run the docker image locally

   docker run --rm -p 8080:80 sandeepsoni/hellowebapp:v1

Azure Pipeline for Build and Publish Docker Image to Docker Hub

Azure Pipelines can be used to build images for any repository containing a Dockerfile. Building of both Linux and Windows containers is possible based on the agent platform used for the build.

Create a New DevOps Project and Commit the Code.

Create Container Registry Using Portal

1. **Create a resource → Containers → Azure Container Registry.**
2. Under **Admin user**, select **Enable**. Take note of the following values:
   - Login server
   - Username

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- password

Create a New Service Connection

Organization Properties → Service Connection → **New Service Connection** → **Docker Registry** → Select **Azure Container Registry**

Create a New Pipeline

1. Project → Pipelines → New Pipeline → Azure Repos → Docker (Build and push an image to Azure container registry)

Azure-pipeline.yml

```
trigger:
- master

resources:
- repo: self

variables:
 # Container registry service connection established during pipeline creation
 dockerRegistryServiceConnection: '487ebce8-02ce-48f8-b549-60b2b3aa6e12'
 imageRepository: 'helloworldapp.web'
 containerRegistry: 'dssdemo.azurecr.io'
 dockerfilePath: '$(Build.SourcesDirectory)/HelloWorldApp.Web/dockerfile'
 tag: '$(Build.BuildId)'

 # Agent VM image name
 vmImageName: 'ubuntu-latest'

stages:
- stage: Build
 displayName: Build and push stage
 jobs:
 - job: Build
  displayName: Build
  pool:
   vmImage: $(vmImageName)
  steps:
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
- task: Docker@2
  displayName: Build and push an image to container registry
  inputs:
    command: buildAndPush
    repository: $(imageRepository)
    dockerfile: $(dockerfilePath)
    buildContext: .
    containerRegistry: $(dockerRegistryServiceConnection)
    tags: |
      $(tag)
```

Deploying to Web App

You can automatically deploy your application to an Azure Web App for Linux Containers after every successful build.

You must supply an Azure service connection to the AzureWebAppContainer task. Add the following YAML snippet to your existing **azure-pipelines.yaml** file. Make sure you add the service connection details in the variables section as shown below-

```
- task: AzureRmWebAppDeployment@4
  displayName: 'Azure App Service Deploy: dssdemoapp123-con'
  inputs:
    azureSubscription: 'Azure Connection - SS2'
    appType: webAppContainer
    WebAppName: 'dssdemoapp123-con'
    DockerNamespace: dssdemo123.azurecr.io
    DockerRepository: helloworldapp
    DockerImageTag: '$(Build.BuildId)'
    AppSettings: '-DOCKER_REGISTRY_SERVER_URL dssdemo123.azurecr.io -
DOCKER_REGISTRY_SERVER_USERNAME dssdemo123 -DOCKER_REGISTRY_SERVER_PASSWORD
B7Vo1f6hBxNdUA8sWwcScBdcLTsMEE7='
```

Very Important:

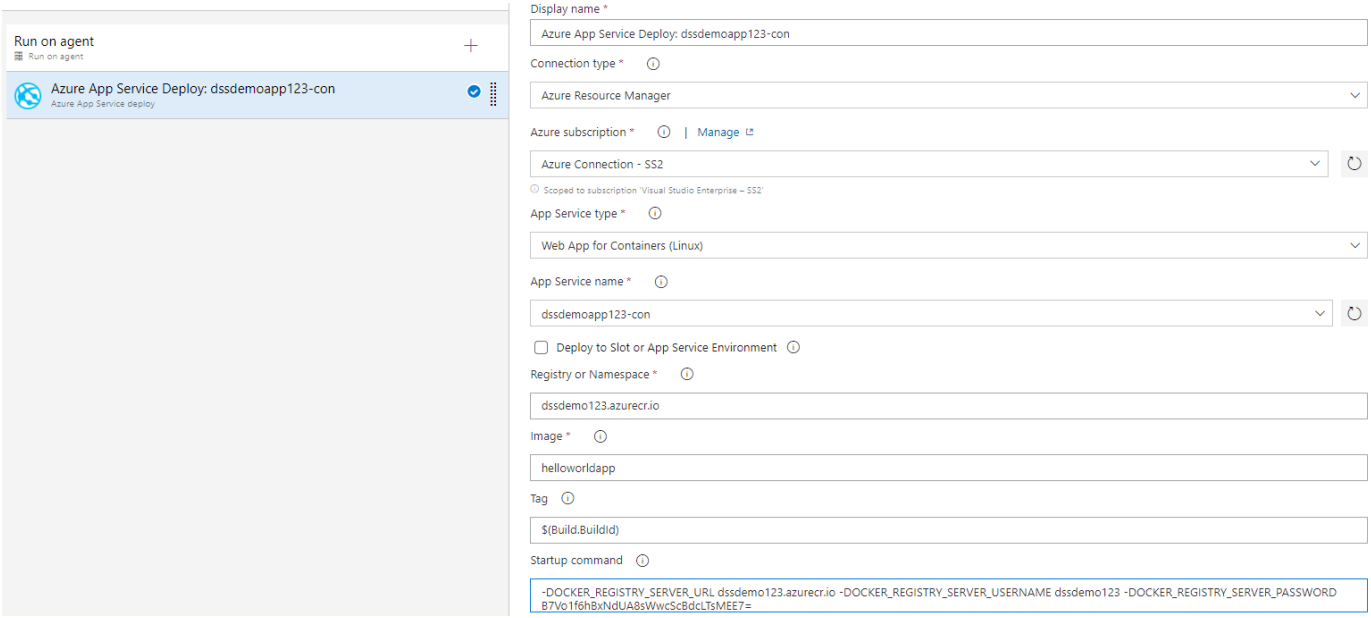
The above task is only setting the value of AppService - AppSettings key: DOCKER_CUSTOM_IMAGE_NAME = <DockerNamespace>/<DockerRepository>:DockerImageTag

The Appservice is then responsible for pulling the image from the Registry using the U/P as mentioned in its following appsettings:

- **DOCKER_REGISTRY_SERVER_URL**
- **DOCKER_REGISTRY_SERVER_USERNAME**

- **DOCKER_REGISTRY_SERVER_PASSWORD**

Note: Azure Pipe doesn't push image to App Service, it only sets the properties…



App Service Settings for Docker:

```
[
  {
    "name": "DOCKER_CUSTOM_IMAGE_NAME",
    "value": "dssdemo123.azurecr.io/helloworldapp:150",
    "slotSetting": false
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_PASSWORD",
    "value": "",
    "slotSetting": false
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_URL",
    "value": "https://mcr.microsoft.com",
    "slotSetting": false
  },
  {
    "name": "DOCKER_REGISTRY_SERVER_USERNAME",
    "value": "",
    "slotSetting": false
  }
]
```

Running in Any VM or Local Machine using Docker Compose

- A command-line tool and YAML file format with metadata for defining and running multi-container applications.
- You define a single application based on multiple images with one or more .yml files that can override values depending on the environment.
- After you have created the definitions, you can deploy the entire multi-container application by using a single command (docker-compose up) that creates a container per image on the Docker host.

1. Create a File: docker-compose.yml

```
version: '3.4'
services:
  hellowebapp:
    image: sandeepsoni/helloworldapp
    ports:
    - '8080:80'
```

2. Execute the following command to execute the application

```
docker-compose up
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

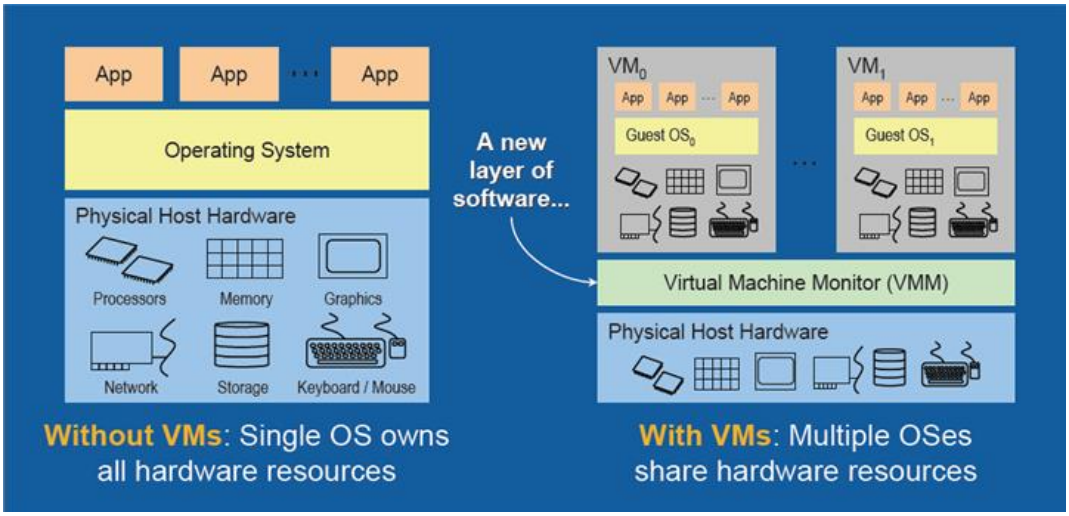
Agenda: Introduction to Containers and Docker

- Understanding VM's and Containers.
- What is Docker?
- Docker Benefits.
- Docker Architecture and Docker Taxonomy.
- The underlying technology

Understanding Virtual Machines and Containers

What is a Virtual Machine

- A virtual machine is a computer file, typically called an image, which behaves like an actual computer – computer within a computer
- It runs in a window, much like any other programme, giving the end user the same experience on a virtual machine as they would have on the host operating system itself.
- Multiple virtual machines can run simultaneously on the same physical computer.
- Hypervisor is the software required for managing VM, emulates the PC or server's CPU, memory, hard disk, network and other hardware resources completely, enabling virtual machines to share the resources.
- The hypervisor (Hyper-v, VMWare, Xen, KVM) can emulate multiple virtual hardware platforms that are isolated from each other, allowing virtual machines to run Linux and Windows Server operating systems on the same underlying physical host.



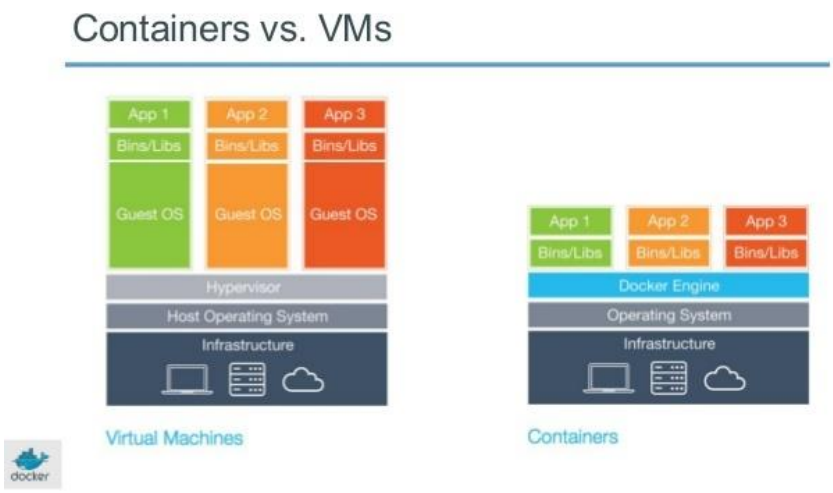
What is a Container?

- Containerization is an **approach** to software development in which **an application or service, its dependencies, and its configuration** are packaged together as a **container image**. You then can test the containerized application **as a unit** and deploy it as a container image instance to the host operating system.

- Placing software into containers makes it possible for developers and IT professionals to deploy those containers **across environments** with little or no modification.
- Containers also isolate applications from one another on a **shared operating system (OS**). Containerized applications run on top of a **container host**, which in turn runs on the OS (Linux or Windows). Thus, containers have a significantly smaller footprint than virtual machine (VM) images.
- Con, agility, scalability, and control across the entire application life cycle workflow. The most important benefit is the isolation provided between Dev and Ops.

Containers vs VMs

Containers are lightweight because they don't need the extra load of a hypervisor, but run directly within the host machine's kernel. This means you can run more containers on a given hardware combination than if you were using virtual machines. You can even run Docker containers within host machines that are actually virtual machines!



What is Docker and its Benefits

The **Docker platform** uses the **Docker Engine** to quickly build and package apps as **Docker images** created using files written in the **Dockerfile** format that then is deployed and run in a layered **container**.

Benefits of Docker / Container:

1. **Resource Efficiency**: Docker is lightweight and fast. Process level isolation and usage of the container host's kernel is more efficient when compared to virtualizing an entire hardware server using VM.
2. **Fast and consistent delivery of your applications:** By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

3. **Continuous Deployment and Testing**: The ability to have consistent environments and flexibility with patching has made Docker a great choice for teams that want to move from waterfall to the modern DevOps approach to software delivery.

Consider the following example scenario:

a) Your developers write code locally and share their work with their colleagues using Docker containers.
b) They use Docker to push their applications into a test environment and execute automated and manual tests.
c) When developers find bugs, they can fix them in the development environment and redeploy them to the test environment for testing and validation.
d) When testing is complete, getting the fix to the customer is as simple as pushing the updated image to the production environment.

Docker is available for implementation across a wide range of platforms:

- **Desktop**: Mac OS, Windows 10.
- **Server**: Various Linux distributions and Windows Server 2019.
- **Cloud**: Amazon Web Services, Google Compute Platform, Microsoft Azure, IBM Cloud, and more.

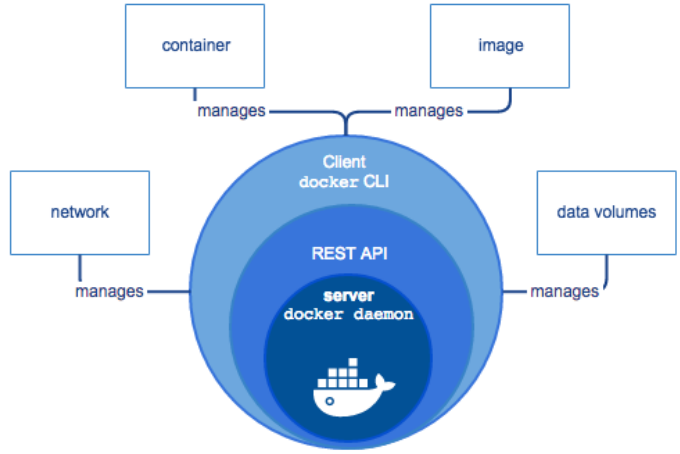
Alternatives to Docker

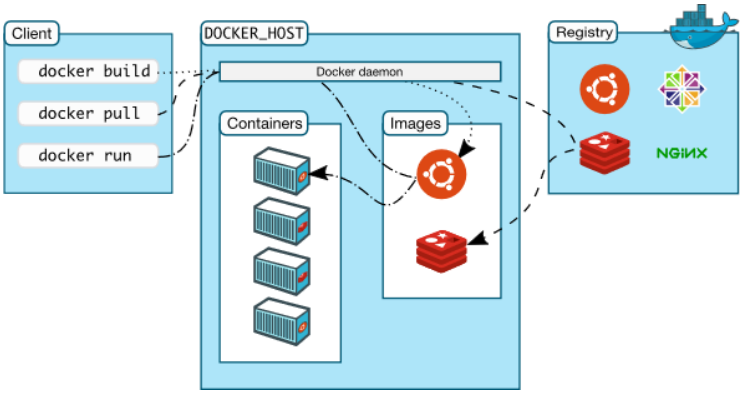
1. Open Container Initiative (OCI)
2. CoreOS and rkt
3. Apache Mesos and Mesosphere
4. Canonical and LXD

Docker Architecture and its Taxonomy

Docker Engine is a client-server application with these major components:

- A server which is a type of long-running program called a **daemon** process.
- A REST API which specifies interfaces that programs can use to talk to the daemon and instruct it what to do.
- A command line interface (CLI) client (the docker command).



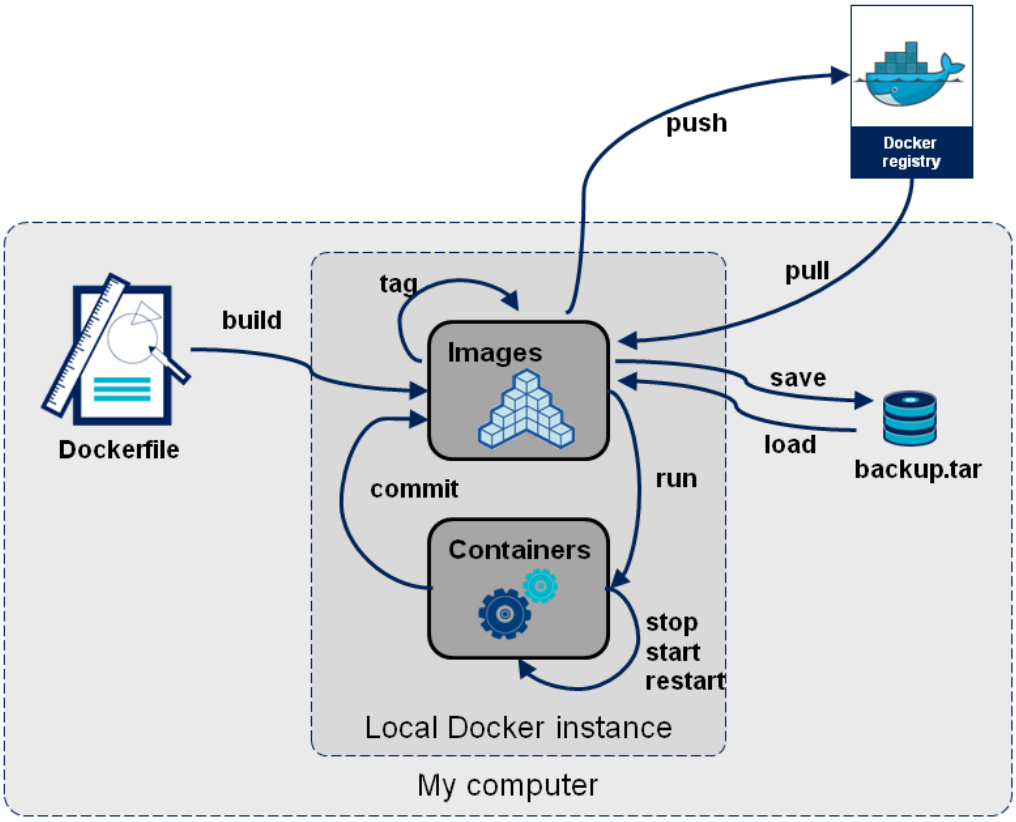


Docker Daemon:

- The Docker daemon (dockerd) listens for Docker API requests and manages Docker objects such as **images, containers, networks, and volumes**.
- A daemon can also communicate with other daemons to manage Docker services.

Docker client:

- The Docker client (docker) is the primary way that many Docker users interact with Docker.
- When you use commands such as ***docker run***, the client sends these commands to Docker Daemon, which carries them out.
- The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon.
- The Docker client and daemon communicate using a REST API, over sockets or a network interface.
- The Docker client can communicate with more than one daemon.



Docker Image:

- A **package with all of the dependencies** and information needed to create a container. An image includes all of the dependencies (such as frameworks) plus deployment and configuration to be used by a container runtime.
- Usually, an image derives from multiple base images that are **layers stacked** one atop the other to form the container's file system.
- An image is **immutable** after it has been created.
- Docker image containers can **run NATIVELY on Linux and Windows**.
    - Windows images can run only on Windows hosts.
    - Linux images can run only on Linux hosts, meaning a host server or a VM.
    - Developers who work on **Windows** can create images for **either Linux** or **Windows** Images.

Container:

- An instance of an image is called a container. Its an process by itself and each process has its own hostname(IP address) like a Virtual Machine.
- Its an isolated area of the OS with resource usage limits applied.
- The container or instance of a Docker image will contain the following components:

  - An operating system selection (for example, a Linux distribution or Windows)
  - Files added by the developer (for example, app binaries)
  - Configuration (for example, environment settings and dependencies)
  - Instructions for what processes to run by Docker
- A container represents a runtime for **a single application**, process, or service. It consists of the contents of a Docker image, a runtime environment, and a standard set of instructions.
- You can create, start, stop, move, or delete a container using the Docker API or CLI.
- Container makes it possible for each application to have its own networking stack, which means it can have its own IP address, file system and windows registry. In short its an isolated process.

Dockerfile:

- A text file that contains instructions for how to build a Docker image.

Build:

- The action of building a container image based on the information and context provided by its Dockerfile as well as additional files in the folder where the image is built.
- You can build images by using the Docker ***docker build*** command.

Networking:

- Docker implements networking in an application-driven manner and provides various options while maintaining enough abstraction for application developers.
- By default, you get three different networks on the installation of Docker - **none, bridge, and host**. The none and host networks are part of the network stack in Docker. The bridge network automatically creates a gateway and IP subnet and all containers that belong to this network can talk to each other via IP addressing.
- In addition to the built-in default Docker networks, administrators can configure multiple **user-defined networks**. The three types of these networks are **Bridge, Overlay and Macvlan Network.**

Storage Volume:

- Data Volumes provide the ability to create **persistent** storage, with the ability to rename volumes, list volumes, and also list the container that is associated with the volume.
- Data Volumes sit on the host file system, outside the containers copy on write mechanism and are fairly efficient.

Repository (also known as repo):

- A collection of related Docker images labeled with a tag that indicates the image version or OS or any other information.

- Some repositories contain multiple variants of a specific image, such as an image containing SDKs (heavier), an image containing only runtimes (lighter), and so on. Those variants can be marked with tags.
- A single repository can contain platform variants, such as a Linux image and a Windows image.

Tag:

- A mark or label that you can apply to images so that different images or versions of the same image (depending on the version number or the destination environment) can be identified.

Registry:

- A service that provides access to repositories.
- The default registry for most public images is Docker Hub (owned by Docker as an organization).
- A registry usually contains repositories from multiple teams.
- Companies often have private registries to store and manage images that they've created.
- *Azure Container Registry* or AWS Elastic Container Registry are other example.

Docker Trusted Registry (DTR):

- A Docker registry service (from Docker) that you can install on-premises so that it resides within the organization's datacenter and network. It is convenient for private images that should be managed within the enterprise. Docker Trusted Registry is included as part of the Docker Datacenter product. For more information, go to https://docs.docker.com/docker-trusted-registry/overview/.

Compose:

- A command-line tool and **YAML** file format with metadata for defining and running multi-container applications.
- You define a single application based on multiple images with one or more .yml files that can override values depending on the environment.
- After you have created the definitions, you can deploy the entire multi-container application by using a single command **(docker-compose up)** that creates a container per image on the Docker host.

Cluster:

- A collection of Docker hosts exposed as if they were a single virtual Docker host so that the application can scale to multiple instances of the services spread across multiple hosts within the cluster.
- You can create Docker clusters by using Docker Swarm, Mesosphere DC/OS, Kubernetes, and Azure Service Fabric. (If you use Docker Swarm for managing a cluster, you typically refer to the cluster as a *swarm* instead of a cluster.)

Orchestrator:

- A tool that simplifies management of clusters and Docker hosts.
- Using orchestrators, you can manage their images, containers, and hosts through a CLI or a graphical user interface.
- You can manage container networking, configurations, load balancing, service discovery, high availability, Docker host configuration, and more.
- An orchestrator is responsible for running, distributing, scaling, and healing workloads across a collection of nodes.
- Typically, orchestrator products are the same products that provide cluster infrastructure, like Mesosphere DC/OS, **Kubernetes**, Docker Swarm, and Azure Service Fabric.

Docker Community Edition (CE):

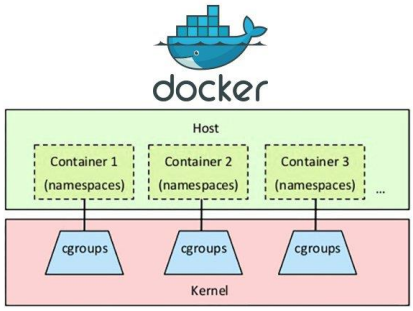
- Development tools for Windows and macOS for building, running, and testing containers locally.
- Docker CE for Windows provides development environments for both Linux and Windows Containers.
- The Linux Docker host on Windows is based on a Hyper-V VM. The host for Windows Containers is directly based on Windows.
- Docker CE for Mac is based on the Apple Hypervisor framework and the xhyve hypervisor, which provides a Linux Docker host VM on Mac OS X.
- Docker CE for Windows and for Mac replaces Docker Toolbox, which was based on Oracle VirtualBox.

Docker Enterprise Edition:

- It is designed for enterprise development and is used by IT teams who build, ship, and run large business-critical applications in production.

The Underlying technology

Docker is written in **Go** programming language and takes advantage of several features of Linux Kernel to deliver its functionality.



Namespaces

Docker uses a technology called namespaces to provide the **isolated workspace** called the container. When you run a container, Docker creates a **set of namespaces** for that container.

Docker uses namespaces of various kinds to provide the isolation that containers need in order to remain portable and refrain from affecting the remainder of the host system.

Each aspect of a container runs in a separate namespace and its access is limited to that namespace.

Docker Engine uses namespaces such as the following on Linux:

- The **pid** namespace: Process isolation (PID: Process ID).
- The **net** namespace: Managing network interfaces (NET: Networking).
- The **ipc** namespace: Managing access to IPC resources (IPC: InterProcess Communication).
- The **mnt** namespace: Managing filesystem mount points (MNT: Mount).
- The **uts** namespace: Isolating kernel and version identifiers. (UTS: Unix Timesharing System).

Control groups (cgroups).

A cgroup **limits** an application to a specific set of resources. Control groups allow Docker Engine to share available hardware resources to containers and optionally enforce **limits** and **constraints**. For example, you can limit the CPU, Memory, Network I/O or access to filesystem available to a specific container.

To lock a Docker container to the first CPU core: **docker run --cpuset-cpus=0**.

To limit container memory: **docker run –memory=**.

Common control groups

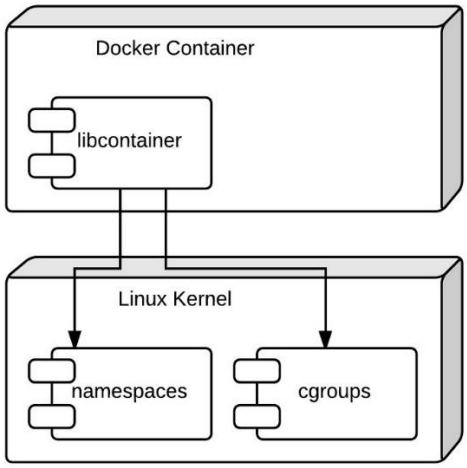
1. CPU
2. Memory
3. Network Bandwidth
4. Disk
5. Priority

Union file systems (UnionFS)

UnionFS, are file systems that operate by creating layers, making them very lightweight and fast. Docker Engine uses UnionFS to provide the building blocks for containers.

Container format

Docker Engine combines the **namespaces, control groups, and UnionFS** into a wrapper called a container format. The default container format is **libcontainer**. In the future, Docker may support other container formats by integrating with technologies such as BSD Jails or Solaris Zones.



Agenda: Azure Artifacts

- What are Artifacts
- Public and Download Build Artifacts
- Publish and Download Pipeline Artifacts
- Working with Feed and NuGet Packages
- Upstream Sources
- Public NuGet Package from Pipeline to NuGet Feeds
- Views
- Share Packages Publicly

What are Artifacts?

An artifact is a deployable component of your application. These components can then be deployed to one or more environments.

How do we get an artifact? There are different ways to create and retrieve artifacts, and not every method is appropriate for every situation.

1. Build Artifact
2. Azure Repos
3. GitHub
4. TFVC
5. Azure Artifacts
6. Azure Containers
7. Docker Hub
8. Jenkins

The most common and most used way to get an artifact within the release pipeline is to use a **build artifact**.

The build pipeline compiles, tests, and eventually produces an **immutable package**, which is stored in a secure place (storage account, database etc.).

The release pipeline then uses a secure connection to this **secured place to get the build artifact** and perform additional actions to deploy this to an environment.

Publish and Download Build Artifacts

- Artifacts are the files that you want your build to produce.
- Artifacts can be anything that your team needs to test or deploy your app.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

PublishBuildArtifacts@1: Use this task in a **build pipeline** to publish build artifacts to Azure Pipelines, TFS, or a file share.

Example: Publish a text file as an artifact to drop1 and drop2

```
- task: PublishBuildArtifacts@1
  inputs:
    pathtoPublish: '$(Build.ArtifactStagingDirectory)
    artifactName: drop

- task: PublishBuildArtifacts@1
  inputs:
    pathtoPublish: '$(Build.ArtifactStagingDirectory)/Release'
    artifactName: dropRelease

- task: PublishBuildArtifacts@1
  inputs:
    pathtoPublish: '$(Build.ArtifactStagingDirectory)/Debug'
    artifactName: dropDebug
```

DownloadBuildArtifacts@0

Use this task in a **build or release** pipeline to download build artifacts.

You can consume an artifact produced by one job in a subsequent job of the pipeline, even when that job is in a different stage (YAML pipelines). This can be useful to test your artifact.

```
- task: DownloadBuildArtifacts@0
  inputs:
    buildType: 'current'
    downloadType: 'single'
    artifactName: 'dropRelease'
    downloadPath: '$(System.ArtifactsDirectory)'
    itemPattern: 'dropRelease/*.txt'
```

Publish and Download Pipeline Artifacts

- Pipeline artifacts provide a way to share files **between stages in a pipeline or between different pipelines**.
- They are typically the output of a build process that need to be consumed by another job or be deployed.

- Artifacts are associated with the run they were produced in and remain available after the run has completed.

PublishPipelineArtifact@1: Use this task in a pipeline to publish artifacts for the Azure Pipeline (note that publishing is NOT supported in release pipelines. It is supported in build pipelines, multi stage YAML pipelines)

To publish (upload) an artifact for the current run of a CI/CD or classic pipeline:

```
- task: PublishPipelineArtifact@1
  inputs:
    path: $(System.DefaultWorkingDirectory)/bin/WebApp
    artifact: WebApp
```

Limiting which files are included

.artifactignore files use the identical file-globbing syntax of .gitignore to provide a version-controlled way to specify which files should *not* be added to a pipeline artifact.

Using an. **artifactignore** file, it is possible to omit the path from the task configuration, if you want to create a Pipeline Artifact containing everything in and under the working directory, minus all of the ignored files and folders. For example, to include only files in the artifact with a .exe extension:

```
**/*
!*.exe
```

To download a specific artifact in CI/CD or classic pipelines:

```
- task: DownloadPipelineArtifact@2
  inputs:
    artifact: WebApp
```

The **Download Pipeline Artifact** task can download both build artifacts (published with the PublishBuildArtifacts task) and pipeline artifacts

By default, files are downloaded to **$(Pipeline.Workspace)/{artifact}**, where artifact is the name of the artifact. The folder structure of the artifact is always preserved.

Artifact Selection

- A **single download** step can download one or more artifacts.
- To **download multiple** artifacts, do not specify an artifact name and optionally use file matching patterns to limit which artifacts and files are downloaded.
- The default file matching pattern is **, meaning all files in all artifacts.

Single artifact: *When an artifact name is specified:*

1. Only files for this artifact are downloaded. If this artifact does not exist, **the task will fail.**
2. Unless the specified download path is absolute, a folder with the same name as the artifact is created under the download path, and the artifact's files are placed in it.
3. File matching patterns are evaluated relative to the root of the artifact. For example, the pattern *.jar matches all files with a .jar extension at the root of the artifact.

For example, to download all *.js from the artifact WebApp are downloaded to $(Build.SourceDirectory)/bin

```
- task: DownloadPipelineArtifact@2
  inputs:
    artifact: WebApp
    patterns: '**/*.js'
    path: $(Build.SourcesDirectory)/bin
```

Multiple artifacts: *When no artifact name is specified:*

1. Files from **multiple artifacts can be downloaded**, and the task does not fail if no files are downloaded.
2. A folder is always created under the download path for each artifact with files being downloaded.
3. File matching patterns should assume the first segment of the pattern is (or matches) an artifact name. For example, WebApp/** matches all files from the WebApp artifact. The pattern */*.dll matches all files with a .dll extension at the root of each artifact.

For example, to download all .zip files from all source artifacts:

```
- task: DownloadPipelineArtifact@2
  inputs:
    patterns: '**/*.dll'
    path: $(Build.SourcesDirectory)/bin
```

Artifacts in release and deployment jobs

- If you're using pipeline artifacts to deliver artifacts into a **classic release pipeline or deployment job in YAML pipeline**, you do not need to add a download step --- **a step is injected automatically.**
- If you need to control over the location where files are downloaded, you can add a **Download Pipeline Artifact** task or use the **download** YAML keyword.

To **stop artifacts** from being downloaded automatically, add a download step and set its value to none:

```
steps:
- download: none
```

Migrating from Build Artifacts (PublishBuildArtifact) to Pipeline Artifacts(PublishPipelineArtifact)

Pipeline artifacts are the **next generation** of build artifacts and are the recommended way to work with artifacts. Artifacts published using the **Publish Build Artifacts** task can continue to be downloaded using **Download Build Artifacts**, but can also be downloaded using the latest **Download Pipeline Artifact** task.

When migrating from build artifacts to pipeline artifacts:

1. For build artifacts, it's common to copy files to $(Build.ArtifactStagingDirectory) and then use the Publish Build Artifacts task to publish this folder. With the Publish Pipeline Artifact task, just publish directly from the path containing the files.
2. By default, the Download Pipeline Artifact task downloads files to $(Pipeline.Workspace). This is the default and recommended path for all types of artifacts.
3. File matching patterns for the **Download Build Artifacts** task are expected to start **with (or match) the artifact name**, regardless if a specific artifact was specified or not. In the **Download Pipeline Artifact** task, patterns should **not include the artifact name** when an artifact name has already been specified.

What is a Package?

A package is a formalized way of creating a **distributable unit of software** artifacts that can be consumed from another software solution. The package describes the content it contains and usually provides **additional metadata**. This additional information uniquely identifies the individual packages and to be **self-descriptive**. It helps to better store packages in **centralized locations** and consume the contents of the package in a predictable manner. In addition, it **enables tooling** to manage the packages in the software solution.

Packaging Formats

- **NuGet:** A NuGet package is essentially a compressed folder structure with published **.NET** project files in ZIP format and has the **.nupkg** extension.
- **NPM**: A NPM package is a file or folder that contains **JavaScript** files and a **package.json** file describing the metadata of the package.
- **Maven**: Each **Java** based projects package has It has a Project Object Model file describing the metadata of the project
- **PyPI**: The Python Package Index, abbreviated as PyPI and also known as the Cheese Shop, is the official third-party software repository for Python.

Package feeds

- Packages should be stored in a centralized place for distribution and consumption by others to take dependencies on the components it contains. The centralized storage for packages is most commonly called a package feed.
- Each package type has its own type of feed.
- Package feeds offer versioned storage of packages. A certain package can exist in multiple versions in the feed, catering for consumption of a specific version.
- There are NuGet feeds, NPM feeds, Maven repositories, PyPi feed and Docker registries.
- The feeds can be exposed in public or private to distinguish in visibility. Public feeds can be consumed by anyone.

Public Feeds:

Package type	Package source	URL
NuGet	NuGet Gallery	**https://nuget.org**
NPM	NPMjs	**https://npmjs.org**
Maven	Maven	**https://search.maven.org**
Docker	Docker Hub	**https://hub.docker.com**
Python	Python Package Index	**https://pypi.org**

The table above does not contain an extensive list of all public sources available. There are other public package sources for each of the types.

Private Feeds:

There are two options for private feeds:

1. **Self hosted** = We have a create a Physical / VM and install the feed software in our on-premise env.
2. **SaaS Service** = Cloud based Service hosted and accessible through internet.

The following table contains a non-exhaustive list of self-hosting options and SaaS offerings to privately host package feeds for each of the types covered

Package type	Self-hosted private feed	SaaS private feed
NuGet	NuGet server	Azure Artifacts, MyGet
NPM	Sinopia, cnpmjs.org, Verdaccio	NPMjs.org, MyGet, Azure Artifacts
Maven	Nexus, Artifactory, Archivia	Azure Artifacts, Bintray, JitPack
Docker	Portus, Quay, Harbor	Docker Hub, Azure Container Registry, Amazon Elastic Container Registry
Python	PyPI Server	Gemfury

Working with Feed and NuGet Packages

A feed is a container for packages. You **consume and publish** packages through a particular feed.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Artifacts → Create Feed

In dialog box

- Give the feed a name.
- Choose who can read and contribute (or update) packages in your feed.
- Choose the upstream sources for your feed.
- When you're done, select **Create**.

Step 1: Create a Feed

A feed is a container for packages. You consume and publish packages through a particular feed.

1. Azure DevOps → Select DevOps Project → Azure Artifacts → **+ Create Feed** → Name="DotNetPackages" → Create

Step2: Create a NuGet Package (Local Developer Machine)

2. Create a .**NET Standard Classlibrary** Project.
3. Edit the Project file (*.csproj) as below

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <TargetFramework>netstandard2.0</TargetFramework>
  <PackageId>HelloWorldApp_Library</PackageId>
  <Version>1.0.0</Version>
  <Authors>Sandeep</Authors>
  <Company>Deccansoft</Company>
  <GeneratePackageOnBuild>true</GeneratePackageOnBuild>
 </PropertyGroup>
</Project>
```

4. Build the project to generate .\bin\Debug**HelloWorldApp_Library.1.0.0.nupkg** file.

Step 3: Publish a Package from the project folder to Feed.

5. Download and **install the Credential Provider** from https://github.com/microsoft/artifacts-credprovider#azure-artifacts-credential-provider → Click on PowerShell helper script → Convert to RAW format and Save
    a. Save the downloaded PowerShell Script as d:\installcredprovider.ps1
    b. Open Powershell Window in Administrator Mode
    c. D:\> **Set-ExecutionPolicy** -ExecutionPolicy RemoteSigned -Scope LocalMachine
    d. D:\> .**installcredprovider.ps1**

6. Add a new file to project: HelloWorldApp.ClassLibrary/**nuget.config**

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
 <packageSources>
  <clear />
  <add key="DotNetPackages" value="https://pkgs.dev.azure.com/DotNet Packages/_packaging/DotNet Packages/nuget/v3/index.json" />
 </packageSources>
</configuration>
```

7. Execute the following command from the same folder where we have saved **nuget.config**

   dotnet nuget push --interactive --source "DotNetPackages" --api-key az D:\Temp\ClassLibrary1\bin\Debug\HelloWorldApp_Library.1.0.0.nupkg

   Provide the required credentials

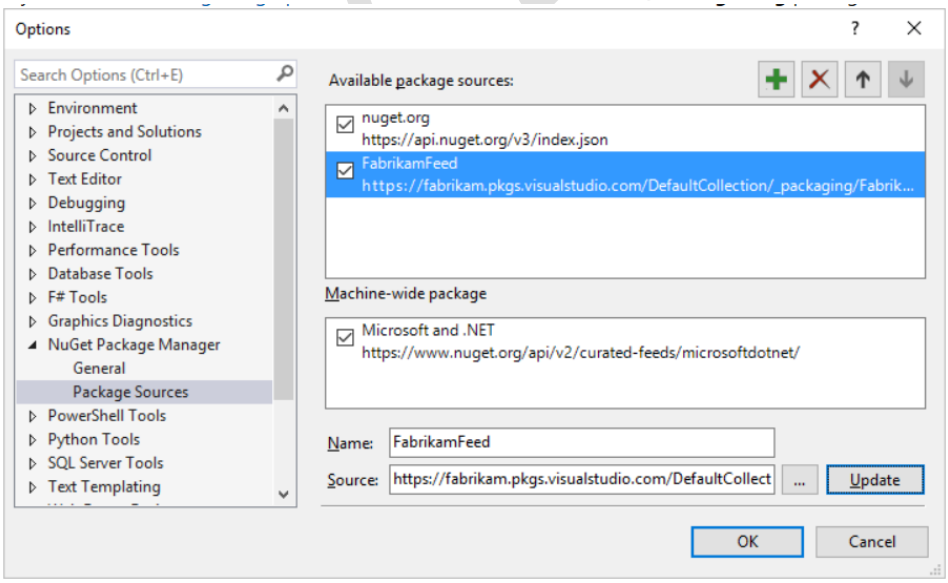
Result: Package is now successfully installed in the Feed.

Step 4: Consume your package in Visual Studio

To consume NuGet packages from a feed, add the feed's NuGet endpoint as a package source in Visual Studio.

8. Go to your feed → Select **Connect to feed**: → **Visual Studio** tab

9. Visual Studio → Tools → Options → Expand NuGet Package Manager → Package Sources. Select the green plus in the upper-right corner and enter the name and source URL → OK

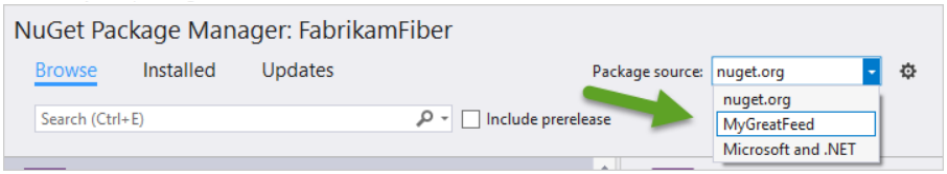


Step 5: Consume the package

1. Any Visual Studio Project → Right-click **References** →Select **Manage NuGet Packages** → In the **Package source** drop-down list, select your feed.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

2. Look for your package in the list.



Secure and share packages using feed permissions

By default, the Project Collection Build Service is a Contributor and your project team is a Reader.

Permission	Reader	Collaborator	Contributor	Owner
List and restore/install packages	✓	✓	✓	✓
Save packages from upstream sources		✓	✓	✓
Push packages			✓	✓
Unlist/deprecate packages			✓	✓
Delete/unpublish package				✓
Edit feed permissions				✓
Rename and delete feed				✓

Editing permissions for a feed

With your feed selected, select **Edit feed** (the gear icon) → Select **Permissions** tab

In the edit feed dialog:

- Choose to make each person or team an Owner, Contributor, Collaborator, or Reader.
- When you're done, select **Save**.

Upstream Sources

Upstream sources:

- When combining private and public sources, the order of resolution of the sources becomes important.

- One way to specify multiple packages sources is by choosing a primary source and specifying an upstream source. The package manager will evaluate the primary source first and switch to the upstream source when the package is not found there.
- The upstream source might be one of the official public sources or a private source. The upstream source could refer to another upstream source itself, creating a chain of sources.
- A typical scenario is to use a private package source referring to an public upstream source for one of the official feeds. This effectively enhances the packages in the upstream source with packages from the private feed, avoiding the need to publish private packages in a public feed.
- A source that has an upstream source defined may download and **cache the packages** that were requested it does not contain itself.
- Upstream sources enable you to use a **single feed** to store both the packages you **produce** and the packages you **consume** from "remote feeds": both public feeds (e.g. npmjs.com, nuget.org, Maven Central, and PyPI) and authenticated feeds (i.e. other Azure DevOps Services feeds in your organization or in organizations in your Azure Active Directory (AAD) tenant).
- Once you've enabled an upstream source, any user connected to your feed can install a package from the remote feed, **and your feed will save a copy**.

Use a single feed on the client

In order for your feed to provide deterministic restore, it's important to ensure that your package feed configuration file—your .npmrc or nuget.config—references only your Azure Artifacts feed with upstream sources enabled. For NuGet, the <packageSources> section should look like:

Step1: Steps to Create a feed with upstream sources enabled

1. Azure Artifacts → + Create feed → Name="DeccansoftFeed"… **Check Include packages from common public sources** → Create

Step2: Replace the public registry in configuration files with the Azure Artifacts feed

1. Azure Artifacts → select Feed → Select NuGet.exe Tab → Copy the XML snippet under Project Setup
2. In VS, create a new file named **nuget.config** in the root of your project → Paste the XML snippet.

```
<packageSources>
 <clear />
 <add key="DeccansoftFeed"
value="https://pkgs.dev.azure.com/demoorg/_packaging/DeccansoftFeed/nuget/v3/index.json" />
</packageSources>
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

3. Clear your local package cache:

```
nuget locals -clear all
```

or

Visual Studio → Tools → NuGet Package Manager → General → Click **Clear All NuGet Cache(s)**

Note: If you're using upstream sources, package versions in the upstream source that haven't yet been saved into your feed (by using them at least once) won't appear in the NuGet Package Manager search.

4. To install the packages and get it listed in our Custom Feed of DevOps Project
    a. Go to **https://www.nuget.org (one of the upstream sources)**
    b. Search the required Package and copy the **Install-Package** command.
        Example: **Install-Package Newtonsoft.Json -Version 12.0.3**
    c. In Visual Studio, open the Package Manager Console from **Tools → NuGet Package Manager**.
    d. Paste and run the command: **Install-Package Newtonsoft.Json -Version 12.0.3**.
5. Download and install packages from the upstream sources:

```
nuget restore
dotnet restore
```

6. Check your feed to see the saved copy of everything you used from the public registry
7. Now we can add Newtonsoft package reference to our project from the Custom Feed instead of Nuget.org

Views

- Views enable you to share **subsets** of the NuGet, npm, Maven, and Python package-versions in your feed with consumers.
- A common use for views is to share package versions that have been tested, validated, or deployed but hold back packages still under development and packages that didn't meet a quality bar.
- All Azure DevOps Services feeds come with 3 views**: @local, @prerelease, and @release**. The latter two are suggested views that you can rename or delete as desired. The @local view is a special view that's commonly used in upstream sources.
- Views are read-only, which means that users connected to a view can only use packages that are published to the feed and packages previously saved from upstream sources by users connected to the feed.

1. Go to Feed → Settings (Gear Icon on right) → Views Tab
2. **Edit a View**
    a. You can edit the name of Prerelease and Release but not Local
    b. You can restrict visibility to either all people in organization or specific people

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

3. **Promoting to a release view**
    a. Select the package → Click on . . . →
    b. select **Promote**
4. **Consuming from a release view**
    a. Select **Connect to feed**
    b. Use the feed URL with View name

https://pkgs.dev.azure.com/DevOpsDemoInJan2020/HelloWorld/_packaging/MyDemoFeed@release/nuget/v3/index.json

Publish NuGet packages from your build to NuGet feeds

You can publish NuGet packages from your build to NuGet feeds. You can publish these packages to:

- Azure Artifacts or the TFS Package Management service.
- Other NuGet services such as NuGet.org.
- Your internal NuGet repository.

Step1: To create a package, add the following snippet to your azure-pipelines.yml file.

```
variables:
  Major: '1'
  Minor: '0'
  Patch: '0'

steps:
- task: NuGetCommand@2
  inputs:
    command: pack
    versioningScheme: byPrereleaseNumber
    majorVersion: '$(Major)'
    minorVersion: '$(Minor)'
    patchVersion: '$(Patch)'
OR
- task: DotNetCoreCLI@2
  displayName: 'dotnet pack'
  inputs:
    command: pack
    versioningScheme: byBuildNumber
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Step2: Publish Your Packages to Azure Artifacts feed

1. To publish to an Azure Artifacts feed, set the **Project Collection Build Service** identity to be a **Contributor** on the feed
2. Add the following snippet to your azure-pipelines.yml file.

```
steps:
- task: NuGetCommand@2
  displayName: 'NuGet push'
  inputs:
    command: push
    publishVstsFeed: '<feedName>'
    allowPackageConflicts: true
OR
- task: DotNetCoreCLI@2
  displayName: 'dotnet push'
  inputs:
    command: push
    publishVstsFeed: '<feedName>'
```

Step3: To publish to an external NuGet feed:

1. create a service connection to point to that feed. **Project settings**, → **Service connections**, → **New service connection**. → **NuGet** option for the service connection.
2. To connect to the feed, fill in the feed URL and the API key or token.
3. Add the following snippet to your azure-pipelines.yml file.

```
- task: NuGetAuthenticate@0
  inputs:
    nuGetServiceConnections: '<Name of the NuGet service connection>'

- task: NuGetCommand@2
  inputs:
    command: push
    nuGetFeedType: external
    versioningScheme: byEnvVar
    versionEnvVar: <VersionVariableName>
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Universal Packages

Share code easily by storing Apache Maven, npm, and NuGet packages together. You can store packages using Universal Packages, eliminating the need to store binaries in Git.

Universal Packages store one or more files together in a single unit that has a name and version. You can publish Universal Packages from the command line by using the Azure CLI.

Log in to Azure DevOps

```
az login
az devops configure --defaults organization=https://dev.azure.com/[your-organization]
project=ContosoWebApp
```

Publish a Universal Package

```
az artifacts universal publish --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --description "Your description" --path .
```

Download a Universal Package

```
az artifacts universal download --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --path .
```

Filtered Universal Package downloads

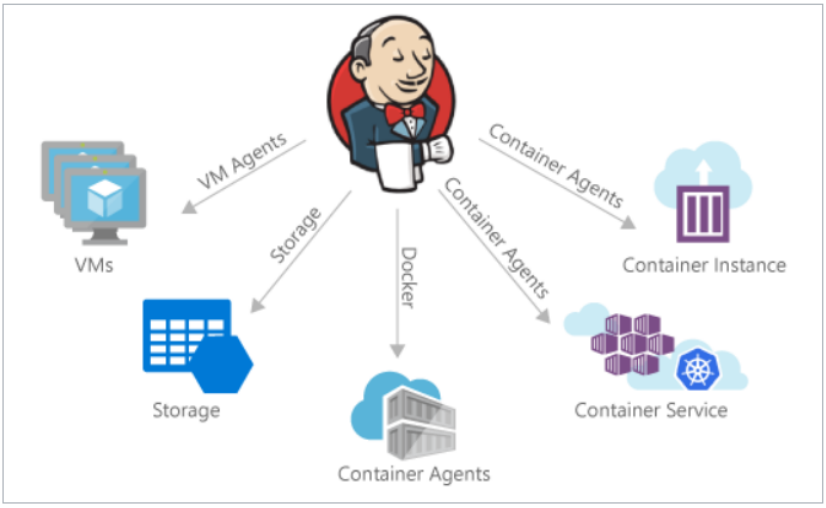
```
az artifacts universal download --organization https://dev.azure.com/fabrikam --feed FabrikamFiber --name my-first-package --version 1.0.0 --path .  --file-filter **/*.exe;**/*.dll
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Jenkins Overview

Jenkins is a very popular Java-based open source continuous integration (CI) server that allows teams to continuously build applications across platforms.

Azure Pipeline also integrates well with Jenkins for teams who already use or prefer to use Jenkins for CI.



There are two ways to integrate Jenkins with Azure Pipelines:

- One way is to run CI jobs in Jenkins separately. This involves configuration of a CI pipeline in Jenkins and a web hook in Azure DevOps that invokes the CI process when source code is pushed to a repository or a branch.
- The alternate way is to wrap a Jenkins CI job inside an Azure pipeline. In this approach, a build definition will be configured in Azure Pipelines to use the Jenkins tasks to invoke a CI job in Jenkins, download and publish the artifacts produced by Jenkin.

In this hands-on lab, Integrating Jenkins CI with Azure Pipelines, you will try both approaches and learn how to:

- Provision Jenkins on Azure VM using the Jenkins template available on the Azure Marketplace
- Configure Jenkins to work with Maven and Azure DevOps
- Create a build job in Jenkins
- Configure Azure Pipeline to integrate with Jenkins
- Configure a CD pipeline in Azure Pipelines to deploy the build artifacts

Setting up the Jenkins VM

To configure Jenkins, the Jenkins VM image available on the Azure Marketplace will be used. This will install the latest stable Jenkins version on an Ubuntu Linux VM along with the tools and plugins configured to work with the Azure.

1. Open in Browser: https://portal.azure.com/#create/bitnami.jenkins1-650
2. Create a New Resource Group and chose all default options

Preparing Jenkins machine to use service hook in Azure DevOps

3. Open https://portal.azure.com/ and access your virtual machine with Jenkins.
4. Click the “Networking” link in Settings tab and click “Add inbound port rule” button.
5. Set inbound rule for port 8080

Connecting to Jenkins from Browser

6. To initiate an SSH tunnel, the following command needs to be run from a Command Prompt.

   D:\> ssh -L 8080:localhost:8080 dssadmin@23.101.205.55

7. For security reasons, Jenkins generates an initial password and save it in a file on the server. This password will need to be retrieved and provided to unlock Jenkins

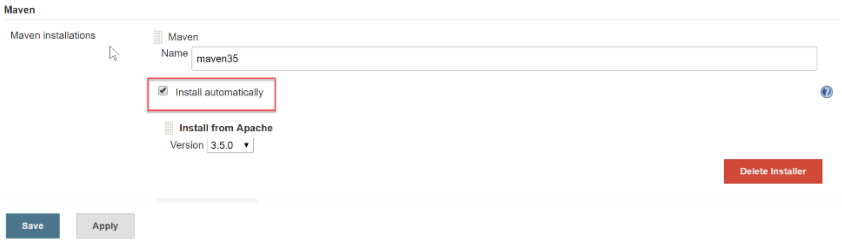
   $ `sudo cat /home/bitnami/bitnami_credentials`

   Note the username and password

8. Open URL http://localhost:8080 in browser → Login → Paste the password from previous step → Continue.
9. Jenkins → People → Select **user** → Configure → Under Password Section → Set the New Password
10. Login again with new password

Installing and Configuring Plugins

11. Manage Jenkins → Manage Plugins → Available tab → search for Team Services → Select **VS Team Services Continuous Deployment** plugin and select **Install without restart**.
12. Manage Jenkins → Manage Plugins → Available tab → search for Maven PlugIn → Select the **Maven Integration Plugin** and select the **Install without restart** button to install the plugin.
13. Restart Jenkins to reflect the updated versions.
14. Manage Jenkins → select the **Global Tool Configuration** option.
15. To install Maven, select the **Install automatically** option and select the Apply button.



Creating a new build job in Jenkins

In Azure Repos

16. In Browser Open https://azuredevopsdemogenerator.azurewebsites.net/?TemplateId=77371&Name=MyShuttle create a new project.
17. Go to your **Azure Repos** and select the **Clone** option. Select **Generate Credentials** and enter a User name and Password. Click **Save Git Credentials** to save.

In Browser: http://localhost:8080

18. **Jenkins** → Home Page → **New Item** → Name = **MyShuttle** → Select Maven Project → OK
19. Select **MyShuttle** → **Configure** → Go to **Source code Management** section. Select Git and provide the clone URL of the Azure DevOps Git repo in the format **http://dev.azure.com/{your org name}/{team project name}/_git/MyShuttle**
20. Select the **Add | Jenkins** option to add a new credential. Provide the **Username and Password** created earlier in Azure Repos click the Add button to close the wizard.

The source code for this application has both **unit tests and UI tests**. We are not ready to run the UI test at this point. So, we will specify to run only the unit tests.

21. **Build** section → **Goals and options field** = **package -Dtest=FaresTest,SimpleTest**
22. **Post-build Actions:** Expand Add post-build action → Archive the artifacts → **File to archive** = **target/*.war, *.sql**
23. Save

The configuration is now completed

Testing the Build

24. Select the **Build Now** option.

The build progress will be displayed on the left pane in the Build History section.

25. To view the build details and the list of build artifacts, select the build number displayed in the Build History section.
26. Select the Test Result option to view the results of the unit tests that were included in the build definition.

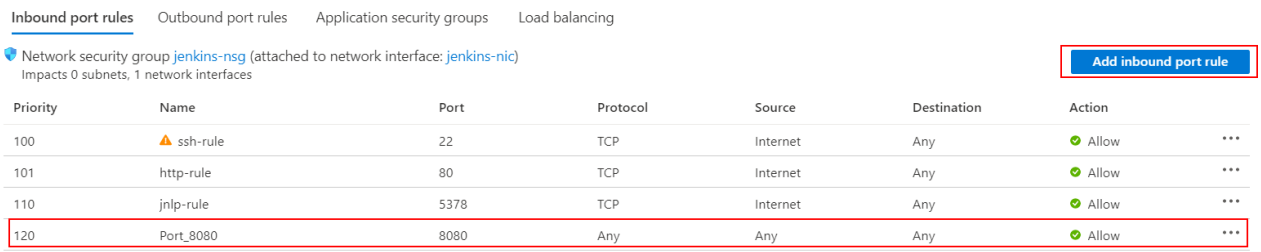
Approach 1: Triggering the CI via a service hook in Azure DevOps

In Jenkins:

1. Jenkins → People → Select admin → configure → **API Token Section** → Add new Token → Copy and Save the token

In Azure Portal

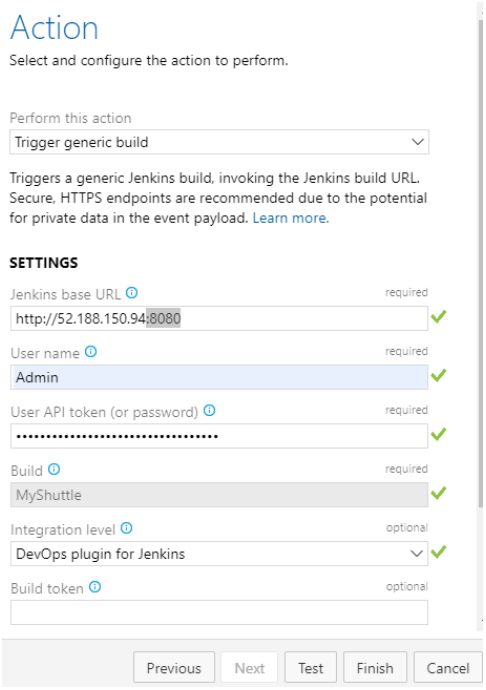
1. Jenkins VM → Networking → Add Inbound Port Rule → Add (by defalt rule is allow 8080 port)



In Azure DevOps:

2. Azure Repos → **Service hooks** under General. Select + Create subscription.

3. Select **Code pushed** for the Trigger on this type of event field. Select the **MyShuttle** repository and then click Next
4. Provide the following details in the Select and configure the action to perform.
    a. Select the **Trigger generic build** option
    b. Provide the Jenkins base URL in **http://{ip address}:8080** format
    c. Provide the **Username (admin) and Password (API Token)** to trigger the build. Note that the username and password is the credentials of the Jenkins administrator user that you configured earlier
    d. Select the Build job you created in Jenkins.

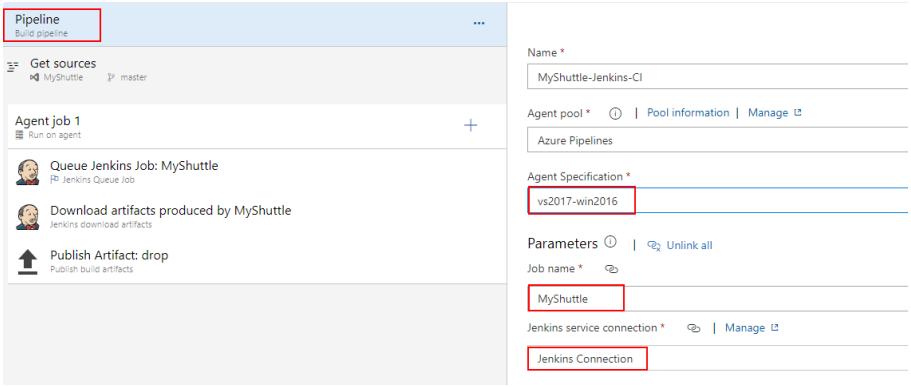


5. Try making a commit to the code - **src/main/webapp/index.jsp** would be a good candidate. This should trigger the MyShuttle build on Jenkins. You can confirm it by checking the history tab of the Jenkins services hook.

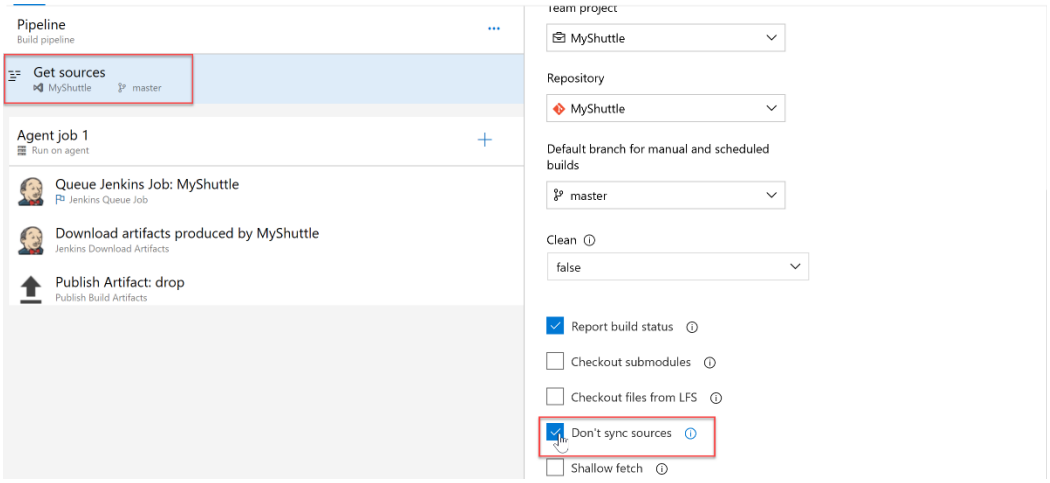
Approach 2: Wrapping Jenkins Job within Azure Pipelines

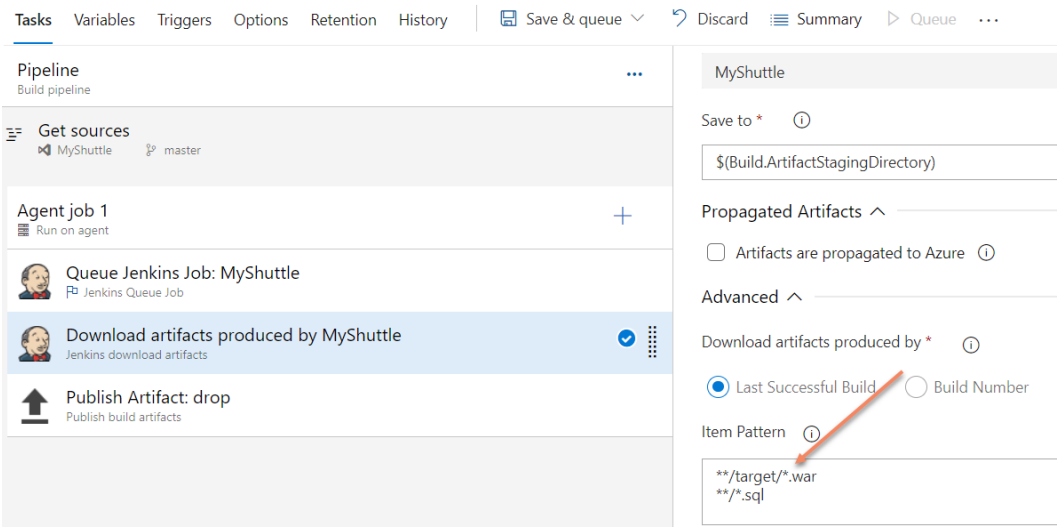
In this approach, Jenkins CI job will be nested within an Azure CI pipeline. The key benefit of this approach is that you can have end-to-end traceability from work items to source code to build and release pipelines.

1. Create a Service Connection (Use API Token) to Jenkins with admin username.
2. Create a New Classic Pipeline with Jenkins Template
3. Provide MyShuttle as the Job name (name of the build definition that was created in Jenkins) and then select the Jenkins service endpoint created earlier.



4. Next, select the Get Sources step. Since Jenkins is being used for the build, **there is no need to download the source code to the build agent.** To skip syncing with the agent, select Don't sync sources option.



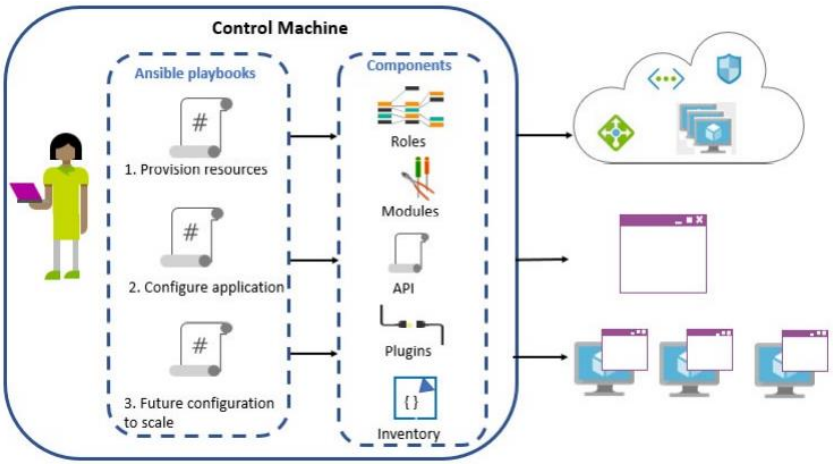


Ansible Overview

- Ansible is an open-source platform by Red Hat that automates cloud provisioning, configuration management, and application deployments.
    - **Provision resources.** Playbooks can provision resources for example load-balancer, virtual networks, network security groups, and VM scale sets on Azure.
    - **Configure the application.** Playbooks can deploy applications to run particular services, such as installing Apache Tomcat on a Linux machine or IIS on Windows to allow you to run a web application.
    - **Future configurations to scale.** Playbooks can alter configurations by applying playbooks to existing resources and applications—in this instance to scale the VMs
- Unlike Puppet or Chef, Ansible is **agentless**.

Ansible Workflow:

The following workflow and component diagram outlines how playbooks can run in different circumstances, one after another. In the workflow, Ansible playbooks:



- **Control node:** Any machine with Ansible installed. You can run Ansible commands and playbooks by invoking the ansible or ansible-playbook command from any control node. You can use any computer that has a Python installation as a control node - laptops, shared desktops, and servers can all run Ansible. However, you cannot use a Windows machine as a control node.
- **Managed nodes:** The network devices (and/or servers) you manage with Ansible. Managed nodes are also sometimes called “hosts”. Ansible is not installed on managed nodes (Ansible as agentless).
- **Inventory:** A list of managed nodes. An inventory file is also sometimes called a “**hostfile**”. Your inventory can specify information like IP address for each managed node. An inventory can also organize managed nodes, creating and nesting groups for easier scaling.
- **Collections:** Collections are a distribution format for Ansible content that can include playbooks, roles, modules, and plugins. You can install and use collections through **Ansible Galaxy**.
- **Modules:** The units of code Ansible executes. Each module has a particular use, from administering users on a specific type of database to managing VLAN interfaces on a specific type of network device. You can

invoke a single module with a task, or invoke several different modules in a playbook. Starting in Ansible 2.10, modules are grouped in collections.

- **Tasks:** The units of action in Ansible. You can execute a single task once with an ad hoc command.
- **Playbooks:** Ordered lists of tasks, saved so you can run those tasks in that order repeatedly. Playbooks can include variables as well as tasks. Playbooks are written in YAML and are easy to read, write, share and understand. To learn more about playbooks, see Intro to playbooks.

Ansible Playbooks.

- Playbooks are ordered lists of tasks that have been saved so you can run them repeatedly in the same order. Playbooks are Ansible's language for configuration, deployment, and orchestration. Playbooks are coded using YAML so as to be human-readable.

You run a playbook using the following command:

ansible-playbook < playbook name >

You can also check the syntax of a playbook using the following command.

ansible-playbook --syntax-check

To see a list of hosts that would be affected by running a playbook, run the command:

ansible-playbook playbook.yml --list-hosts

Ansible Modules.

- A playbook is typically made up of many modules. For example, you could have one playbook containing three modules: a module for creating an Azure Resource group, a module for creating a virtual network, and a module for adding a subnet.
- Ansible works by connecting to your nodes, and then pushing small programs (or units of code)—called modules—out to the nodes. Modules are the units of code that define the configuration. They represent the desired state of the system (declarative), are executed over SSH by default, and are removed when finished.
- For interacting with Azure services, Ansible includes a suite of Ansible cloud modules. These modules enable you to create and orchestrate your infrastructure on Azure.

```
- hosts: localhost
  connection: local
  tasks:
    - name: Creating resource group - "{{ name }}"
      azure_rm_resourcegroup:
        name: "{{ name }}"
        location: "{{ location }}"
      register: rg
    - debug:
        var: rg
```

Installing Ansible

An Ansible installation has the following characteristics:

- You only need to install Ansible on one machine, which could be a workstation or a laptop. You can manage an entire fleet of remote machines from that central point.
- No database is installed as part of the Ansible setup.
- No daemons are required to start or keep Ansible running

Working with Cloud Shell

```
az account list
# Ansible uses by exporting the AZURE_SUBSCRIPTION_ID environment variable.
export AZURE_SUBSCRIPTION_ID=<your-subscription-id>
```

Create and upload file: create_rg.yml

```
- hosts: localhost
  connection: local
  tasks:
    - name: Creating resource group - "{{ name }}"
      azure_rm_resourcegroup:
        name: "{{ name }}"
        location: "{{ location }}"
      register: rg
    - debug:
        var: rg
```

Run the playbook

```
ansible-playbook create-rg.yaml --extra-vars "name=demo_rg location=eastus"
```

Note: Due to the **register** variable and **debug** section of the playbook, the results display when the command finishes.

Delete the Azure Resource Group

```
- hosts: localhost
  tasks:
    - name: Deleting resource group - "{{ name }}"
      azure_rm_resourcegroup:
        name: "{{ name }}"
        state: absent
```

```
  register: rg
- debug:
  var: rg
```

✔ Note: The Windows operating system is not supported as a control machine. However, you can run Ansible from a Windows machine by utilizing other services and products such as Windows Subsystem for Linux, Azure Cloud Shell, and Visual Studio Code

Steps to Configure a VM to Install / Configure for Ansible (Control Machine)

1. Create an Azure Service Principal and note the App ID and App Secret.
2. Create a Ubuntu VM and Configure Ansible.
    a. Create a Linux Ubuntu VM and Install Ansible on it
        i. SSH to VM: SSH dssadmin@20.185.191.3

        If the VM is **Ubuntu 20.04** Linux

```
# Update all packages that have available updates.
sudo apt-get update && sudo apt dist-upgrade -y
# Install Python 3 and pip
sudo apt-get install -y python3-pip
```

        If the VM is **CentOS** Linux

```
# Update all packages that have available updates.
sudo yum update -y
# Install Python 3 and pip.
sudo yum install -y python3-pip
```

        ii. Install Ansible.

```
sudo pip3 install ansible
```

        iii. Install Ansible modules and plugins for interacting with Azure.

```
sudo ansible-galaxy collection install azure.azcollection
```

        iv. Install required modules for Ansible on Azure

```
wget https://raw.githubusercontent.com/ansible-collections/azure/dev/requirements-azure.txt
```

        v. # Install Ansible modules

```
sudo pip3 install -r requirements-azure.txt
sudo ansible --version
```

    b. Create a directory named .azure in the home directory and a credentials file under it.

```
mkdir ~/.azure
nano ~/.azure/credentials
```

c. Insert the following lines into the **credentials** file. Replace the placeholders with the information from the service principal details you copied in the previous task. Press **Ctrl+O** to save the file and **Ctrl+X** to exit from the text editor.

```
[default]
subscription_id=f9baec73-91eb-4458-bd0d-965c1973526d
client_id=c1c5e79f-364b-4194-a49b-af9eec02a1dd
secret=f8lV-~mH1t6eqCnkzLnR1-IpsIc_z-v0~7
tenant=82d8af3b-d3f9-465c-b724-0fb186cc28c7
```

The below two steps are needed only if SSH Service Connection is created using Private Key instead of Password.

d. Ansible is an agentless architecture based automation tool . Only it needs ssh authentication using Ansible Control Machine private/public key pair. Now let us create a pair of private and public keys. Run the following command to generate a private/public key pair for ssh and to install the public key in the local machine.

```
ssh-keygen -t rsa
chmod 755 ~/.ssh (Allow everyone read and execute – owner can also write)
touch ~/.ssh/authorized_keys (Creates a File for keys)
chmod 644 ~/.ssh/authorized_keys (only owner can read and write and others can read only)
ssh-copy-id dssadmin@127.0.0.1
yes
<enter password>
```

Note: https://chmod-calculator.com/

e. In the next task, you need SSH private key to created SSH endpoint in Azure DevOps service. Run the following command to get the private key. Copy the private key to notepad.

```
cat ~/.ssh/id_rsa
```

3. Create a SSH Service Connection in Azure DevOps
   a. Project Properties → Service Connection → New Service Connection → SSH
   b. In **Add an SSH service connection** window provide the required details and click **OK** to save the connection.
   c. Paste the private key which you copied in the previous task.
4. Add the Ansible Playbook create_vm.yml File to the repository

```
- name: Create Azure VM
  hosts: localhost
  connection: local
  vars:
    resource_group: ansible_rg
```

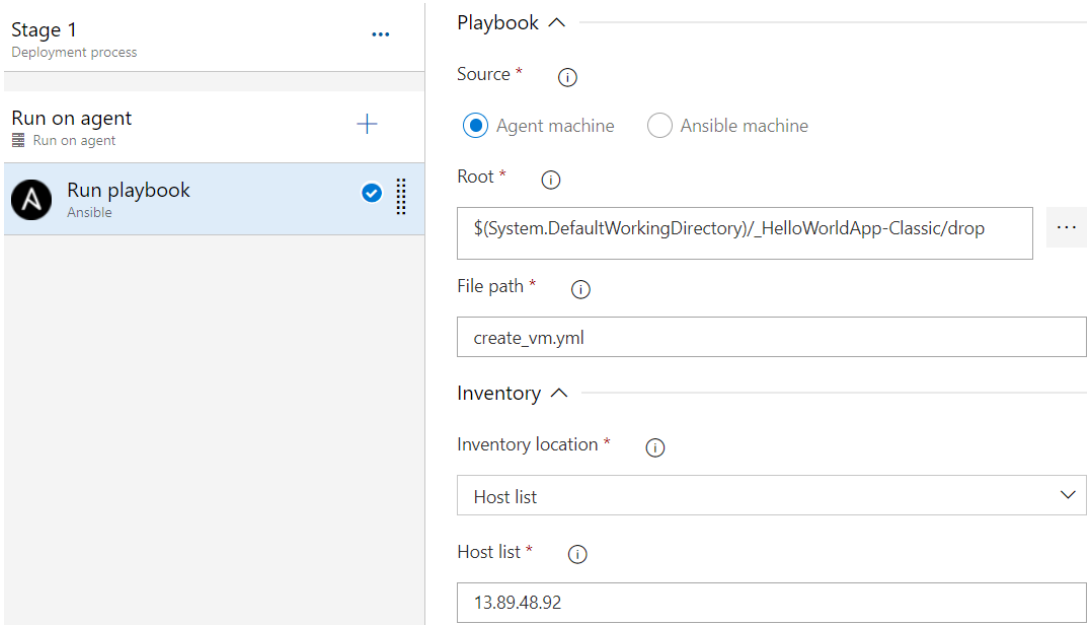
```
  location: eastus
tasks:
- name: Create resource group
  azure_rm_resourcegroup:
    name: "{{ resource_group }}"
    location: "{{ location }}"

- name: Create virtual network
  azure_rm_virtualnetwork:
    resource_group: "{{ resource_group }}"
    name: myVnet
    address_prefixes: "10.0.0.0/16"

- name: Add subnet
  azure_rm_subnet:
    resource_group: "{{ resource_group }}"
    name: mySubnet
    address_prefix: "10.0.1.0/24"
    virtual_network: myVnet

- name: Create public IP address
  azure_rm_publicipaddress:
    resource_group: "{{ resource_group }}"
    allocation_method: Static
    name: myPublicIP
  register: output_ip_address

- name: Dump public IP for VM which will be created
  debug:
    msg: "The public IP is {{ output_ip_address.state.ip_address }}."

- name: Create Network Security Group that allows SSH
  azure_rm_securitygroup:
    resource_group: "{{ resource_group }}"
    name: myNetworkSecurityGroup
    rules:
      - name: SSH
        protocol: Tcp
```

```
      destination_port_range: 22
      access: Allow
      priority: 1001
      direction: Inbound

- name: Create virtual network inteface card
  azure_rm_networkinterface:
    resource_group: "{{ resource_group }}"
    name: myNIC
    virtual_network: myVnet
    subnet_name: mySubnet
    security_group: myNetworkSecurityGroup
    ip_configurations:
      - name: ipconfig1
        public_ip_address_name: myPublicIP
        primary: True

- name: Create VM
  azure_rm_virtualmachine:
    resource_group: "{{ resource_group }}"
    no_log: true
    name: CreatedbyAnsible-vm
    vm_size: Standard_DS1_v2
    admin_username: azureuser
    ssh_password_enabled: true
    admin_password: Password@123
    network_interfaces: myNIC
    image:
      offer: CentOS
      publisher: OpenLogic
      sku: '7.5'
      version: latest
```

5. In the build pipeline add a step to copy the YML to **$(build.outputstagingdirectory)**
6. Run the build pipeline
7. Create / Edit a Release Pipeline and Add the following steps
    a. **Replace Tokens** to replace any tokens if used in Playbook file

b. **Ansible Task**: This task is to integrate with Ansible. This task executes a given Ansible playbook on a specified list of inventory nodes via command line interface. This task requires that the Playbook files be located either on a private Linux agent or on a remote machine where Ansible automation engine has been installed. Select Ansible Location as **Remote Machine** and select **Ansible SSH endpoint** that you created in **Task 3**.

c. Under the **Inventory** section, select **Host list** as inventory location and enter **pubic ip** of your ansible vm in **Host list** field.



Ansible to **deallocate (stop)** an Azure virtual machine.

```
- name: Deallocate Azure VM
  hosts: localhost
  connection: local
  tasks:
    - name: Stop virtual machine
      azure_rm_virtualmachine:
        resource_group: {{ resource_group_name }}
        name: {{ vm_name }}
        allocated: no
```

Note: In the same script you can delete **allocated property** to start the VM.

Ansible Script to Login, Pull and Run Docker Image

```
- name: Log into DockerHub
  docker_login:
    username: sandeepsoni
    password: XXXXXXXX
```

```
- name: pull an image
  docker_image:
    name: sandeepsoni/helloworldapp
    source: pull

- name: Create a data container
  docker_container:
    name: mydata
    image: sandeepsoni/helloworldapp
    volumes:
      - /data
```

Playbook for installing IIS

To install it use: ansible-galaxy collection install ansible.windows**.**

```
- name: Install IIS Web-Server with sub features and management tools
  ansible.windows.win_feature:
    name: Web-Server
    state: present
    include_sub_features: yes
    include_management_tools: yes
  register: win_feature

- name: Reboot if installing Web-Server feature requires it
  ansible.windows.win_reboot:
  when: win_feature.reboot_required
```

https://www.ntweekly.com/2020/06/10/manage-windows-machines-with-ansible/

Agenda: IaC using ARM Templates

- About Infrastructure as Code (IaC)
- About ARM Templates
- Sample to Create Storage Account using ARM Template
- Deploy Templates using PowerShell
- Deploy Templates using Azure Portal
- Deploy Templates using Azure Pipeline
- Incremental and Complete Deployment
- Creating VM using ARM Template
- Create linked ARM Templates
- Creating Resource Group and Resources at Subscription Level

About Infrastructure as Code (IaC)

This module is about expressing your infrastructure requirements as code. By adopting this practice, you can automatically provision the infrastructure that you need for your application or service.

Keeping up with changes in your infrastructure is a challenge. New resources need to be built. Old resources need maintenance. Environments need to be consistent. *Infrastructure as code* enables you to describe, through code, the infrastructure that you need for your application.

What is infrastructure as code (IaC)?

Infrastructure as code enables you to describe, through code, the infrastructure that you need for your application.

Infrastructure as code enables you to maintain both your application code and everything you need to deploy your application in a central code repository. In that repository, you can build, test, and deploy as a unit.

Your infrastructure and configuration code is stored in version control, along with the application code. You'll have lots more wall space.

As you move to infrastructure as code, you're going to see a lot of improvements.

- Consistent configurations
- Improved scalability
- Faster deployments
- Less documentation because the scripts replace it
- Better traceability

Infrastructure as code is a best practice in DevOps because it helps bring teams together. The development team gets a better understanding of where their code is being deployed to and run. Operations teams get

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

involved in the process much earlier and can use infrastructure choices to help developers make the right design decisions.

What tools can you use to apply infrastructure code?

- Azure CLI
- Azure PowerShell
- Azure SDK
- Azure Resource Manager templates
- Terraform
- Ansible

Azure Resource Manager templates

About Resource Provider

Each resource provider offers a set of resources and operations for working with an Azure service. For example, if you want to store keys and secrets, you work with the Microsoft.KeyVault resource provider. This resource provider offers a resource type called vaults for creating the key vault.

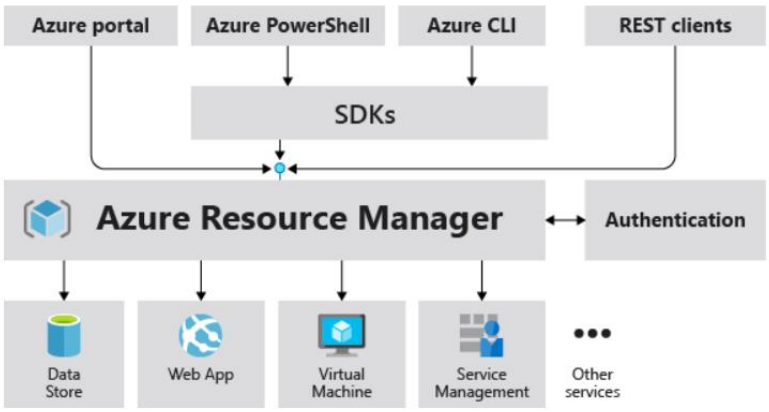
The name of a resource type is in the format: **{resource-provider}/{resource-type}**. For example, the key vault type is **Microsoft.KeyVault/vaults**.

Before getting started with deploying your resources, you should gain an understanding of the available resource providers. Knowing the names of resource providers and resources helps you define resources you want to deploy to Azure. Also, you need to know the valid locations and API versions for each resource type.

About Azure Resource Manager:

The infrastructure that makes up your application is often composed of various different components. For instance, you might simply be running a web site, but behind the scenes you have an Azure web site deployed, a Storage account for tables, blobs, and queues, a couple of VMs running a database cluster, etc…

The old way of doing things was to use the **Service Management API**. Each piece of infrastructure was **treated separately** in the Azure Portal. Deployment consisted of manually creating them, or lots of effort creating scripts using PowerShell. You would have to handle trying to initialize infrastructure in serial or parallel yourself. If you wanted to deploy just part of your infrastructure you would have to understand **what was already there** and take actions accordingly, either manually or in your scripts.



Azure Resource Manager Overview:

Azure Resource Manager enables you to work with the resources in your solution as a group. You can deploy, update, or delete all the resources for your solution in a single, coordinated operation.

- **Azure Resource Manager** allows you to define **a declarative template** for your group of resources.
- A resource manager template is **a JavaScript Object Notation (JSON)** file that defines one or more resources to deploy to a resource group.
- You use that template for deployment and it can work for different environments such as **testing, staging, and production** and have confidence your resources are deployed in a consistent state.
- Azure handles things like **parallelizing** as much of the deployment as possible without any extra effort on your part.
- You can define the dependencies between resources so they're deployed in **the correct order**.
- It also deploys in an **idempotent** way i.e. **incrementally adds** anything **missing** while leaving anything already deployed in place, so you can **rerun** the template deployment multiple times safely.

About Azure Resource Manager (ARM) Templates

Resource Manager enables you to export a Resource Manager template from existing resources in your subscription.

It is important to note that there are **two** different ways to export a template:

1. You can **export the actual template that you used for a deployment**. The exported template includes all the parameters and variables exactly as they appeared in the original template. This approach is helpful when you have deployed resources through the portal. Now, you want to see how to construct the template to create those resources.

   Resource Group → Deployments
2. You can **export a template that represents the current state of the resource group**. It creates a template that is a snapshot of the resource group. The exported template has many hard-coded values and probably not as many parameters as you would typically define. This approach is useful when you have

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

modified the resource group through the portal or scripts. Now, you need to capture the resource group as a template.

Resource Group → Export Template.

Sample to Create a Storage Account: Template.json

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentTemplate.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "storageName": {
   "type": "string",
   "minLength": 3,
   "maxLength": 24
  },
  "storageSKU": {
   "type": "string",
   "defaultValue": "Standard_LRS",
   "allowedValues": [
    "Standard_LRS",
    "Standard_GRS",
    "Standard_RAGRS",
    "Standard_ZRS",
    "Premium_LRS",
    "Premium_ZRS",
    "Standard_GZRS",
    "Standard_RAGZRS"
   ]
  },
  "location": {
   "type": "string",
   "defaultValue": "[resourceGroup().location]"
  }
 },
 "resources": [
  {
   "type": "Microsoft.Storage/storageAccounts",
```

```
    "apiVersion": "2019-04-01",
    "name": "[parameters('storageName')]",
    "location": "[parameters('location')]",
    "sku": {
      "name": "[parameters('storageSKU')]"
    },
    "kind": "StorageV2",
    "properties": {
      "supportsHttpsTrafficOnly": true
    }
  }
 ],

  "outputs": {
     "storageUri": {
        "type": "string",
        "value": "[reference(parameters('storageName')).primaryEndPoints.blob]"
     }
  }
}
```

Sample: **Parameters.json**

```
{
 "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
 "contentVersion": "1.0.0.0",
 "parameters": {
  "storageName": {
    "value": "devstore"
  },
  "storageSKU": {
    "value": "Standard_LRS"
  }
 }
}
```

Deploy Templates using PowerShell

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Azure Portal → Cloud Shell → Upload file

PowerShell commands to Execute the ARM Template

```
$templateFile = "$Home/StorageTemplate.json"
$parameterFile="$Home/StorageParameter.json"
New-AzResourceGroup `
 -Name myDemoRG `
 -Location "East US"

New-AzResourceGroupDeployment `
 -Name devenvironment `
 -ResourceGroupName myDemoRG `
 -TemplateFile $templateFile `
 -TemplateParameterFile $parameterFile
```

Note: Validate your deployment settings to find problems before creating actual resources.

```
Test-AzResourceGroupDeployment -ResourceGroupName Sandbox-rg -TemplateFile <PathToTemplate>
```

You have the following options for providing parameter values:

```
New-AzResourceGroupDeployment -Name ExampleDeployment -ResourceGroupName
ExampleResourceGroup -TemplateFile <PathToTemplate> -TemplateParameterFile $parameterFile -
myParameterName1 "parameterValue1" -myParameterName2 "parameterValue2"
```

Deploy the Template using Portal

To Save the template:

1. More Services → Template spec → +Add
2. Name="DemoVMTemplate", Description="This is a test template" → OK
3. Copy the content of Storage.json and paste in the Text Editor → OK
4. Click Add.

To Execute the Template

5. More Services → Search for "**Template spec**"
6. Select the template with the name you saved earlier (DemoVMTemplate)**.**
7. Template Blade → Deploy
8. Click on Edit parameters → Copy and paste content from Storage-Parameters.json → Save
9. Create a New RG and change the parameters as needed → Check I agree . . . → Purchase

CI with Azure Pipeline

1. Add the below two files to Git repository
2. Put the following in **BUILD** Stage so the Template and its parameter files are included as Build Artifacts.

```
- task: CopyFiles@2
  displayName: 'Copy Files to: $(build.artifactstagingdirectory)'
  inputs:
    SourceFolder: HelloWorldApp.Templates
    TargetFolder: '$(build.artifactstagingdirectory)'
```

3. Put the following in **Dev stage**

```
- task: AzureResourceManagerTemplateDeployment@3
  displayName: 'ARM Template deployment: Resource Group scope'
  inputs:
    azureResourceManagerConnection: 'AzureSubscriptionServiceConnection'
    subscriptionId: 'f9baec73-91eb-4458-bd0d-965c1973526d'
    resourceGroupName: DemoRG
    location: 'East US'
    csmFile: '$(System.DefaultWorkingDirectory)/**/Storage-Template.json'
    csmParametersFile: '$(System.DefaultWorkingDirectory)/**/Storage-Parameters.json'
    templateLocation: 'Linked artifact'
    deploymentMode: 'Incremental'
```

4. Save and Run the pipeline

AzSK ARM Template Checker

With the adoption of infrastructure as code and development teams taking on more of the Ops activities, organisations using Azure for enterprise hosting are practically at risk of breach with every new release they roll out to these environments… In this tutorial we'll walk through AzSDK Security Verification Tests that can be setup in the CI/CD pipeline using Azure DevOps to automate the inspection of your infrastructure in Azure and block releases that can potentially compromise your infrastructure.

The DevOps way of working coupled with the use of cloud technologies has practically removed the biggest barriers from the software delivery life-cycle… Development teams don't care about security as much as they should. It's hard to blame the development teams though, the tools used by security are not easy to understand… The reports are long and hard to interpret… The approach of compliance driven security doesn't practically fit into the fast pace of software delivery we are used to today.

DevSecOps helps bring a fresh perspective by introducing a culture of making everyone accountable for security, using automation to move the process of inspection left and overall looking at security with a 360 lens.

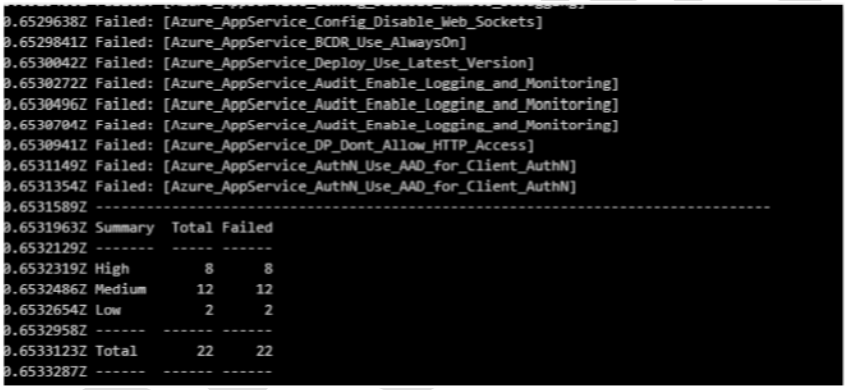
Anyone doing cloud at scale is likely to have multiple Azure subscriptions with hundreds of resource groups in Azure dedicated to the application in question. Resource Groups comprising of resources such as Web App's, Azure Functions, Blob Storage, Redis Cache, App Insights, Service Bus, SQL warehouse among some other Azure resource types. The ratio of security consultants to the number of releases makes it practically impossible for security to inspect the changes to these resource types to guarantee compliance of best practices.

Walkthrough:

1. Start by installing the **AzSDK** extension from the Azure DevOps marketplace
2. Create a build definition that is mapped to the git repository where you store your ARM templates

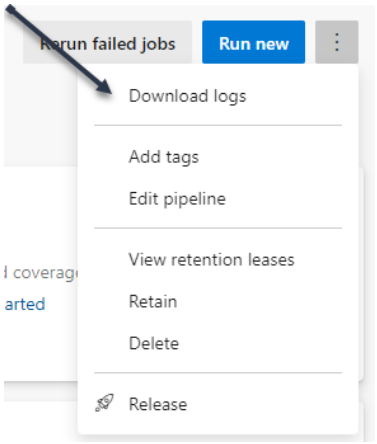
```
- task: azsdktm.AzSDK-task.azsk-cicd-armchecker.AzSKARMTemplateChecker@4
  displayName: 'AzSK_ARMTemplateChecker '
  inputs:
    ARMTemplateFilePath: HelloWorldApp.Web/scripts
    ARMTemplateParameterFilePath: HelloWorldApp.Web/scripts
```

3. Run the build and let the build complete. You will see glimpse of the scan in the build output itself.



4. Once the build completes, you will find that task has attached all the results to the build log. You will be surprised to see the issues you find in your ARM templates
5. See the summary "CSV" and detailed "LOG" output files for the AzSK_ARMTemplateChecker. The locations are on the **build/release agent machine**. These are also packaged up in an overall ZIP file and are available

to download. The overall ZIP file can be downloaded by clicking on the "**Download logs**" option.



Note: The task currently scans App Service, Storage, SQL, CDN, Traffic Manager, Document DB, Redis Cache, and Data Lake services only.

More: https://github.com/azsk/DevOpsKit-docs/blob/master/03-Security-In-CICD/Readme.md#execute-arm-template-checker-in-baseline-mode

Incremental and Complete Deployments

When deploying your resources, you specify that the deployment is either an **incremental** update or a **complete** update.

- In complete mode, Resource Manager **deletes** resources that exist in the resource group but are not specified in the template.
- In incremental mode, Resource Manager **leaves unchanged** resources that exist in the resource group but are not specified in the template.

Existing Resource Group contains:

- Resource A
- Resource B
- Resource C

New Template defines:

- Resource A
- Resource B
- Resource D

When deployed in **incremental** mode, the resource group contains:

- Resource A
- Resource B
- Resource C
- Resource D

When deployed in **complete** mode, Resource C is deleted. The resource group contains:

- Resource A
- Resource B
- Resource D

PowerShell Command:

```
New-AzResourceGroupDeployment -Name "ExampleDeployment123" -Mode Complete -ResourceGroupName DemoRG -TemplateFile "D:\template.json" -TemplateParameterFile 'D:\parameters.json'
```

CLI Command:

```
az deployment group create --name ExampleDeployment --mode Complete --resource-group DemoRG --template-file template.json --parameter-file @parameters.json
```

Create Windows VM using ARM Template

1. Open the following files from DevOps folder
   - AzureVMTemplate.json
   - AzureVMTempalte-parameters.json

There are five resources defined by the template:

1. Microsoft.Storage/storageAccounts.
2. Microsoft.Network/publicIPAddresses.
3. Microsoft.Network/virtualNetworks.
4. Microsoft.Network/networkInterfaces.
5. Microsoft.Compute/virtualMachines.

2. Using Cloud Shell, upload the file AzureVMTempalte.json
3. Copy and Paste the following in Cloud Shell

```
$resourceGroupName = Read-Host -Prompt "Enter the resource group name"
$storageAccountName = Read-Host -Prompt "Enter the storage account name"
$newOrExisting = Read-Host -Prompt "Create new or use existing (Enter new or existing)"
$location = Read-Host -Prompt "Enter the Azure location (i.e. centralus)"
$vmAdmin = Read-Host -Prompt "Enter the admin username"
$vmPassword = Read-Host -Prompt "Enter the admin password" -AsSecureString
$dnsLabelPrefix = Read-Host -Prompt "Enter the DNS Label prefix"

New-AzResourceGroup -Name $resourceGroupName -Location $location
```

```
New-AzResourceGroupDeployment `
  -ResourceGroupName $resourceGroupName `
  -adminUsername $vmAdmin `
  -adminPassword $vmPassword `
  -dnsLabelPrefix $dnsLabelPrefix `
  -storageAccountName $storageAccountName `
  -newOrExisting $newOrExisting `
  -TemplateFile "$HOME/AzureVMTemplate.json"
```

Integrate Key Vault

open *azuredeploy.parameters.json*

```
"adminPassword": {
  "reference": {
    "keyVault": {
    "id":
"/subscriptions/<SubscriptionID>/resourceGroups/<ResGroup>/providers/Microsoft.KeyVault/vaults/<KeyVaultName>"
    },
    "secretName": "vmAdminPassword"
  }
},
```

Create linked Azure Resource Manager templates

- **The main template**: create all the resources except the storage account.
- **The linked template**: create the storage account.

Call the linked template

1. Open AzureVMTemplate.json
2. Replace the storage account resource definition with the following json snippet:

```
{
 "name": "linkedTemplate",
 "type": "Microsoft.Resources/deployments",
 "apiVersion": "2018-05-01",
 "properties": {
   "mode": "Incremental",
   "templateLink": {
```

```
      "uri":"https://raw.githubusercontent.com/deccansoft/Demos/master/StorageTemplate.json"
    },
    "parameters": {
      "storageAccountName":{"value": "[variables('storageAccountName')]"},
      "location":{"value": "[parameters('location')]"}
    }
  }
},
```

Pay attention to these details:

- A Microsoft.Resources/deployments resource in the main template is used to link to another template.
- The deployments resource has a name called linkedTemplate. This name is used for configuring dependency.
- You can only use Incremental deployment mode when calling linked templates.
- templateLink/uri contains the linked template URI. Update the value to the URI you get when you upload the linked template (the one with a SAS token).
- Use parameters to pass values from the main template to the linked template.

3. Make sure you have updated the value of the uri element
4. Expand the virtual machine resource definition, update **dependsOn** and **storageUri** as shown in the following screenshot:

5. Azure Portal → Cloud Shell → Upload file
6. PowerShell commands to Execute the ARM Template

```
$templateFile = "$Home/AzureVMTemplateWithLinkToStorage.json"
$parameterFile="$Home/AzureVMTemplateWithLinkToStorage-parameters.json"
New-AzResourceGroup `
 -Name myResourceGroupDev `
 -Location "East US"
New-AzResourceGroupDeployment `
 -Name devenvironment `
 -ResourceGroupName myResourceGroupDev `
 -TemplateFile $templateFile `
 -TemplateParameterFile $parameterFile
```

Creating Resource group and Resources at Subscription level

To create the resource group and deploy resources to it, use a nested template. The nested template defines the resources to deploy to the resource group. Set the nested template as dependent on the resource group to make sure the resource group exists before deploying the resources.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

The following example creates a resource group, and deploys a storage account to the resource group.

```
{
 "$schema": "https://schema.management.azure.com/schemas/2018-05-
01/subscriptionDeploymentTemplate.json#",
 "contentVersion": "1.0.0.1",
 "parameters": {
  "rgName": {
   "type": "string",
   "defaultValue": "mydemorg"
  },
  "rgLocation": {
   "type": "string",
   "defaultValue": "eastus"
  },
  "storagePrefix": {
   "type": "string",
   "defaultValue": "dss",
   "maxLength": 11
  }
 },
 "variables": {
  "storageName": "[concat(parameters('storagePrefix'), uniqueString(subscription().id, parameters('rgName'))
)]"
 },
 "resources": [
  {
   "type": "Microsoft.Resources/resourceGroups",
   "name": "[parameters('rgName')]",
   "apiVersion": "2018-05-01",
   "location": "[parameters('rgLocation')]",
   "properties": {}
  },
  {
   "name": "storageDeployment",
   "type": "Microsoft.Resources/deployments",
   "apiVersion": "2018-05-01",
```

```
    "name": "storageDeployment",
    "resourceGroup": "[parameters('rgName')]",
    "dependsOn": [
      "[resourceId('Microsoft.Resources/resourceGroups/', parameters('rgName'))]"
    ],
    "properties": {
      "mode": "Incremental",
      "template": {
        "$schema": "https://schema.management.azure.com/schemas/2015-01-
01/deploymentTemplate.json#",
        "contentVersion": "1.0.0.0",
        "parameters": {},
        "variables": {},
        "resources": [
          {
            "type": "Microsoft.Storage/storageAccounts",
            "apiVersion": "2017-10-01",
            "name": "[variables('storageName')]",
            "location": "[parameters('rgLocation')]",
            "sku": {
              "name": "Standard_LRS"
            },
            "kind": "StorageV2"
          }
        ],
        "outputs": {}
      }
    }
  ],
  "outputs": {}
}
```

Execute the Command at the subscription-level

```
New-AzDeployment `
 -Name demoDeployment `
```

```
-Location centralus `
-TemplateUri $HOME/storagedeploy.json `
```

Note: Parameters have default value.

Note that we have used the command New-AzDeployment and not New-Az**ResourceGroup**Deployment

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Key Vault

- Introduction to Key Vault
- Secrets vs Keys
- How it Works
- Key Vault in CI and CD Pipeline
- Replace Token Task

About Key Vault

- Key Vault serves as a store of cryptographic keys and secrets, such as authentication keys, storage account keys, data encryption keys, .PFX files, and passwords.
- Developers can create keys for development and testing in minutes, and then seamlessly migrate them to production keys.
- Security administrators can grant (and revoke) permission to keys, as needed.

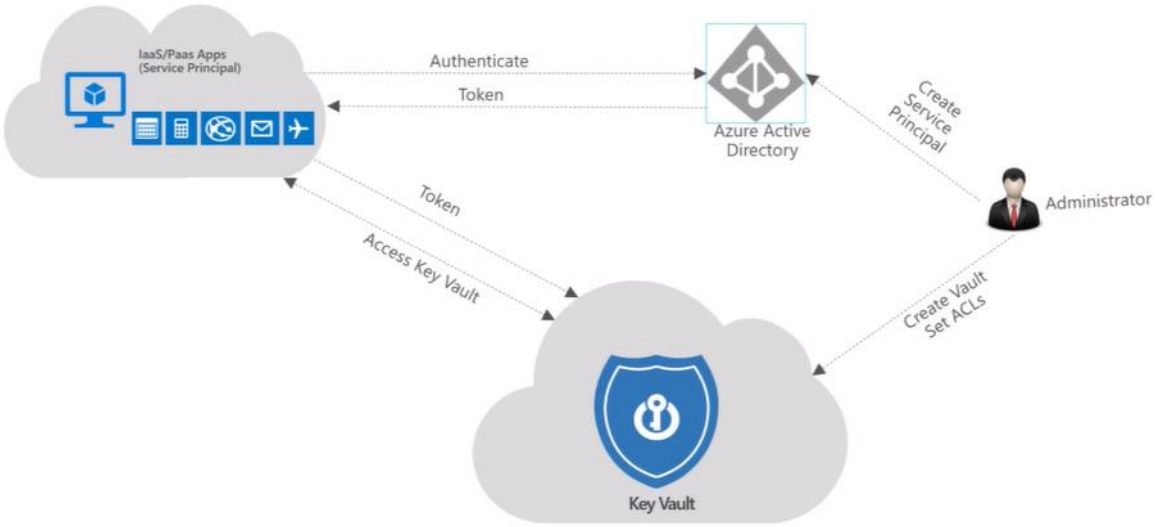
1. Creates a key vault.
2. Authorizes applications and users for specific operations.
3. Add keys and secrets to key vault.
4. Configure application with URI of key or secret or entire vault
5. Use secrets and keys in the key vault.
   Or, less commonly, add / update keys and secrets in the key vault.
6. Monitors key vault logs.
7. Update keys and secrets as needed.
8. Updates permissions as needed.
9. Delete key or secret when no longer needed.
10. Deletes key vault when no longer needed.

How to access keys and secrets:

- To access keys and secrets, users and applications must possess valid Azure Active Directory tokens representing security principal with sufficient permissions to the target vault.
- You can use a REST-based API or Azure PowerShell to retrieve secrets and public parts of keys (in JSON format) from Key Vault.

How it works:

1. Applications that need access to Key Vault are registered with Azure Active Directory as Service Principals.
2. Administrator will then create a Key Vault and sets Access Control Lists on the vault so that applications can access it.
3. Applications then authenticates with Azure Active Directory and gets the Token.
4. The application then presents this token to Azure Key Vault
5. Azure Key vault then grants access based on Access Control List (ACL)



Walkthrough: Steps to create Hello Key Vault Application

1. Create Service Principal (Azure AD Application)
2. Create Key Vaults and Set ACL's
3. Add Keys and Secrets to the Key Vault.
4. Get Sample Code and Edit config configuration.
5. Build and Run the application.

Step 1: Create Azure Active Directory Application (Service Principal)

1. Azure Portal → Azure Active Directory → App registrations → **+ New application registration**
2. Provide Name="**KeyVaultDemoApp**"
3. KeyVaultDemoApp → Properties → **Copy App ID URI**
4. KeyVaultDemoApp → Certificates and Secrets → Click on **+ New client secret** and copy the same.

Step 2: Create Key Vaults

5. Azure Portal → All Services → Key Vault → +Add
6. Enter Name = DssDemoKeyVault
7. Add **Access Policy** → +Add
    a. Select Secret Persmissions (Get / List)
    b. Select Principals = "***KeyVaultDemoApp***" (AAD Application Created before)
    c. Authorized Application is disabled
8. Create
9. Go to Created Key Vault → Essentials → Copy DNS Name: https://dssdemokeyvault.vault.azure.net/

Step 3: Add a key or secret to the key vault

10. Select the DssDemoKeyVault → Secrets → +Add
11. Options = Generate, Name=MyFirstSec → Create
12. Click on Key → Current Version → Copy Identifier

You can reference a key that you created or uploaded to Azure Key Vault by using its URI.

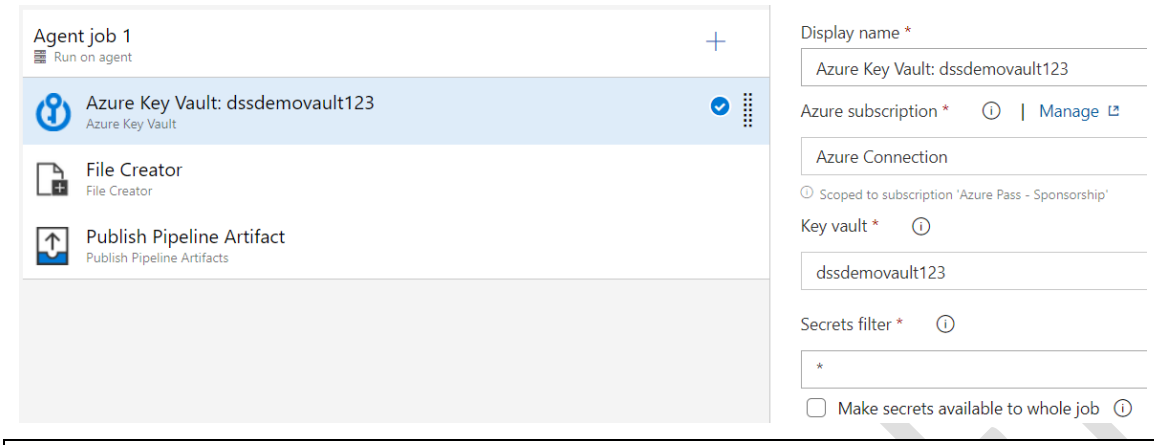
13. Get the current version, you can use https://dssdemokeyvalut.vault.azure.net/secrets/MyFirstSecret and use https://dssdemokeyvalut.vault.azure.net/secrets/MyFirstSecret/cgacf4f763ar42ffb0a1gca546aygd87 to get this specific version.

Key Vault in Azure DevOps CI and CD Pipeline

Azure Key Vault Task:

Azure Pipelines will fetch the latest values of the secrets and set them as task variables which can be consumed in the following tasks.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)




```yaml
trigger:
- master

pool:
  vmImage: ubuntu-latest

steps:
- task: AzureKeyVault@1
  displayName: 'Azure Key Vault: dssdemovault123'
  inputs:
    azureSubscription: 'Azure Subscription MPN'
    KeyVaultName: 'dss-keyvault-demo'
    SecretsFilter: '*'
    RunAsPreJob: false

- task: file-creator@6
  inputs:
    filepath: '$(build.artifactstagingdirectory)/demo.txt'
    filecontent: |
      $(Server)
      $(Password)

- task: PublishPipelineArtifact@1
  displayName: 'Publish Pipeline Artifact'
```

```
inputs:
  targetPath: '$(build.artifactstagingdirectory)'
```

Replace Token Task

Note: Replace Token Task can be used to replace Tokens in any file (mostly configuration file) with values of Variables in the pipeline. The token in the pipelines must be enclosed in between some prefix and suffic. Example below replaces all values in appsettings.json with variables of pipeline provided they are enclosed inbetween [* and *]

```
- task: qetza.replacetokens.replacetokens-task.replacetokens@3
  displayName: 'Replace tokens in $(build.sourcedirectory)/**/appsettings.json'
  inputs:
    targetFiles: '$(build.sourcedirectory)/**/appsettings.json'
    tokenPrefix: '[*'
    tokenSuffix: '*]'
```

Example of AppSettings.json

```
{
  "AllowedHosts": "*",
  "ConnectionString": "server=[*SQLServer*];Database=[*Database*];uid=[*Login*];Password=[*Password*]"
}
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Scan Code During CI

- Sources and Impacts of Technical Dept
- Managing Technical Dept with DevOps and Sonar Cloud
- Scan open-source components using WhiteSource Bolt

What is Ruged DevOps?

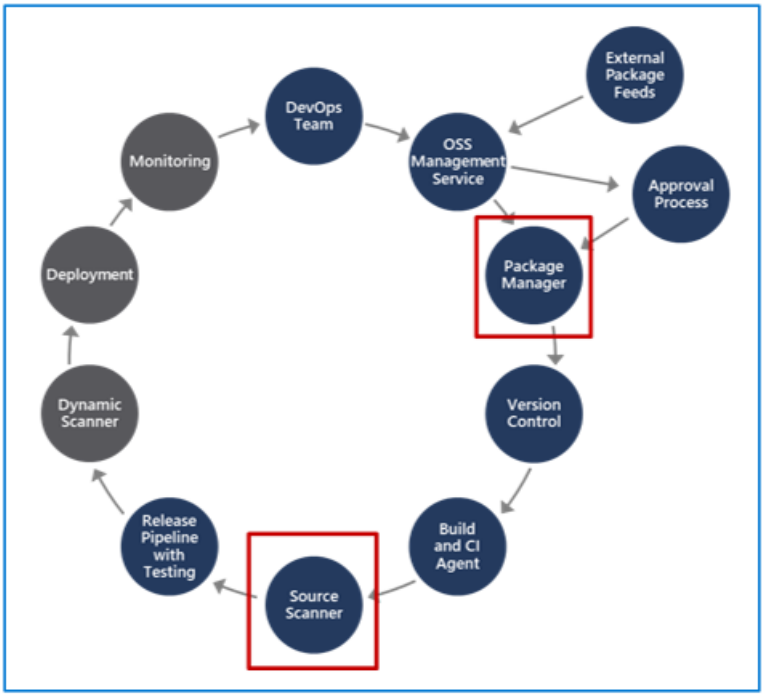
- Rugged DevOps is a set of practices designed integrate DevOps and security.
- The goal is to enable development teams to work fast without introducing unwanted vulnerabilities.
- Security strategy includes access control, environment hardening, perimeter protection, and more…

Rugged DevOps includes bigger questions such as:

- Is my pipeline consuming third-party components, and if so, are they secure?
- Are there known vulnerabilities within any of the third-party software we use?
- How quickly can I detect vulnerabilities (*time to detect*)?
- How quickly can I remediate identified vulnerabilities (*time to remediate*)?

Rugged DevOps Pipeline:

- **Source Scanner** is for performing a security scan to verify certain security vulnerabilities are not present in our application source code.
- **Package Management,** and the approval process associated with it, accounts for how software packages are added to the pipeline, and the approval process they need to go through.



Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Sources and Impacts of Technical Debt

Technical Debt is a term that describes the **future penalty** that you incur today by making easy or quick choices in software development practices, rather than taking harder or longer choices that are more appropriate longer-term. Over time, it accrues in much the same way that monetary debt does.

Common sources of technical debt are:

- Lack of coding style and standards.
- Insufficient comments and documentation.
- Not writing self-documenting code (including class, method and variable names that are descriptive or indicate intent).
- Lack of or poor design of unit test cases.
- Ignoring or not understanding object orient design principles.
- Monolithic classes and code libraries.
- Poorly envisioned use of technology, architecture and approach. (Forgetting that all attributes of the system, affecting maintenance, user experience, scalability, and others, need to be considered).
- Over-engineering code (adding or creating code that is not needed, adding custom code when existing libraries are sufficient, or creating layers or components that are not needed).
- Taking shortcuts to meet deadlines.
- Leaving dead code in place.

Over time, technical debt must be paid back. Otherwise, the team's ability to fix issues, and to implement new features and enhancements will take longer and longer, and eventually become cost-prohibitive.

Using Automated Testing to Measure Technical Debt

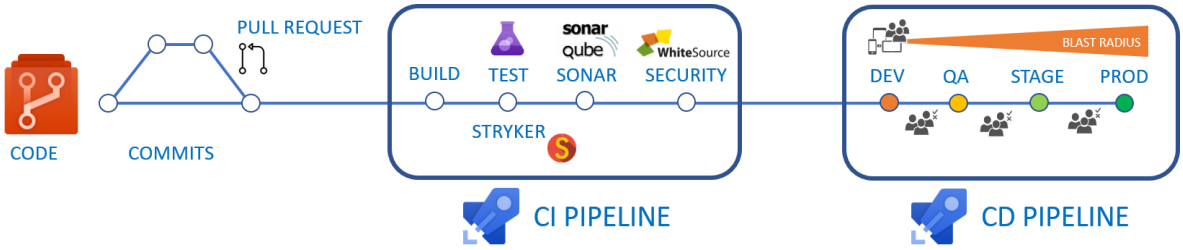
- We have seen that technical debt adds a set of problems during development and makes it much more difficult to add additional customer value.
- Having technical debt in a project saps productivity, frustrates development teams, makes code both hard to understand and fragile, increases the time to make changes, and to validate those changes. Unplanned work frequently gets in the way of planned work.
- Longer term, it also saps the organization's strength. Technical debt tends to creep up on an organization. It starts small, and grows over time. Every time a quick hack is made, or testing is circumvented because changes needed to be rushed through, the problem grows worse and worse. Support costs get higher and higher, and invariably, a serious issue arises.
- Eventually, the organization cannot respond to its customers' needs in a timely and cost efficient way.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Automated Measurement for Monitoring

One key way to minimize the constant acquisition of technical debt, is to use automated testing and assessment.

SonarCloud is a cloud service offered by SonarSource and based on SonarQube. SonarQube is a widely adopted open source platform to inspect continuously the quality of source code and detect bugs, vulnerabilities and code smells in more than 20 different languages.



Managing Technical Debt with Azure DevOps and SonarCloud

Step1: Sonar Project and Security Token

1. Go to https://sonarcloud.io and Login with Azure DevOps account
2. MyProjects → Analyze new project → Import a new organization
    a. Azure DevOps organization name = <Name of your DevOps Organization>
    b. Personal Access Token: <Generated PAT Token in Azure DevOps Portal>
    c. Continue
    d. Select Free plan → Create Organization
3. Select Your Azure DevOps Project in which we want to use Sonar Cloud. → Setup
4. Select With Azure DevOps Pipelines (Recommended)

Step 2: Add a new SonarCloud Service Endpoint in DevOps Project

1. Azure DevOps → Project → **Marketplace** → **Browse Marketplace**.
2. Search for **"SonarCloud"** → Select the **SonarCloud** → . . . → Proceed to Organization
3. Go to **Project settings > Service connections**
4. Add a new service connection of the type SonarCloud
5. Use this token: ce3983468d6d765fc74dd2923bd86221b5bf6f3d
6. Click on **Verify** to check that everything is linked correctly.

Step3: Integrating a build with SonarCloud

1. Create a **New Pipeline (Classic)** and use the template **.NET Core with SonarCloud. This will add 3 steps related to Sonar Cloud**
    a. Prepare analysis on Sonar Cloud
        i. Select the SonarCloud endpoint.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

ii. Select the SonarCloud organization DevOpsDemoOrg

iii. In Choose the way to run the analysis, select **Integrate with MSBuild**.

iv. In the Project Key field, enter DevOpsDemoOrg_HelloWorldApp

v. In the Project Name field, enter HelloWorldApp

b. Run Code Analysis: This task needs to run after your build step.

c. Publish Quality Gate Result*

i. This task is not mandatory but will allow you to decorate your Pull Request.

ii. If you plan not to use such a feature, you can omit it. Be aware that this task may increase your build time.

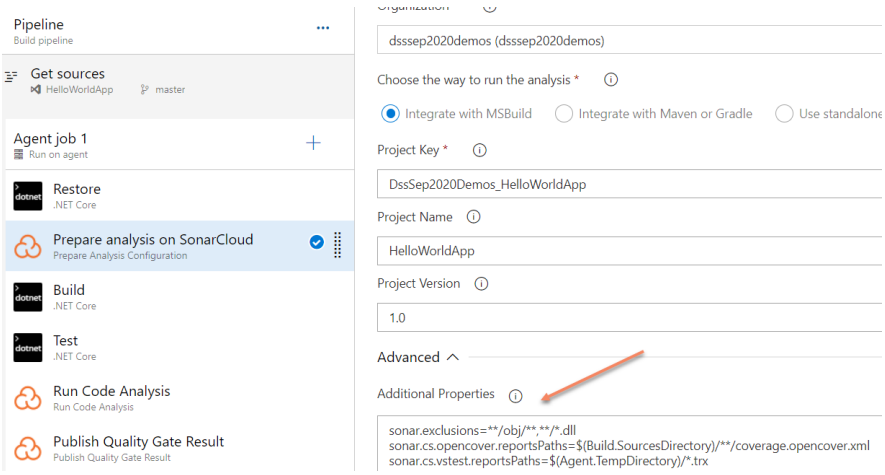
2. Use **Windows Agent for the Job**

3. Add the following to Additional Properties Section of Prepare analysis on SonarCloud

```
sonar.exclusions=**/obj/**,**/*.dll
sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/coverage.opencover.xml
sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
```

4. Add the following arguments to the Test task

```
--configuration $(BuildConfiguration) /p:CollectCoverage=true /p:CoverletOutputFormat=opencover --logger trx
```



Step 4: Configuring Sonar Qube

Goto Sonar Cloud Website → Select Project → Administration → New Code → Select Previous Version

Step 5: Run the Pipeline and review the report in the Sonar Cloud Website

1. From the left panel, select the **Run Code Analysis** task. This contains the processes where SonarCloud analyzes the code.
2. Near the end of the log, locate the URL to the results viewer and open it.

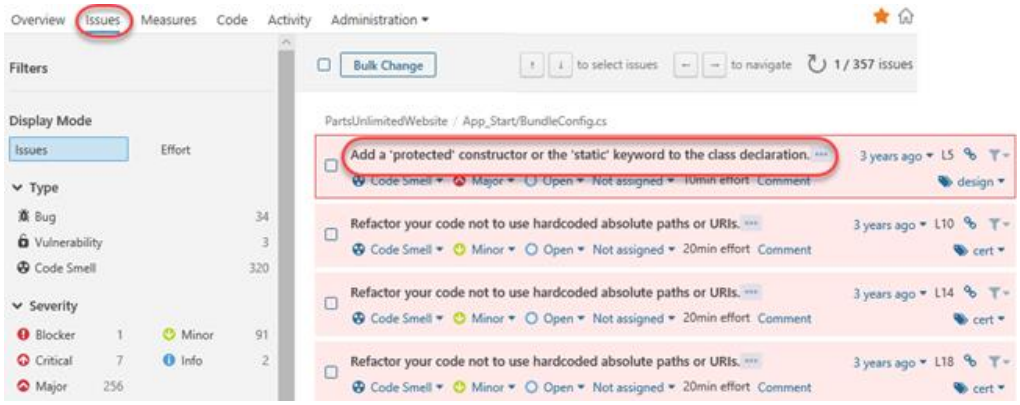
Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
161  2018-09-05T17:50:51.2673999Z WARNING: WARN: Too many duplication references on file PartsUnlimited-aspnet45/src/PartsUnlimitedWebsi
162  2018-09-05T17:50:51.3947493Z WARNING: WARN: Too many duplication references on file PartsUnlimited-aspnet45/src/PartsUnlimitedWebsi
163  2018-09-05T17:50:51.4633714Z INFO: CPD calculation finished
164  2018-09-05T17:50:51.8708252Z INFO: Analysis report generated in 390ms, dir size=6 MB
165  2018-09-05T17:50:52.3440983Z INFO: Analysis reports compressed in 468ms, zip size=1 MB
166  2018-09-05T17:50:53.8844203Z INFO: Analysis report uploaded in 1554ms
167  2018-09-05T17:50:53.8958997Z INFO: ANALYSIS SUCCESSFUL, you can brows https://sonarcloud.io/dashboard?id=partsunlimited.johndoe
168  2018-09-05T17:50:53.8959382Z INFO: Note that you will be able to access the updated dashboard once the server has processed the sub
169  2018-09-05T17:50:53.8959767Z INFO: More about the report processing at https://sonarcloud.io/api/ce/task?id=AWWq28N_HZ_RHZXc9nPS
```

3. Open the URL in new browser tab.

Terms	Description
Bugs	An issue that represents something wrong in the code. If this has not broken yet, it will, and probably at the worst possible moment. This needs to be fixed
Vulnerabilities	A security-related issue which represents a potential backdoor for attackers
Code Smells	A maintainability-related issue in the code. Leaving it as-is means that at best maintainers will have a harder time than they should make changes to the code. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes
Coverage	To determine what proportion of your project's code is actually being tested by tests such as unit tests, code coverage is used. To guard effectively against bugs, these tests should exercise or 'cover' a large proportion of your code
Duplications	The duplications decoration shows which parts of the source code are duplicated
Size	Provides the count of lines of code within the project including the number of statements, Functions, Classes, Files and Directories

4. Select the **Issues** tab. This provides a convenient way to filter and sort the results so that you can attack the section you feel needs immediate attention.



5. The code view provides an in-depth review of each issue, along with suggestions and configuration options.

   For this issue, select **Open | Resolve as won't fix**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

6. Select the **Measures** tab. This provides a visualization of issues as selected by the available filters.
7. Filter down to see the **Reliability | Overview**. This enables you to hover over the various assets to see the amount of effort required to fix and/or maintain various components for reliability.

8. Select the **Code** tab and drill into the project. This provides a way to review project issues at a file level.
9. Open the file with Bugs and see the issues.
10. Expand the **Administration** option. Note that there is an incredible amount of flexibility available here for customizing your SonarCloud analysis.

Set up Pull request Validation Integration

- A SonarCloud project needs to be provided with an access token so it can add PR comments to Azure DevOps, and
- Branch Policy needs to be configured in Azure DevOps to trigger the PR build

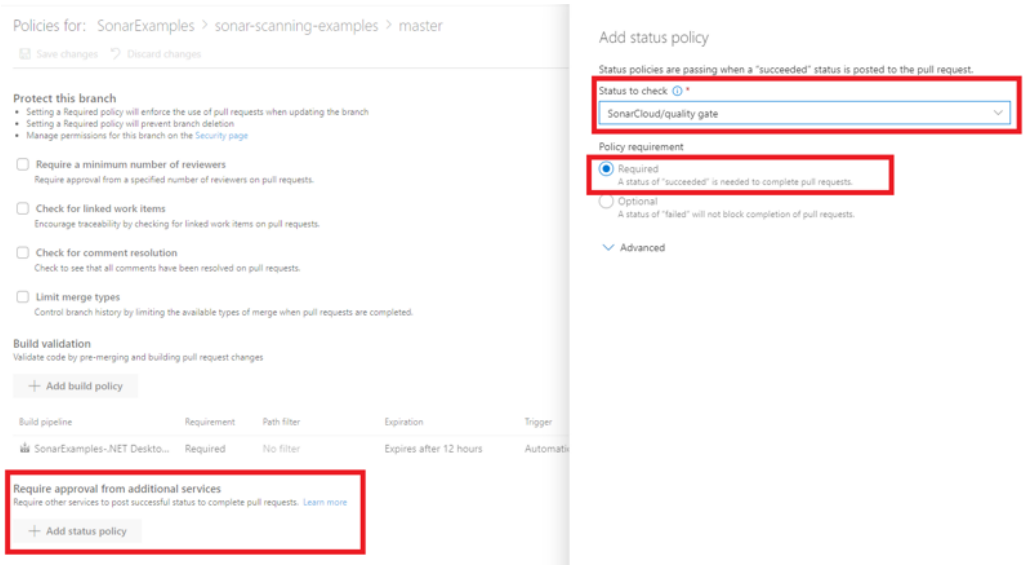
1. Configure Sonar Cloud to be able to post comments in Pull Request.
    a. **Azure DevOps Site**: Create a **Personal Access Token** in Azure DevOps with **Code (read and write)** scope.
    b. Sonar Cloud Site → Select the Project → **Administration** → General Settings → Pull Requests → Provider **= Azure DevOps Services**, Personal Access Token = <Paste from previous Step>.

Note: This is required for posting comments in Pull Request.

2. Configure the branch policy for the project in Azure DevOps
    a. Repos → Branches → . . . → Branch policies
    b. Click **Add Build Policy . . . → Display name = Sonar Cloud analysis → Save**
3. Create a Pull Request
    a. Create a New Branch
    b. Edit some code in New Branch
    c. Create a Pull Request from New Branch to Master
4. If the pull request integration is correctly configured the UI will show that.
5. Complete the Pull Request and Note the pipeline has Run and you will be able to complete the pull request only if Pipeline is completed successfully.

Part2:

6. Block pull requests if the Code Quality check failed
    a. Return to the **Master Branch Policy** page → Click **Add status policy**
    b. Select **SonarCloud/quality gate** from the **Status to check** drop-down
    c. Set the **Policy requirement** to **Required** → Click **Save**



7. Create a New Branch
8. Edit Program.cs and add the following inside the class.

```
public void Unused()
{
}
```

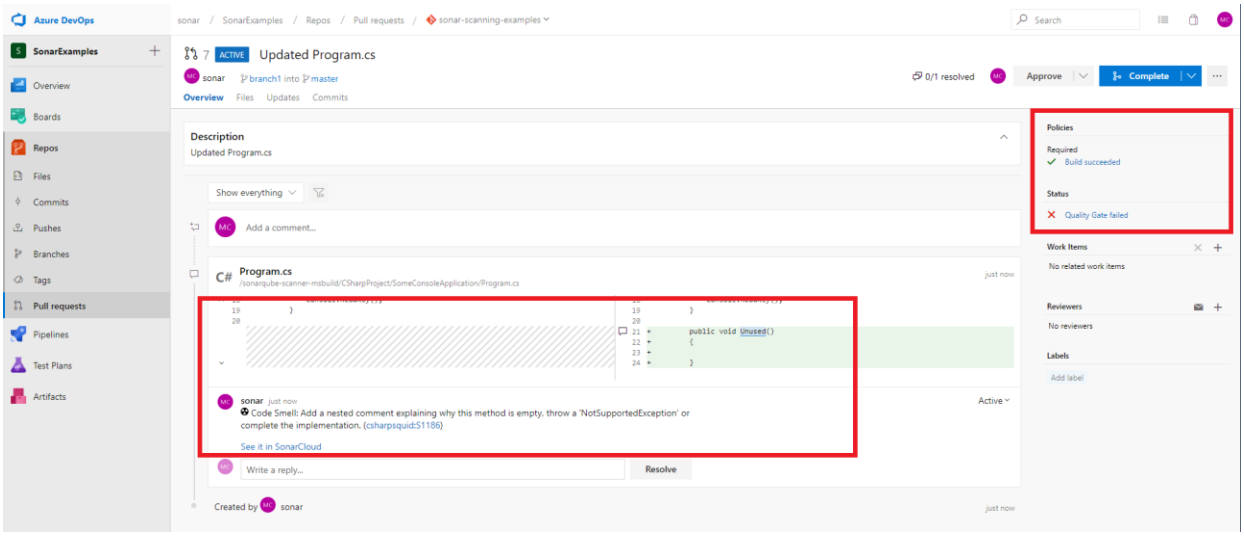
9. Save and Commit
10. Create a Pull Request and note that this time the Build Succeeds but **Gate is failed**.

Note that the only issues in code that was changed or added in the pull request are reported - pre-existing issues in **Program.cs** and other files are ignored.

11. Users will now be unable to merge the pull request until the Code Quality check is successful, either because all of the issues have been fixed or the issues have been marked as **confirmed** or **resolved** in SonarCloud.

Check the SonarCloud Quality Gate status in a Continuous Deployment scenario

12. Review the results of the Pull Request analysis



Sonar Cloud Tasks in YAML.

```
- task: SonarCloudPrepare@1
  displayName: 'Prepare SonarCloud analysis'
  inputs:
    SonarCloud: 'SonarCloud connection 1'
    organization: '$(SonarOrganization)'
    scannerMode: 'MSBuild'
    projectKey: '$(SonarProjectKey)'
    projectName: '$(SonarProjectName)'
    projectVersion: '$(Build.BuildNumber)'
    extraProperties: |
                sonar.exclusions=**/obj/**,**/*.dll
                sonar.cs.opencover.reportsPaths=$(Build.SourcesDirectory)/**/coverage.opencover.xml
                sonar.cs.vstest.reportsPaths=$(Agent.TempDirectory)/*.trx
```

```
//.....Task for Build and Test
- script: dotnet build --configuration $(buildConfiguration)
  displayName: 'dotnet build $(buildConfiguration)'

- script: dotnet test --configuration $(BuildConfiguration) /p:CollectCoverage=true
/p:CoverletOutputFormat=opencover --logger trx'
  displayName: 'dotnet test'

- task: SonarCloudAnalyze@1
  displayName: 'Run SonarCloud code analysis'

- task: SonarCloudPublish@1
  displayName: 'Publish SonarCloud quality gate results'
```

This condition limits scans to only when the build is for a pull request to the master branch. condition: |

```
condition: |
and
(
  succeeded(),
  eq(variables['Build.Reason'], 'PullRequest'),
  eq(variables['System.PullRequest.TargetBranch'], 'master')
)
```

Note: SonarCloud extension is not available for Azure DevOps Server. You can use SonarQube which is an onpremise version for code analysis which can be integrated with both Azure DevOps services and Server.

Managing technical debt with SonarQube and Azure DevOps

https://azuredevopslabs.com/labs/vstsextend/sonarqube/

Scan open-source components using WhiteSource Bolt

Open-source software (OSS) is a type of computer software in which source code is released under a license in which the copyright holder grants users the rights to study, change, and distribute the software to anyone and for any purpose

Corporate Concerns with Open-Source Software Components:

- **Are of low quality**: This would impact maintainability, reliability, and performance of the overall solution

- **Have no active maintenance**: The code would not evolve over time or be alterable without making a copy of the source code, effectively forking away from the origin
- **Contain malicious code**: The entire system that includes and uses the code will be compromised. Potentially the entire company's IT and infrastructure is affected
- **Have security vulnerabilities**: The security of a software system is as good as its weakest part. Using source code with vulnerabilities makes the entire system susceptible to attack by hackers and misuse
- **Have** unfavorable **licensing restrictions:** The effect of a license can affect the entire solution that uses the open-source software.

Important Notes about OSSC:

- It's important to have an **inventory of the open-source components** that your project uses.
- It's important to understand which **libraries** have **vulnerability** issues, when those vulnerabilities were addressed, and what versions you can use. Based on this information, you might even choose to use a different library or write your own.
- You also need to understand what **licenses** these libraries use. Some licenses require you to make public any code that uses that library if you've made changes to the library's code. This requirement is problematic when your source code is not open source as well. You'll need to check with your own legal team to determine which licenses you can use.

WhiteSource is the leader in continuous open source **software security and compliance management**. WhiteSource integrates into your build process, irrespective of your programming languages, build tools, or development environments. It works automatically, continuously, and silently in the background, **checking the security, licensing, and quality** of your **open source components** against WhiteSource constantly-updated definitive database of open source repositories.

Azure DevOps integration with WhiteSource Bolt will enable you to:

1. Detect and remedy vulnerable open source components.
2. Generate comprehensive open source inventory reports per project or build.
3. Enforce open source license compliance, including dependencies' licenses.
4. Identify **outdated** open source libraries with recommendations to update.

Walkthrough

1. From Gallery install WhiteSource Bolt
2. Pipelines → **WhiteSource Bolt** tab → You see a form that asks for your email address and company name.
3. Enter Work Email = <Your Email Id>, Company Name and Country → Get Started

You will get a message: **You are using a FREE version of WhiteSource Bolt**

4. Add the following step to the end of the YAML file

```
- task: WhiteSource Bolt@20
  displayName: 'Run WhiteSource Bolt'
```

5. Save and Run the Build Pipeline

The following comprehensive reports and dashboards will be generated automatically:

- Security vulnerabilities dashboard
- Security vulnerabilities report
- Outdated libraries report
- License risks and compliance dashboard
- Inventory report

6. When the build completes, navigate to the **WhiteSource Bolt Build Report** tab.

Other Vulnerability Tools for OSS Code Scanning

Micro Focus Fortify:

- Can add build tasks to CI/CD pipeline to help identify vulnerabilities in your source code
- Provides a comprehensive set of software security analyzers
- Fortify Static Code Analyzer: Identifies root causes of software security vulnerabilities
- Fortify on Demand: Automatically submits static and dynamic scan requests.

Checkmarx:

Identifies, tracks, and fixes technical and logical security flaws:

- Best fix location
- Quick and accurate scanning
- Incremental scanning
- Seamless integration
- Code portfolio protection
- Easy to initiate Open-Source Analysis with *Checkmarx Open-Source Analysis (CxOSA)* and *Checkmarx Static Application Security Testing (CxSAST)*

Static versus Dynamic Application Security Testing.

- Static application security testing takes place while the application is not running and identifies flaws, vulnerabilities, back doors in the code while in that state. Static analysis can take place early in the dev cycle and help identify potential issues early on.

- Dynamic application security testing takes place while the code is running, it will monitor memory usage, functional behaviours, performance, responses times and other items.

Veracode:

- Integrate application security into your development tools.
- Don't stop for false alarms.
- Align your application security practices with your development practices.
- Don't just find vulnerabilities, fix them.
- Onboarding process allows for scanning on day one.

Secure DevOps Kit for Azure (AzSK)

https://github.com/azsk/DevOpsKit-docs

AzSK is a collection of scripts, tools, extensions, automations

- Helps secure the subscription
- Enables secure development
- Integrates security into CI/CD
- Provides continuous assurance
- Assists with alerts & monitoring
- Governing cloud risks

Azure Governance

Azure Policy

- Is a service in Azure that you use to create, assign, and manage policies
- Provides enforcement by using policies and initiatives
- Runs evaluations on your resources and scans for those not compliant
- Comes with several built-in policy and initiative definitions

An example of an Azure policy that you can integrate with your DevOps pipeline is the Check Gate task

Azure Security Center

- Provide security recommendations
- Monitor security settings
- Monitor all your services
- Use Azure Machine Learning
- Analyze and identify potential attacks
- Supports Windows and Linux operating systems

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Scenario 1:

Many organizations learn how to respond to security incidents only after suffering an attack. To reduce costs and damage, it's important to have an incident response plan in place before an attack occurs. You can use Azure Security Center in different stages of an incident response.

You can use Security Center during the detect, assess, and diagnose stages.

- **Detect:** Review the first indication of an event investigation. Example: Use the Security Center dashboard to review the initial verification that a high-priority security alert was raised.
- **Assess:** Perform the initial assessment to obtain more information about the suspicious activity. Example: Obtain more information about the security alert.
- **Diagnose:** Conduct a technical investigation and identify containment, mitigation, and workaround strategies. Example: Follow the remediation steps described by Security Center in that particular security alert.

Scenario 2:

You can reduce the chances of a significant security event by configuring a security policy, and then implementing the recommendations provided by Azure Security Center.

- A *security policy* defines the set of controls that are recommended for resources within that specified subscription or resource group. In Security Center, you define policies according to your company's security requirements.
- Security Center analyzes the security state of your Azure resources. When Security Center identifies potential security vulnerabilities, it creates recommendations based on the controls set in the security policy. The recommendations guide you through the process of configuring the needed security controls

Resource Locks

Prevent accidental deletion or modification of your Azure resources.

Locks help you prevent accidental deletion or modification of your Azure resources. You can manage these locks from within the Azure portal. To view, add, or delete locks, go to the SETTINGS section of any resource's settings blade. You may need to lock a subscription, resource group, or resource to prevent other users in your organization from accidentally deleting or modifying critical resources. You can set the lock level to CanNotDelete or ReadOnly.

Azure Blueprints:

Allows for the definition of a repeatable set of Azure resources that implement and adhere to an organization's standards, patterns, and requirements

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Azure Advanced Thread Protection (ATP)

- **Azure ATP portal.** Azure ATP has its own portal, through which you can monitor and respond to suspicious activity. The portal allows you to create your Azure ATP instance, and view the data received from Azure ATP sensors.

You can also use the portal to monitor, manage, and investigate threats in your network environment. You sign in to the Azure ATP portal at https://portal.atp.azure.com. However, note that you must sign in with a user account that is assigned to an Azure AD security group that has access to the Azure ATP portal.

- **Azure ATP sensor**. Azure ATP sensors are installed directly on your domain controllers. The sensor monitors domain controller traffic without requiring a dedicated server or configured port mirroring.
- **Azure ATP cloud service.** Azure ATP cloud service runs on the Azure infrastructure, and is currently deployed in the United States, Europe, and Asia. The cloud service is connected to Microsoft Intelligent Security Graph

Agenda: UI Test using Selenium

- Overview about Functional Tests
- Kind of Functional Tests
- UI Test with Selenium on Local System
- UI Tests in Build and Release Pipeline
- Capture Video

Overview about Functional Tests

What are functional and nonfunctional tests?

- ***Functional tests*** verify that each function of the software does what it should. How the software implements each function isn't important in these tests. What's important is that the software behaves correctly. You provide an input and check that the output is what you expect.
- ***Nonfunctional tests*** check characteristics like performance and reliability. An example of a nonfunctional test is checking to see how many people can sign in to the app simultaneously. Load testing is another example of a nonfunctional test.

Kind of functional tests:

1. Smoke testing

Smoke testing verifies the most basic functionality of your application or service. These tests are often run before more complete and exhaustive tests. Smoke tests should run quickly.

For example, say you're developing a website. Your smoke test might use curl to verify that the site is reachable and that fetching the home page produces a 200 (OK) HTTP status. If fetching the home page produces another status code, such as 404 (Not Found) or 500 (Internal Server Error), you know that the website isn't working. You also know that there's no reason to run other tests. Instead, you diagnose the error, fix it, and restart your tests.

2. Unit testing

unit testing verifies the most fundamental components of your program or library, such as an individual function or method. You specify one or more inputs along with the expected results. The test runner performs each test and checks to see whether the actual results match the expected results.

3. Integration testing

Integration testing verifies that multiple software components work together to form a complete system. For example, an e-commerce system might include a website, a products database, and a payment system. You might write an integration test that adds items to the shopping cart and then purchases the items. The test verifies that the web application can connect to the products database and then fulfill the order.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

4. Regression testing

A *regression* occurs when existing behavior either changes or breaks after you add or change a feature. *Regression testing* helps determine whether code, configuration, or other changes affect the software's overall behavior.

Regression testing is important because a change in one component can affect the behavior of another component. For example, say you optimize a database for write performance. The read performance of that database, which is handled by another component, might unexpectedly drop. The drop in read performance is a regression.

5. Sanity testing

Sanity testing involves testing each major component of a piece of software to verify that the software appears to be working and can undergo more thorough testing. You can think of sanity tests as being less thorough than regression tests or unit tests. But sanity tests are broader than smoke tests.

Although sanity testing can be automated, it's often done manually in response to a feature change or a bug fix. For example, a software tester who is validating a bug fix might also verify that other features are working by entering some typical values. If the software appears to be working as expected, it can then go through a more thorough test pass.

6. User Interface Testing

User interface (UI) testing verifies the behavior of an application's user interface. UI tests help verify that the sequence, or order, of user interactions leads to the expected result. These tests also help verify that input devices, such as the keyboard or mouse, affect the user interface properly. You can run UI tests to verify the behavior of a native Windows, macOS, or Linux application. Or you can use UI tests to verify that the UI behaves as expected across web browsers.

A unit test or integration test might verify that the UI *receives* data correctly. But UI testing helps verify that the user interface ***displays*** correctly and that the result functions as expected for the user.

For example, a UI test might verify that the correct animation appears in response to a button click. A second test might verify that the same animation appears correctly when the window is resized.

What tools I can use to write UI tests?

- **Windows Application Driver** (WinAppDriver): WinAppDriver helps you automate UI tests on Windows apps. These apps can be written in Universal Windows Platform (UWP) or Windows Forms (WinForms).
- **SpecFlow**: SpecFlow is for .NET projects. It's inspired by a tool called Cucumber. Both SpecFlow and Cucumber support behavior-driven development (BDD). BDD uses a natural-language parser called Gherkin to help both technical teams and nontechnical teams define business rules and requirements. You can combine either SpecFlow or Cucumber with Selenium to build UI tests.

- **Selenium**: Selenium is a portable open-source software-testing framework for web applications. It runs on most operating systems, and it supports all modern browsers. You can write Selenium tests in several programming languages, including C#. In fact, there are NuGet packages that make it easy to run Selenium as NUnit tests. We already use NUnit for our unit tests.

UI Test with Selenium

The Windows agent that's hosted by Microsoft is already set up to run Selenium tests.

What is Selenium

- Selenium Is an **open source tool** which provides support for automating web applications.
- Selenium supports the following:
    - **Languages**: Java, C#, Python, ruby, Perl, PHP.
    - **Operating Systems**: Windows, Linux, Mac etc.
    - **Browsers**: Chrome, Firefox, Safari, Edge, IE etc.

Components of Selenium:

Selenium is an API having group of components

1. Selenium IDE
2. Selenium RC
3. Selenium Web driver
4. Selenium Grid

What are locators in Selenium?

In a Selenium test, a *locator* selects an HTML element from the DOM (Document Object Model) to act on.

Think of the DOM as a tree or graph representation of an HTML document. Each node in the DOM represents a part of the document.

In a Selenium test, you can locate an HTML element by its:

- id attribute.
- name attribute.
- XPath expression.
- Link text or partial link text.
- Tag name, such as body or h1.
- CSS class name.
- CSS selector.

UI Testing with Selenium on Local System

SeleniumTestSolution

MyWebApp

NUnitTestProject1

Create the Main Web Application

1. Create a New **ASP.NET Core Web Application: MyMathWebApp** in Solution **SeleniumDemoSolution**
2. Add the following MyMath class to the Models Folder **(/Models/MyMath.cs)**

```
public class MyMath
{
    public int N1 { get; set; }
    public int N2 { get; set; }
    public int Result { get; set; }
}
```

3. Add the following to **/Controllers/HomeController.cs – Replace the existing Index method with below code.**

```
public IActionResult Index()
{
    MyMath m = new MyMath() { N1 = 10, N2 = 5, Result = 15 };
    return View(m);
}
[HttpPost]
public IActionResult Index(MyMath m)
{
    ModelState.Clear();
    m.Result = m.N1 + m.N2;
    return View(m);
}
```

4. Replace the content of **/Views/Home/Index.cshtml** with the following

```
@model MyMath
<div>
    <form method="post">
        N1: <input type="text" id="n1" asp-for="N1" /><br />
        N2: <input type="text" id="n2" asp-for="N2" /><br />
        Result: <input type="text" id="result" asp-for="Result"/><br />
        <input type="submit" name="submit" value="Add" /><br />
    </form>
</div>
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Add a Unit Test Project

5. Right Click on Solution → Add → New Project
6. Select **NUnit Test Project** (.NET Core) → Next
7. Project Name = "**MyMathWebApp.UITests**" → Create
8. Add the following two drivers to the project:
    a. Download Microsoft Edge Web Driver from Microsoft Edge Driver - Microsoft Edge Developer and rename the file to **MicrosoftWebDriver.exe.**
    b. Copy Chrome Driver from ChromeDriver - WebDriver for Chrome - Downloads (chromium.org) and file name should be **chromedriver.exe**

    Note: The version of Chrome and Edge Browser on your local machine should match to the version of Drivers you download from above URL's
9. Add the following **NuGet Packages**.
    a. Selenium.WebDriver
    b. Selenium. Support
    c. Selenium.WebDriver.ChromeDriver
    d. Selenium.WebDriver.MSEdgeDriver
    e. Selenium.Firefox.WebDriver

OR

Add the following to **MyMathWebApp.UITests.csproj**

```
<Project Sdk="Microsoft.NET.Sdk">
 <PropertyGroup>
  <TargetFramework>net5.0</TargetFramework>
  <IsPackable>false</IsPackable>
 </PropertyGroup>

 <ItemGroup>
  <PackageReference Include="NUnit" Version="3.12.0" />
  <PackageReference Include="NUnit3TestAdapter" Version="3.16.1" />
   <PackageReference Include="Newtonsoft.Json" Version="13.0.1" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.5.0" />
  <PackageReference Include="Selenium.Support" Version="3.141.0" />
  <PackageReference Include="Selenium.WebDriver" Version="3.141.0" />
  <PackageReference Include="Selenium.WebDriver.ChromeDriver" Version="93.0.4577.1500" />
  <PackageReference Include="Selenium.WebDriver.MSEdgeDriver" Version="93.0.967" />
 </ItemGroup>

 <ItemGroup>
```

```
  <None Update="chromedriver.exe">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </None>
  <None Update="MicrosoftWebDriver.exe">
    <CopyToOutputDirectory>Always</CopyToOutputDirectory>
  </None>
 </ItemGroup>
</Project>
```

Note: The version of ChromeDriver and MSEdigeDriver must be same and match the version of Drivers downloaded.

10. Add the following class to the project

```
using NUnit.Framework;
using OpenQA.Selenium;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium.Edge;
using System;
using System.Threading;

[TestFixture("Chrome")]
[TestFixture("Edge")]
//[TestFixture("Firefox")]
public class IndexPageTest
{
   private string browser;
   private IWebDriver driver;
   public IndexPageTest(string browser)
   {
      this.browser = browser;
   }

   [OneTimeSetUp]
   public void Init()
   {
      var cwd = Environment.CurrentDirectory;
      // Create the driver for the current browser.
      switch (browser)
```

```
    {
        case "Chrome":
            driver = new ChromeDriver(cwd);
            break;
        case "Edge":
            driver = new EdgeDriver(cwd);
            break;
        default:
            throw new ArgumentException($"'{browser}': Unknown browser");
    }
}

[OneTimeTearDown]
public void Cleanup()
{
    driver.Quit();
}
private void ClickElement(IWebElement element)
{
    // We expect the driver to implement IJavaScriptExecutor.
    // IJavaScriptExecutor enables us to execute JavaScript code during the tests.
    IJavaScriptExecutor js = driver as IJavaScriptExecutor;
    // Through JavaScript, run the click() method on the underlying HTML object.
    js.ExecuteScript("arguments[0].click();", element);
}
[Test]
public void AddTest()
{
    if (driver == null)
    {
        Assert.Ignore();
        return;
    }
    driver.Manage().Window.Maximize();
    driver.Url = "https://localhost:44344/";// Environment.GetEnvironmentVariable("SITE_URL");
    IWebElement n1 = driver.FindElement(By.Name("N1"));
    IWebElement n2 = driver.FindElement(By.Name("N2"));
```

```
        n1.Clear();
        n2.Clear();
        n1.SendKeys("10");
        n2.SendKeys("3");

        ClickElement(driver.FindElement(By.Name("submit")));

        IWebElement result = driver.FindElement(By.Name("Result"));
        Assert.AreEqual(result.GetAttribute("value"), "13");
        Thread.Sleep(3000);
    }
}
```

Run the Test

11. Run the Web Application
12. Using Visual Studio - Run the Unit Test using Test Explorer

    OR

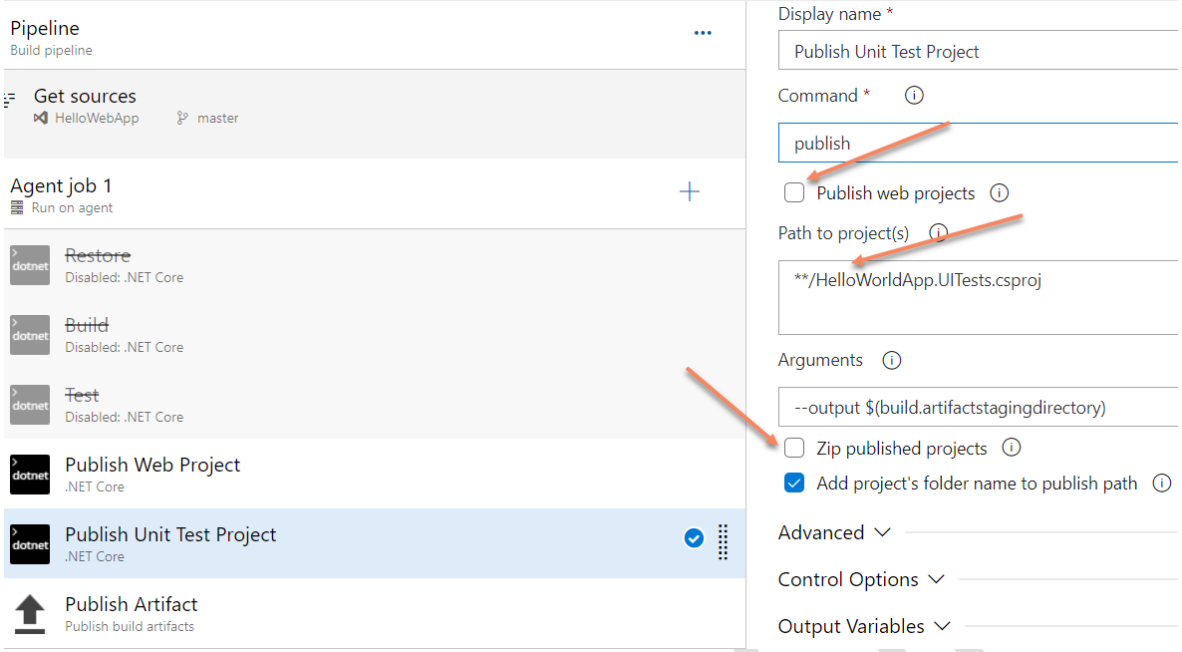
    dotnet test --configuration Release MyMathUITest

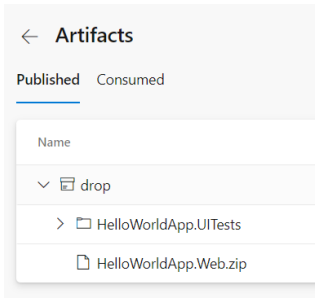
UI Tests in Pipeline

Move the Project to Azure Repos

1. Add **VisualStudio.gitignore** NuGet Package to SeleniumDemoSolution
    a. Right Click on Solution → Manage NuGet Package for Solution…
    b. Browse Tab → Search **gitignore**
    c. Select both project and Install
2. In Solution Folder execute the following commands
    a. git init
    b. git add .
    c. git commit -m "Initial Commit"
3. Azure DevOps → Create a New Project **SeleniumDemoSolution**
    a. Azure Repos → Files → Copy Command to Set Origin and Add
4. Execute the command from local folder to push the project to Repository

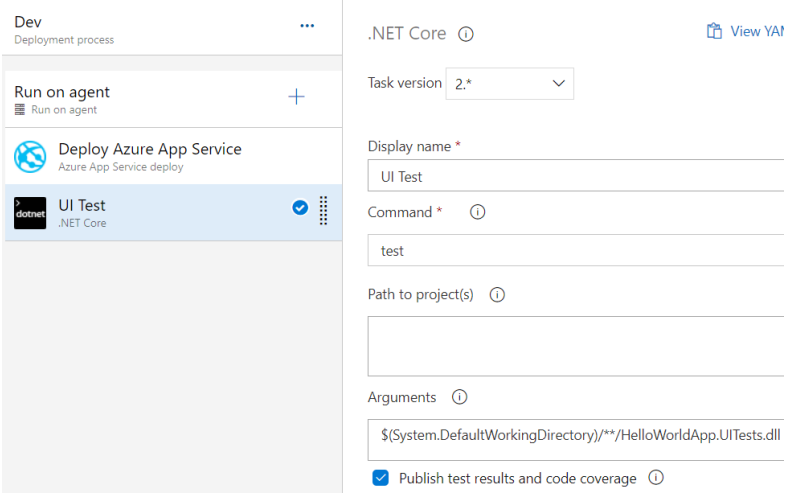
Create a Build Pipeline



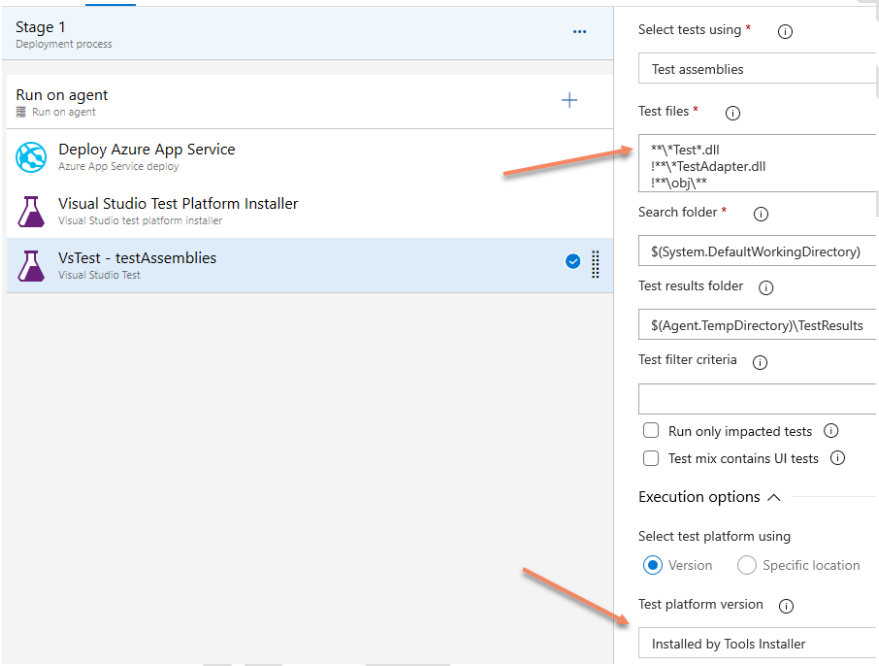


Create a Release Pipeline

1. In Azure Portal, Create an App Service – Web App, Name=MyMathWebApp
2. In DevOps Portal Create a New Service Connection to Azure DevOps
3. Create a New Release Pipeline
   a. Change the Agent to "**windows-2019**"
   b. For Deploy Azure App Service Task: Set Package or folder =
   $(System.DefaultWorkingDirectory)**\HelloWorldApp.UITests.dll)
   (Don't mention ** in the path above)
   c. Add Steps **Visual Studio Test Platform Installer** and **Visual Studio Test**



OR For .NET Framework 4.8 (Older Version)



Note: We can also use **.NET Core Task** with command as Test.

4. Variables Tab → Add Variable SITE_URL = https://<demosite>.azurewebsites.net
5. Create a Release

Note: You will now get an error - The file D:\a\r1\a_HelloWorldApp-ASP.NET Core-CI\drop\HelloWorldApp.UITests\chromedriver.exe does not exist…

6. In Vistual Studio UITest Project → Copy chromedriver.exe (from its output directory)
7. In Solution Explorer → Select chromedriver.exe → F4 (properties) → **Copy to output directory = Copy Always**
8. Push the changes to Azure Repo.
9. Run - Build and Release Pipelines
10. Look at the Test Results in Test Tab (takes couple of minutes to populate)

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Capture Video

If you use the Visual Studio test task to run tests, video of the test can be captured and is automatically available as an attachment to the test result. For this, you must configure the video data collector in a **.runsettings** file and this file must be specified in the task settings.

1. To test project add the file **test.runsettings**

```
<?xml version="1.0" encoding="utf-8" ?>
<RunSettings>
 <DataCollectionRunSettings>
  <DataCollectors>
   <DataCollector uri="datacollector://microsoft/VideoRecorder/1.0"
assemblyQualifiedName="Microsoft.VisualStudio.TestTools.DataCollection.VideoRecorder.VideoRecorderData
Collector, Microsoft.VisualStudio.TestTools.DataCollection.VideoRecorder, Version=15.0.0.0, Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" friendlyName="Screen and Voice Recorder">
     <!--Video data collector was introduced in Visual Studio 2017 version 15.5 -->
     <Configuration>
      <!-- Set "sendRecordedMediaForPassedTestCase" to "false" to add video attachments to failed tests only
-->
      <MediaRecorder sendRecordedMediaForPassedTestCase="true"  xmlns="">
       <ScreenCaptureVideo bitRate="512" frameRate="2" quality="20" />
      </MediaRecorder>
     </Configuration>
   </DataCollector>
  </DataCollectors>
 </DataCollectionRunSettings>
</RunSettings>
```

2. Select **test.runsettings** → Properties → **Copy to output directory = "Copy always"**
3. Visual Studio → Test → Select Settings File → Browse to **test.runsettings**
4. Run the Test and View the Video of UI Test.

5. Wait for Build Pipeline to complete.
6. Edit Release Pipeline → VsTest – testAssemblies → **Settings file** = (browse and visit the file)
   $(System.DefaultWorkingDirectory/**/HelloWorldApp.UITests/test.runsettings
7. Create Release
8. Goto **Tests Tab** → Expant Outcomes and Check Passed → Select Passed Test → Attachments Tab → Select **ScreenCapture.wmv** file → Download and Play

Automating Selenium Tests in Azure Pipelines

https://azuredevopslabs.com/labs/vstsextend/selenium/

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Integrating Quality Tests in Pipeline

- Add Unit Tests to your Application
- Integrating Unit Test with CI Pipeline
- Add the Test Widget to Dashboard
- Perform Code Coverage Testing using Cobertura
- Filtering Tasks based on branch being built

Add Unit Tests to your application

1. Add the following to **HelloWorldApp.Web** Application

MathOperations.cs

```
public class MathOperations
{
   public double Add(double num1, double num2)
   {
      return num1 + num2;
   }
   public double Subtract(double num1, double num2)
   {
      return num1 - num2;
   }
   public double Divide(double num1, double num2)
   {
      return num1 / num2;
   }
   public double Multiply(double num1, double num2)
   {
      // To trace error while testing, writing + operator instead of * operator.
      return num1 + num2;
   }
}
```

2. Add a Test Project to the Solution

```
dotnet new nunit --name HelloWorldApp.Tests
dotnet sln HelloWorldApp.sln add HelloWorldApp.Tests\HelloWorldApp.Tests.csproj
dotnet add HelloWorldApp.Tests\HelloWorldApp.Tests.csproj reference
HelloWorldApp.Web\HelloWorldApp.Web.csproj
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

3. Add the following class to Test Project.

```
using NUnit.Framework;
using HelloWorldApp;

public class MathOperationsTests
{
   [Test()]
   public void AddTest()
   {
      MathOperations bm = new MathOperations();
      double res = bm.Add(10, 10);
      Assert.AreEqual(res, 20);
   }

   [Test()]
   public void SubtractTest()
   {
      MathOperations bm = new MathOperations();
      double res = bm.Subtract(10, 10);
      Assert.AreEqual(res, 0);
   }

   [Test()]
   public void DivideTest()
   {
      MathOperations bm = new MathOperations();
      double res = bm.Divide(10, 5);
      Assert.AreEqual(res, 2);
   }

   [Test()]
   public void MultiplyTest()
   {
      MathOperations bm = new MathOperations();
      double res = bm.Multiply(10, 5);
```

```
        Assert.AreEqual(res, 50);
    }
}
```

4. Add the following Task to YAML

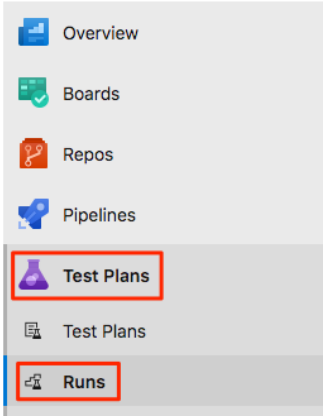
```
- task: DotNetCoreCLI@2
  displayName: 'Run unit tests - $(buildConfiguration)'
  inputs:
    command: 'test'
    arguments: '--no-build --configuration $(buildConfiguration)'
    publishTestResults: true
    projects: '**/*Tests.csproj'
```

Updated YAML file will as below

```
trigger:
- none

pool:
  vmImage: 'vs2017-win2016'

variables:
  buildConfiguration: 'Release'
  dotnetsdk: 3.1.x

steps:
- task: DotNetCoreCLI@2
  displayName: 'Restore project dependencies'
  inputs:
    command: 'restore'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
    arguments: '--configuration $(buildConfiguration)'
```

```
- task: DotNetCoreCLI@2
  displayName: 'Run unit tests - $(buildConfiguration)'
  inputs:
    command: 'test'
    arguments: '--no-build --configuration $(buildConfiguration)'
    publishTestResults: true
    projects: '**/*Tests.csproj'

- task: DotNetCoreCLI@2
 displayName: 'Publish the project - Release'
 inputs:
   command: 'publish'
   projects: '**/*.csproj'
   publishWebProjects: false
   arguments: '--no-build --configuration Release --output $(Build.ArtifactStagingDirectory)/Release'
   zipAfterPublish: true

- task: PublishBuildArtifacts@1
 displayName: 'Publish Artifact: drop'
 condition: succeeded()
```

5. Navigate back to the pipeline summary.
6. Move to the **Tests** tab.
7. In Azure DevOps, select **Test Plans**, and then select **Runs**.



You see the most recent test runs, including the one you just ran.

8. Double-click the most recent test run.

You see a summary of the results.

Add the widget to the Dashboard

1. In your Azure DevOps project, select **Overview**, and then select **Dashboards**.

   Note

   If you ran the template to create the Azure DevOps project, you won't see the dashboard widgets you set up in previous modules.

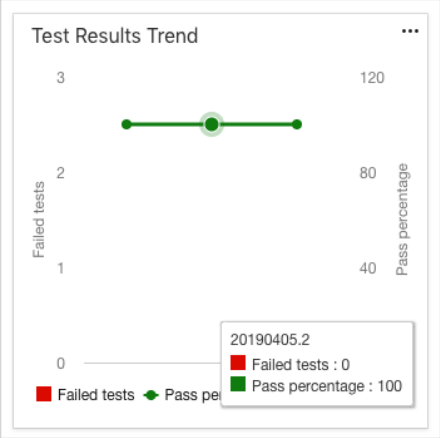
2. Select **Edit**.
3. In the **Add Widget** pane, search for **Test Results Trend**.
4. Drag **Test Results Trend** to the canvas.
5. Select the gear icon to configure the widget.

   a. Under **Build pipeline**, select your pipeline.

   b. Keep the other default settings.

6. Select **Save**.
7. Select **Done Editing**.

Although the widget displays only one test run, you now have a way to visualize and track test runs over time. Here's an example that shows a few successful test runs.



If you begin to see test failures, you can click a point on the graph to navigate directly to that build.

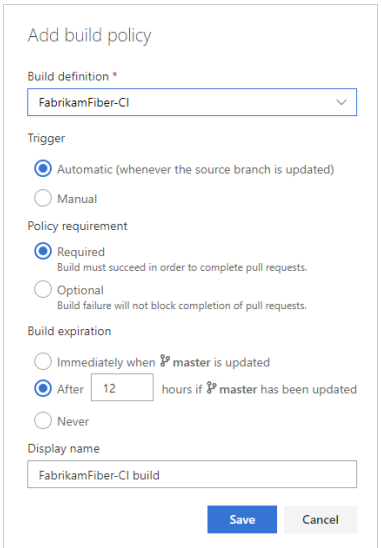
Validate Pull Request based on Build Pipeline result

- Set a policy requiring changes in a pull request to build successfully with the protected branch before the pull request can be completed. Build policies reduce breaks and keep your test results passing.
- Build policies help even if you're using continuous integration (CI) on your development branches to catch problems early.
- If a build validation policy is enabled, a new build is queued when either a new pull request is created, or if changes are pushed to an existing pull request targeting the branch. The build policy then evaluates the results of the build to determine whether the pull request can be completed.

Note that the steps required are different for Azure Repos and GitHub Projects

Steps for Azure Repos and TFS Repositories

1. Azure Repos → Branches → Select a branch → **Branch Policies** → Build Validation →
2. Choose **Add build policy** and configure your options in **Add build policy**.



1. Select the **Build definition**.
2. Choose the type of **Trigger**. Select **Automatic (whenever the source branch is updated)** or **Manual**.
3. Select the **Policy requirement**. If you choose **Required**, builds must complete successfully to complete pull requests. Choose **Optional** to provide a notification of the build failure but still allow pull requests to complete.
4. Set a **Build expiration** to make sure that updates to your protected branch don't break changes for open pull requests.
   - **Immediately when <branch name> is updated**: This option sets the build policy status in a pull request to ***failed*** when the protected branch is updated. Requeue a build to refresh the build status. This setting ensures that the changes in pull requests build successfully even as the protected branch changes. This option is best for teams that have important branches with a lower volume of changes. Teams working in busy development branches may find it disruptive to wait for a build to complete every time the protected branch is updated.
   - **After n hours if branch name has been updated**: This option expires the current policy status when the protected branch updates if the passing build is older than the threshold entered. This option is a compromise between always requiring a build when the protected branch updates and never requiring one. This choice is excellent for reducing the number of builds when your protected branch has frequent updates.

- **Never**: Updates to the protected branch don't change the policy status. This value reduces the number of builds for your branch. It can cause problems when closing pull requests that haven't updated recently.

5. Enter an optional **Display name** for this build policy. This name identifies the policy on the **Branch policies** page. If you don't specify a display name, the policy uses the build definition name.
6. Select **Save**.

To Test Policy

1. To test the policy navigate to the **Pull request** menu in Azure Pipelines or TFS.
2. Select **New pull request**. Ensure your topic branch is set to merge into your master branch. Select **create**.
3. Your screen displays the **policy** being executed.
4. Select the **policy name** to examine the build. If the build succeeds your code will be merged to master. If the build fails the merge is blocked.

For GitHub:

Classic Editor:

Edit Pipeline → Select **Triggers**. Enable the **Pull request validation** trigger. Ensure you include the **master branch** under **Branch filters** → Save

OR

YAML Editor:

```
pr:
- master
```

Or

Pipeline → Edit → ". . ." → Triggers → Check **Override the YAML pull request trigger from here**

To Test

1. Edit a file in branch and **Create the pull request**.
2. Navigate back to your build pipeline. A build will be queued or completed for the merge commit of your pull request.

Life Cylce so far:

```
Pull Master
  Create Feature Branch
    Edit Code in Local Feature Branch
                  Push to remote feature branch
                        Create Pull request from Feature to Master
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Validation Pipeline is run (Branch Validation Policy)

If Failure, Wait for corrected code...from Local Feature Branch

If Success, Complete Pull Request.

Start Build Pipeline to produce Artifacts

Start Release Pipeline to Publish Build Artifacts to Target Env.

Perform Code Coverage Testing

- Code coverage tells us how much of our code has associated unit tests.
- **Coverlet** is a cross-platform, code-coverage library for .NET Core.
- We can add in *code coverage*. That will tell us the percentage of our code that has unit tests. We can use a tool called "**coverlet**" to collect coverage information when the tests run.
- Code coverage results are written to an XML file so that they can be processed by another tool. Azure Pipelines supports Cobertura and JaCoCo coverage result formats.

1. **Add the following NuGet Packages in the Test Project**
   - coverlet.msbuild
   - **Microsoft.CodeCoverage**
   - Microsoft.NET.Test.Sdk
   - ReportGenerator.Core - [by Daniel Palme]

Update the .csproj of Unit Test project

```
<ItemGroup>
  <PackageReference Include="coverlet.msbuild" Version="2.9.0">
    <PrivateAssets>all</PrivateAssets>
    <IncludeAssets>runtime; build; native; contentfiles; analyzers; buildtransitive</IncludeAssets>
  </PackageReference>
  <PackageReference Include="Microsoft.CodeCoverage" Version="16.8.0" />
  <PackageReference Include="ReportGenerator.Core" Version="4.7.1" />
  <PackageReference Include="nunit" Version="3.12.0" />
  <PackageReference Include="NUnit3TestAdapter" Version="3.15.1" />
  <PackageReference Include="Microsoft.NET.Test.Sdk" Version="16.4.0" />
</ItemGroup>
```

2. Run the following dotnet test command to run your unit tests and collect code coverage:

```
dotnet test --configuration Release /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura
/p:CoverletOutput=./TestResults/Coverage/ /p:threshold=80 /p:thresholdType=line /p:thresholdStat=total
```

This command resembles the one you ran previously. The /p: flags tell coverlet which code coverage format to use and where to place the results.

As a result of this command, we will have: **coverage.cobertura.xml** (coverage results in Cobertura format).

To convert Cobertura coverage results to a format that's human-readable, they can use a tool called ReportGenerator .

3. Run the following dotnet tool install command to install ReportGenerator**: (THIS IS FOR WINDOWS)**

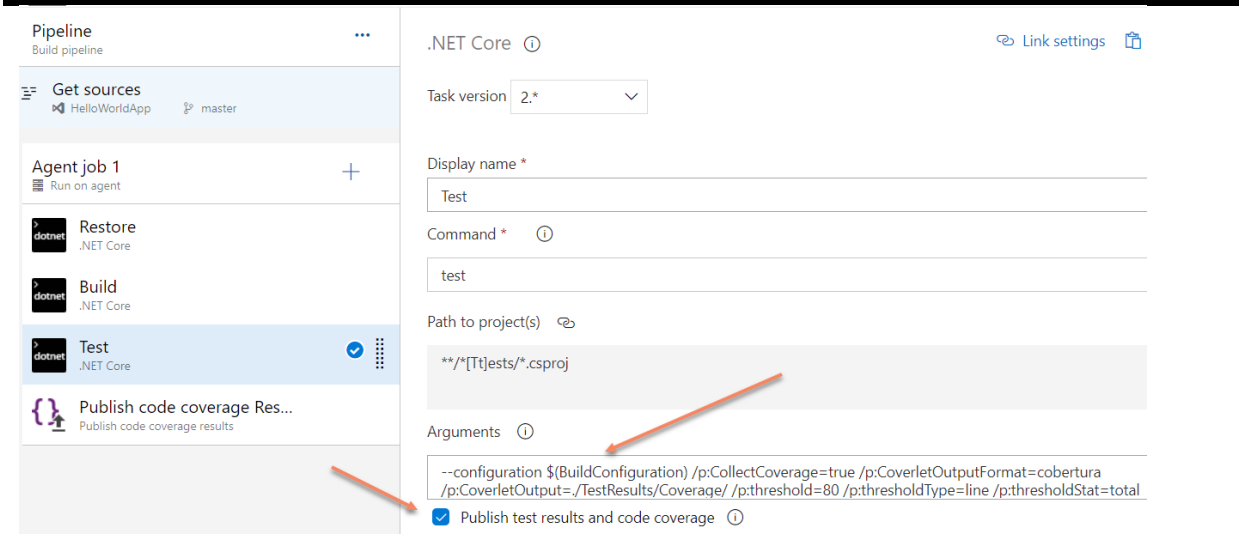
```
dotnet tool install --global dotnet-reportgenerator-globaltool
```

4. Run the following **reportgenerator** command to convert the Cobertura file to HTML:

```
reportgenerator -reports:./HelloWorldApp.Tests/TestResults/Coverage/coverage.cobertura.xml -
targetdir:./CodeCoverage -reporttypes:HtmlInline_AzurePipelines
```

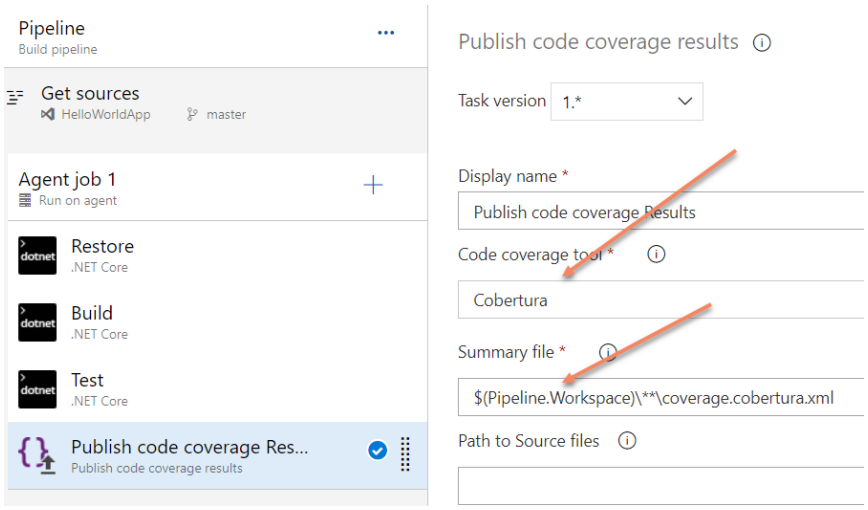
- ReportGenerator provides a number of formats, including HTML. The HTML formats create detailed reports for each class in a .NET project.
- Specifically, there's an HTML format called **HtmlInline_AzurePipelines**, which provides a visual appearance that matches Azure Pipelines.

Classic Pipeline



Arguments:

--configuration $(buildConfiguration) /p:threshold=80 /p:thresholdType=line /p:thresholdStat=total /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./TestResults/Coverage/



YAML Pipeline

```
trigger:
- none

pool:
  vmImage: "windows-latest"

variables:
  buildConfiguration: 'Release'

steps:
- task: DotNetCoreCLI@2
  displayName: 'Restore project dependencies'
  inputs:
    command: 'restore'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  inputs:
    command: 'build'
    projects: '**/*.csproj'
    arguments: '--configuration $(buildConfiguration)'

- task: DotNetCoreCLI@2
  displayName: 'Run unit tests - $(buildConfiguration)'
  inputs:
    command: 'test'
    arguments: '--no-build --configuration $(buildConfiguration) /p:threshold=80 /p:thresholdType=line
/p:thresholdStat=total /p:CollectCoverage=true /p:CoverletOutputFormat=cobertura /p:CoverletOutput=./Test
Results/Coverage/'
```

```
    publishTestResults: true
    projects: '**/*Test.csproj'

  - task: DotNetCoreCLI@2
  - displayName: 'Install ReportGenerator'
  - inputs:
    - command: custom
    - custom: tool
    - arguments: 'install --global dotnet-reportgenerator-globaltool'

  - script: reportgenerator -reports:$(Build.SourcesDirectory)/**/coverage.cobertura.xml -
targetdir:$(Build.SourcesDirectory)/CodeCoverage -reporttypes:HtmlInline_AzurePipelines
  - displayName: 'Create code coverage report'
  - condition: 'succeededOrFailed()'

- task: PublishCodeCoverageResults@1
  displayName: 'Publish code coverage report'
  inputs:
    codeCoverageTool: 'cobertura'
    summaryFileLocation: '$(Pipeline.Workspace)/**/coverage.cobertura.xml'
  condition: 'succeededOrFailed()'
```

Watch Azure Pipelines run the tests

When the build finishes, Navigate back to the **summary page** and select the **Code Coverage** tab.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Manage Pipeline Infrastructure

- About Agent Pools
- About Agents
    - Microsoft Hosted Agents
    - Self Hosted Agents
- Pipeline Demands
- Jobs Deep Dive
    - Deployment Jobs
        - Runonce Deployment Strategy
        - Rolling Deployment Strategy
        - Canary Deployment
    - Agent Pool Jobs
    - Server Jobs
    - Container Jobs
- Build and Release Retention Policies
- About Libraries
    - Variable Groups
    - Secure Files
- Manage Pipeline Resources Permissions
- Pipeline Conditions
- Service Hooks
- Using the REST API

About Agent pools

As you know, a build agent is a piece of installable software that runs one build or deployment job at a time. To build your code or deploy your software you need at least one agent. As you add more code and people, you'll eventually need more than one agent.

About Agent Pools:

- Instead of managing each agent individually, you can organize agents into agent pools.
- In Azure Pipelines, agent pools are scoped to the Azure DevOps organization so you can share an agent pool across projects.
- A project agent pool provides access to an organization agent pool.
- When you create a build or release pipeline, you specify which pool it uses.
- Pools are scoped to your project so you can only use them across build and release pipelines within a project.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- To share an agent pool with multiple projects, in each of those projects, you create a project agent pool **pointing** to an organization agent pool. While multiple pools across projects can use the same organization agent pool, multiple pools within a project cannot use the same organization agent pool. Also, each project agent pool can use only one organization agent pool.

Here are some typical situations when you might want to create self-hosted agent pools:

1. You're a member of a project and you want to use a set of machines owned by your team for running build and deployment jobs only for your projects.
    a. Create Pool at Project Level.
2. You're a member of the infrastructure team and would like to set up a pool of agents for use in all projects.
    a. Create Pool at Organization Level
    b. Select the option to **Auto-provision corresponding agent pools in all projects**
3. You want to share a set of agent machines with multiple projects, but not all of them.
    a. Create Pool at Organization Level.
    b. **Uncheck** the option to **Auto-provision corresponding agent pools in all projects**
    c. Go to each of the other projects, and create a pool in each of them while selecting the option to Use an **existing agent pool** from the organization**.**

Create an Agent Pool

Organization Settings → Pipelines → Agent Pools

Or

Project Settings → Pipelines → Agent Pools

Security of agent pools in organization settings:

1. **Reader:** Members of this role can view the agent pool as well as agents. You typically use this to add operators that are responsible for **monitoring** the agents and their health.
2. **Service Account:** Members of this role can use the organization agent pool to **create a project agent pool in a project**.
3. **Administrator**
    a. In addition to all the above permissions, members of this role can **register or unregister agents from the organization agent pool**.
    b. They can also refer to the organization agent pool when creating a project agent pool in a project.
    c. Finally, they can also **manage membership** for all roles of the organization agent pool. The user that created the organization agent pool is automatically added to the Administrator role for that pool.

Roles for AgentPools in Project Security Settings

1. **Reader**: Members of this role can view the project agent pool. You typically use this to add operators that are responsible for **monitoring** the build and deployment jobs in that project agent pool.
2. **User**: Members of this role can use the project agent pool when authoring pipelines.
3. **Administrator**: In addition to all the above operations, members of this role can **manage membership** for all roles of the project agent pool. The user that created the pool is automatically added to the Administrator role for that pool.

About Build Agents – Microsoft Hosted Agents

MICROSOFT-HOSTED AGENTS

- If your pipelines are in Azure Pipelines, then you've got a convenient option to run your **jobs** using a **Microsoft-hosted agent**. With Microsoft-hosted agents, maintenance and upgrades are taken care of for you. Each time you run a pipeline, you get a fresh virtual machine. The virtual machine is discarded after one use. Microsoft-hosted agents can run jobs directly on the VM or in a container.
- For many teams this is the simplest way to run your jobs. You can try it first and see if it works for your build or deployment. If not, you can use a self-hosted agent.

YAML VM Image Label:

1. **windows-latest** OR windows-2019
2. vs2017-win2016
3. **ubuntu-latest** OR ubuntu-20.04
4. ubuntu-18.04
5. ubuntu-16.04
6. **macOS-latest** OR macOS-10.15
7. macOS-10.14

Pipelines will default to the Microsoft-hosted agent pool. You simply need to specify which virtual machine image you want to use.

```
jobs:
- job: Linux
  pool:
    vmImage: 'ubuntu-latest'
  steps:
  - script: echo hello from Linux
- job: Windows
  pool:
    vmImage: 'windows-latest'
  steps:
  - script: echo hello from Windows
```

Microsoft-hosted agents offer:

- You can add software during your build or release using **tool installer tasks**. Eg: Use .NET Core installs the specified version of .NET SDK on the agent machine.
- Provide at least **10 GB of storage** for your source and build outputs.
- Provide a free tier:
  - **Public project**: **10 free Microsoft-hosted parallel jobs** that can run for up to 360 minutes (6 hours) each time, with no overall time limit per month. Contact us to get your free tier limits increased.
  - **Private project**: **One free parallel job** that can run for up to 60 minutes each time, until you've used 1,800 minutes (30 hours) per month. You can pay for additional capacity per parallel job. Paid parallel jobs remove the monthly time limit and allow you to run each job for up to 360 minutes (6 hours).
- Run on Microsoft Azure general purpose virtual machines **Standard_DS2_v2**
- Run as an **administrator** on Windows and a **passwordless sudo** user on Linux.

Microsoft-hosted agents do not offer:

- The ability to sign in (Remote Desktop Login)
- The ability to drop artifacts to a UNC file share (//servername/drive/folder).
- The ability to run XAML builds. (Xamarin / WPF / UWP Applications)
- Potential performance advantages that you might get by using self-hosted agents which might start and run builds faster.

Image	Classic Editor Agent Specification	YAML VM Image Label	Included Software
Windows Server 2019 with Visual Studio 2019	*windows-2019*	windows-latest OR windows-2019	Link
Windows Server 2016 with Visual Studio 2017	*vs2017-win2016*	vs2017-win2016	Link
Ubuntu 20.04	*ubuntu-20.04*	ubuntu-latest OR ubuntu-20.04	Link
Ubuntu 18.04	*ubuntu-18.04*	ubuntu-18.04	Link
Ubuntu 16.04	*ubuntu-16.04*	ubuntu-16.04	Link
macOS X 10.15	*macOS-10.15*	macOS-latest OR macOS-10.15	Link
macOS X 10.14	*macOS-10.14*	macOS-10.14	Link

Self Hosted Agents

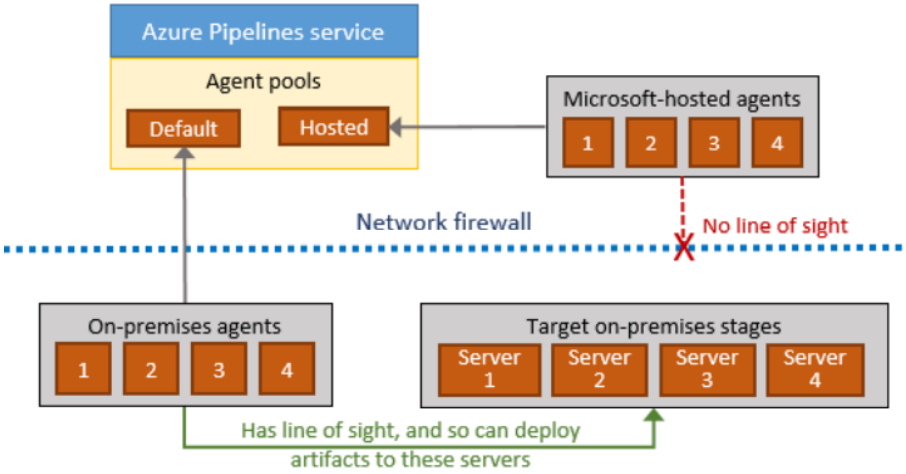
- An agent that you set up and manage on your own to run jobs is a **self-hosted agent**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- Self-hosted agents give you more control to install dependent software needed for your builds and deployments.
- Also, machine-level caches and configuration persist from run to run, which can boost speed.
- You can install the agent on Linux, macOS, or Windows machines. You can also install an agent on a Docker container.

Agents run on a separate server and install and execute actions on a deployment target. By having an agent installed in this way, you need to have permissions to access the target environment. The target environment needs to accept incoming traffic. When you use Hosted agents, running in the cloud, you cannot use these agents to deploy software on your on-premises server.

To overcome this issue, you can run agents in your local network that communicate with the central server



To Create Self-hosted Windows agents

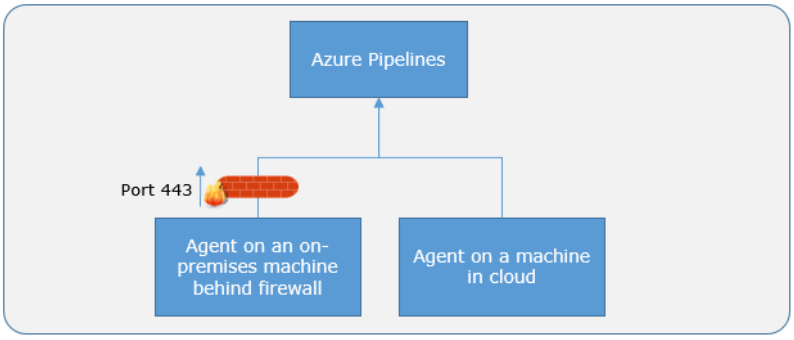
Step1: Make sure your machine has these prerequisites:

- Windows 7, 8.1, or 10 (if using a client OS)
- Windows 2008 R2 SP1 or higher (if using a server OS)
- PowerShell 3.0 or higher
- .NET Framework 4.6.2 or higher

Recommended:

- Visual Studio build tools (2015 or higher)

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



To ensure your organization works with any existing firewall or IP restrictions, ensure that **dev.azure.com** and ***.dev.azure.com** are allowed domains.

In addition, we recommend you **open port 443** to all traffic on these IP addresses and domains.

IPv4 ranges

- 13.107.6.0/24
- 13.107.9.0/24
- 13.107.42.0/24
- 13.107.43.0/24

IPv6 ranges

- 2620:1ec:4::/48
- 2620:1ec:a92::/48
- 2620:1ec:21::/48
- 2620:1ec:22::/48

About Firewall permissions – Read here: https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/v2-windows?view=azure-devops#im-running-a-firewall-and-my-code-is-in-azure-repos-what-urls-does-the-agent-need-to-communicate-with

Step2: Authenticate with a personal access token (PAT)

1. Azure DevOps → **Profile** → Personal access tokens, and then select **+ New Token**.
2. Name your token, **select the organization** where you want to use the token, and then choose a lifespan for your token.
3. Select the **scopes** for this token to authorize for ***your specific tasks***.
4. When you're done, make sure to **copy** the token. You'll use this token as your password.

Step3: Download and configure the agent

1. Log on to the machine using the account for which you've prepared permissions as explained above.
2. In your web browser: **Organization Settings → Agent pools → Default → Agents Tab**
3. **Get the agent** dialog box → Windows Tab → **Download** the agent
4. Open PowerShell **in Administrator mode**

5. Unpack the agent into the directory of your choice.
6. Then run **config.cmd**.

   This will ask you a series of questions to configure the agent.

   a. When setup asks for your server URL, for Azure DevOps Services, answer https://dev.azure.com/{your-organization}.

   b. When setup asks for your authentication type, choose **PAT**. Then paste the PAT token you created into the command prompt window

Step 4: Run the agent

7. **Run interactively**: If you configured the agent to run interactively, to run it:

```
PS c:\agent:>.\run.cmd
```

   Run as a service: If you configured the agent to run as a service, it starts automatically. You can view and control the agent running status from the services snap-in. Run **services.msc** and look for one of:

   - "Azure Pipelines Agent (*name of your agent*)".
   - "VSTS Agent (*name of your agent*)".
   - "vstsagent.(*organization name*).(*name of your agent*)".

To Remove and re-configure an agent:

```
PS c:\agent>.\config.cmd remove
```

More About Agents: https://docs.microsoft.com/en-us/azure/devops/pipelines/agents/agents

Pipeline Demands

Not all agents are the same. We've seen that they can be based on different operating systems, but they can also have different dependencies installed. To describe this, every agent has a set of capabilities configured as name-value pairs. The capabilities such as machine name and type of operating system that are automatically discovered, are referred to as **system capabilities**. The ones that you define are called **user capabilities.**
Some tasks won't run unless one or more demands are met by the agent. For example, the Visual Studio Build task demands that **msbuild** and **visualstudio** are installed on the agent.

Manually entered demands

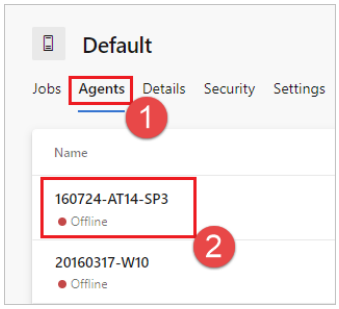
You might need to use self-hosted agents with special capabilities. For example, your pipeline may require **SpecialSoftware** on agents in the Default pool. Or, if you have multiple agents with different operating systems in the same pool, you may have a pipeline that requires a Linux agent.

To add a single demand to your YAML build pipeline, add the demands: line to the pool section.

```
job: Build
pool:
```

```
name: Default
demands:
 - SpecialSoftware # Check if SpecialSoftware capability exists
 - Agent.OS -equals Linux # Check if Agent.OS == Linux
```

1. **Organization Settings** → Agent Pools → Default
2. Switch to Agents Tab → Select the desired agent



3. Choose the **Capabilities** tab → Add a new capability

4. Add something like the following entry:

First box	Second box
SpecialSoftware	C:\Program Files (x86)\SpecialSoftware
Key1	Value1

USER CAPABILITIES

Shows information about user-defined capabilities supported by this host

Add capability

Save changes Undo changes

SYSTEM CAPABILITIES

Shows information about the capabilities provided by this host

Capability name	Capability value
Agent.ComputerName	GREGP50
Agent.HomeDirectory	C:\agent
Agent.Name	GREGP50
Agent.OS	Windows_NT
Agent.OSArchitecture	X64
Agent.OSVersion	10.0.17134
Agent.Version	2.141.2
ALLUSERSPROFILE	C:\ProgramData
APPDATA	C:\Users\Greg\AppData\Roaming
AzurePS	5.7.0
bower	C:\Users\Greg\AppData\Roaming\npm\bower.cmd
Cmd	C:\WINDOWS\system32\cmd.exe
CommonProgramFiles	C:\Program Files\Common Files
CommonProgramFiles(x86)	C:\Program Files (x86)\Common Files
CommonProgramW6432	C:\Program Files\Common Files
COMPUTERNAME	GREGP50

Note: Please read the System capabilities.

Microsoft-hosted agents don't display system capabilities as they have pre-defined list of softwares

https://docs.microsoft.com/en-in/azure/devops/pipelines/agents/hosted

You can specify certain demands that the agent must meet.

Edit Build Pipeline → **Options Tab** → Under Demands → +Add

Jobs Deep Dive

You can organize your pipeline into jobs. Every pipeline has at least one job. A job is a series of steps that run sequentially as a unit. In other words, a job is the smallest unit of work that can be scheduled to run.

Your pipeline may have multiple stages, each with multiple jobs. In that case, use the stages keyword.

```
stages:
- stage: A
 jobs:
 - job: A1
 - job: A2

- stage: B
 jobs:
 - job: B1
 - job: B2
```

Deployment Job: If the primary intent of your job is to deploy your app (as opposed to build or test your app), then you can use a special type of job called **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!
```

Although you can add steps for deployment tasks in a job, we recommend that you instead use a **deployment job**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

A deployment job has a few benefits. For example, you can deploy to an environment, which includes benefits such as being able to see the history of what you've deployed.

To stop downloading artifacts, use **- download: none** or choose specific artifacts to download by specifying Download Pipeline Artifact task.

Deployment strategies

```
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:
        ...
```

LifeCycle Hooks

- **preDeploy**: Used to run steps that initialize resources before application deployment starts.
- **deploy**: Used to run steps that deploy your application. Download artifact task will be auto injected only in the deploy hook for deployment jobs. To stop downloading artifacts, use **- download: none** or choose specific artifacts to download by specifying **Download Pipeline Artifact** task.

- **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 interval.
- **on: failure** or **on: success**: Used to run steps for rollback actions or clean-up.

Deployment Strategy

a) RunOnce Deployment: It's is the simplest deployment strategy wherein all the lifecycle hooks, namely preDeploy deploy, routeTraffic, and postRouteTraffic, are executed once. Then, either on: success or on: failure is executed.

b) Rolling Deployment:

a. All the lifecycle hooks are supported and lifecycle hook jobs are created to run on each VM.

b. You can control the number/percentage of virtual machine targets to deploy to in parallel.

```
strategy:
 rolling:
   maxParallel: [ number or percentage as x% ]
```

c. Currently is only supported for VM resources.

c) Canary Deployment:

a. You can roll out the changes to a small subset of servers first. As you gain more confidence in the new version, you can release it to more servers in your infrastructure and route more traffic to it.

b. 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.

```
strategy:
 canary:
   increments: [ 10,20 ]
```

c. Canary deployment strategy supports the preDeploy lifecycle hook (executed once) and iterates with the deploy, routeTraffic, and postRouteTraffic lifecycle hooks. It then exits with either the success or failure hook.

d. You can only use the canary deployment strategy for Kubernetes resources.

Types of Jobs

- **Agent pool jobs** run on an agent in an agent pool.
- **Server jobs (Agentless jobs)** run on the Azure DevOps Server.
- **Container jobs** run in a container on an agent in an agent pool.

Agent Pool Jobs:

These are the most common type of jobs and they run on an agent in an agent pool. Use demands to specify what capabilities an agent must have to run your job.

```
pool:
  name: myPrivateAgents
  demands:
  - agent.os -equals Windows
  - anotherCapability -equals somethingElse
steps:
- script: echo hello world
```

Server jobs (Agentless jobs)

Tasks in a server job are orchestrated by and executed on the server (Azure Pipelines or TFS). A server job does not require an agent or any target computers. Only a few tasks are supported in a server job at present.

Example: ManualValidation Task

```
jobs:
- job: string
  pool: server
```

Container Jobs

- On Linux and Windows agents, jobs may be run on the host or in a container. (**On macOS and Red Hat Enterprise Linux 6, container jobs are not available**.)
- Containers provide isolation from the host and allow you to pin specific versions of tools and dependencies. Host jobs require less initial setup and infrastructure to maintain.
- You can select the exact versions of operating systems, tools, and dependencies that your build requires.
- When you specify a container in your pipeline, the agent will first fetch and start the container. Then, each step of the job will run inside the container.
- The windows-2019 and ubuntu-16.04 pools support running containers.
- **The Hosted macOS pool does not support running containers.**

The follwing tells the system to fetch the ubuntu image tagged 16.04 from Docker Hub and then start the container.

```
pool:
  vmImage: 'ubuntu-16.04'
container: nginix
steps:
- script: printenv
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Windows Example:

```
pool:
  vmImage: 'windows-2019'
container: mcr.microsoft.com/windows/servercore:ltsc2019
steps:
- script: set
```

Job Dependencies:

When you define multiple jobs in a single stage, you can specify dependencies between them.

Example jobs that build sequentially:

```
jobs:
- job: Debug
  steps:
  - script: echo hello from the Debug build
- job: Release
  dependsOn: Debug
  steps:
  - script: echo hello from the Release build
```

Example jobs that build in parallel (no dependencies):

```
jobs:
- job: Windows
  pool:
    vmImage: 'vs2017-win2016'
  steps:
  - script: echo hello from Windows
- job: macOS
  pool:
    vmImage: 'macOS-10.14'
  steps:
  - script: echo hello from macOS
- job: Linux
  pool:
    vmImage: 'ubuntu-16.04'
  steps:
  - script: echo hello from Linux
```

Job Workspace:

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 **Pipeline.Workspace** variable.

Under this, various sub-directories are created:

- **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.

When you run a pipeline on a self-hosted agent, by default, none of the sub-directories are cleaned* in between two consecutive runs. As a result, you can do **incremental builds and deployments**, provided that tasks are implemented to make use of that.

You can override this behavior using the workspace setting on the job.

```
- job: myJob
  workspace:
    clean: outputs | resources | all # what to clean up before the job runs
```

- outputs: Delete Build.BinariesDirectory before running a new job.
- resources: Delete Build.SourcesDirectory before running a new job.
- all: Delete the entire Pipeline.Workspace directory before running a new job.

*Note: $(Build.ArtifactStagingDirectory) and $(Common.TestResultsDirectory) are **always deleted** and recreated prior to every build regardless of any of these settings.

Build and Release Retention Policies

Build Pipeline => **Run**

Release Pipeline => **Release**

YAML Pipeline => **Run**

Retention policies are used to configure how long **runs and releases** are to be retained by the system.

The primary reasons to delete older runs and releases are to conserve storage and reduce clutter.

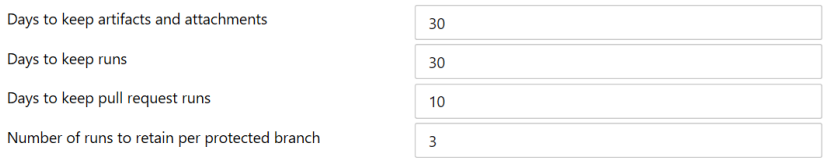
The main reasons to keep runs and releases are for audit and tracking.

The following information is deleted when a run is deleted:

- Logs
- All artifacts
- All symbols
- Binaries
- Test results

- Run metadata

Run retention: Project Settings → Pipelines → Settings

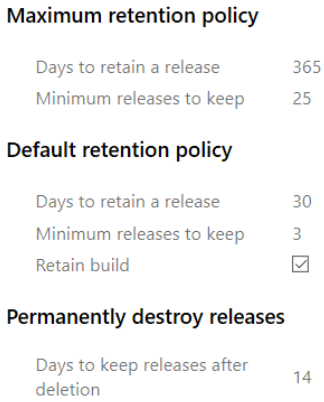


Ranges:

- Artifacts, symbols and attachments range = 1 to 60
- Runs Range = 30 to 731
- Pull request runs Range = 1 to 30
- Number of recent runs to retain per pipeline = 0 to 50

Global Release Retention: Project Settings → Pipelines → Release Retention

- The release retention policies for a release pipeline determine how long a **release and the run linked** to it are retained.
- The retention timer on a release is reset every time a release is modified or deployed to a stage.
- The minimum number of releases to retain setting takes precedence over the number of days.



Note that you can view but not change these settings for your project.

We can override the above settings: Select a Release Pipeline → Edit → Retension Tab → Select Stage…

Interaction between build and release retention

- The build linked to a release has its own retention policy, which may be shorter than that of the release. If you want to retain the build for the same period as the release, set the **Retain associated artifacts** checkbox for the appropriate **stages**. This overrides the retention policy for the build, and ensures that the artifacts are available if you need to redeploy that release.
- When you delete a release pipeline, delete a release, or when the retention policy deletes a release automatically, the retention policy for the associated build will determine when that build is deleted.

About Library

- *Library* is a collection of *shared* build and release assets for a project. Assets defined in a library can be used in multiple build and release pipelines of the project.
- It contains two types of assets:
    1. **variable groups**
    2. **secure files**.

Working with Variables:

Azure Pipelines supports three different ways to dereference variables: **macro, template expression, and runtime expression**. Each syntax can be used for a different purpose and has some limitations.

Syntax	Example	When is it processed?	Where does it expand in a pipeline definition?	How does it render when not found?
macro	`$(var)`	runtime	value (right side)	prints `$(var)`
template expression	`${{ variables.var }}`	compile time	key or value (left or right side)	empty string
runtime expression	`$[variables.var]`	runtime	value (right side)	empty string

Note: If there's no variable by that name, then the macro expression is left unchanged. For example, if $(var) cannot be replaced, $(var) won't be replaced by anything.

Variable Scopes

In the YAML file, you can set a variable at various scopes:

- At the **root level**, to make it available to all jobs in the pipeline
- At the **stage level**, to make it available only to a specific stage
- At the **job level**, to make it available only to a specific job

Variables at the job level override variables at the root and stage level. Variables at the stage level override variables at the root level.

```
variables:
 global_variable: value   # this is available to all jobs


jobs:
- job: job1
 pool:
  vmImage: 'ubuntu-16.04'
 variables:
  job_variable1: value1   # this is only available in job1
```

```
steps:
- bash: echo $(global_variable)
- bash: echo $(job_variable1)
- bash: echo $JOB_VARIABLE1 # variables are available in the script environment too

- job: job2
pool:
  vmImage: 'ubuntu-16.04'
variables:
  job_variable2: value2   # this is only available in job2
steps:
- bash: echo $(global_variable)
- bash: echo $(job_variable2)
- bash: echo $GLOBAL_VARIABLE
```

To Create (Classic) Release Pipeline and Stage Variables

You define and manage these variables in the **Variables** tab in a release pipeline. In the **Release** Pipeline Variables page, open the **Scope drop-down list** and select "**Release**" or the required **stage.**

Note: Store sensitive values in a way that they cannot be seen or changed by users of the release pipelines. Designate a configuration property to be a secure (secret) variable by selecting the (padlock) icon next to the variable.

To reference a variable in YAML, prefix it with a dollar sign and enclose it in parentheses. For example: $(sec)

To use a variable in a script, use environment variable syntax.

Replace . and space with _, capitalize the letters, and then use your platform's syntax for referencing an environment variable. Examples:

Batch script: %SEC%

PowerShell script: ${env:SEC}

Bash script: $(SEC)

Set Secret Variables (YAML)

You should not set secret variables in your YAML file. Instead, you should set them in the pipeline editor using the web interface.

1. Navigate to **YAML file** → Top right click on Vertical … → **Variables** Button.
2. Add or update the variable.
3. Check Keep this value secret to store the variable in an encrypted manner.
4. Save the pipeline.

Secret variables are encrypted at rest with a 2048-bit RSA key.

Secrets are available on the agent for tasks and scripts to use **(so be careful about who has access to alter your pipeline)**.

The following example shows how to use a secret variable called mySecret from a script.

```
steps:
- powershell: |
  # Using an input-macro:
        Write-Host "This works: $(mySecret)"
```

Output:

```
This works: ***
```

Variable Groups

Step 1: Create Variables Groups

1. Pipeline → **Library** → Variable Groups Tab → Click +Variable Group
2. Set Variable group name = "**DemoGroupName**"
3. Add the following Variables
    a) AppServiceName = DssDemoApp
    b) PreProductionSlotName = "PreProd"
    c) ResourceGroupName = "DemoRG"

Step 2.1: Use Variables in Tasks (Classic)

4. Pipelines → Releases → **Edit** Release Pipeline → Select **Production Stage** Tasks (Tasks tab)
5. Ensure that all the tasks you intend to include do not contain any linked parameters. The easy way to do this is to choose **Unlink all** in the settings panel for the entire process.
6. Select Task **Deploy Azure App Service** → Set App Service name = $(AppServiceName), $Resource Group= $(ResourceGroup), Slot=$(PreProductionSlotName)
7. Select Task **Swap Slots**: Set App Service name = $(AppServiceName), $Resource Group= $(ResourceGroup), Source Slot=$(PreProductionSlotName)

Variable Groups in YAML

This example shows how to reference a **variable group** in your YAML file and also add variables within the YAML.

There are two variables used from the variable group: **user** and **token**. The **token variable is secre**t and is mapped to the environment variable $env:MY_MAPPED_TOKEN so that it can be referenced in the YAML.

```
variables:
```

```
  devopsAccount: contoso
```

```
variables:
- group: 'my-var-group' # variable group
- name: 'devopsAccount' # new variable defined in YAML
  value: 'contoso'
- name: 'projectName' # new variable defined in YAML
  value: 'contosoads'

steps:
- task: PowerShell@2
  inputs:
    targetType: 'inline'
    script: |
      Write-Host $(user)
      Write-Host $env:USER
      Write-Host $(env:MY_MAPPED_TOKEN)
  env:
    MY_MAPPED_TOKEN: $(token) # Maps the secret variable $(token) from my-var-group
```

Note: If you use both variables and variable groups, you'll have to use name/value syntax for the individual (non-grouped) variables:

List of Predefined variables

https://docs.microsoft.com/en-us/azure/devops/pipelines/build/variables

Secure Files

- Use the **Secure Files** library to store files such as signing certificates, Apple Provisioning Profiles, Android Keystore files, and SSH keys on the server without having to commit them to your source repository.
- The contents of the secure files are encrypted and can only be used during the build or release pipeline by referencing them from a task.
- The secure files are available across multiple build and release pipelines in the project based on the security settings.
- There's a size limit of 10 MB for each secure file.

Use the **Download Secure File** task to consume secure files within a Build or Release Pipeline.

This example downloads a secure certificate file and installs it to a trusted certificate authority (CA) directory on Linux:

```
- task: DownloadSecureFile@1
  name: caCertificate
  displayName: 'Download CA certificate'
  inputs:
    secureFile: 'myCACertificate.pem'

- script: |
    echo Installing $(caCertificate.secureFilePath) to the trusted CA directory...
    sudo chown root:root $(caCertificate.secureFilePath)
    sudo chmod a+r $(caCertificate.secureFilePath)
    sudo ln -s -t /etc/ssl/certs/ $(caCertificate.secureFilePath)
```

Manage Pipeline Resources Permissions

What is a resource?

A resource is anything used by a pipeline that lives outside the pipeline itself.

Examples include:

- Secure files
- Variable groups
- Service connections
- Agent pools
- Other repositories
- Containers

Resources must be authorized before they can be used. A resource owner controls the **users and pipelines** that can access that resource.

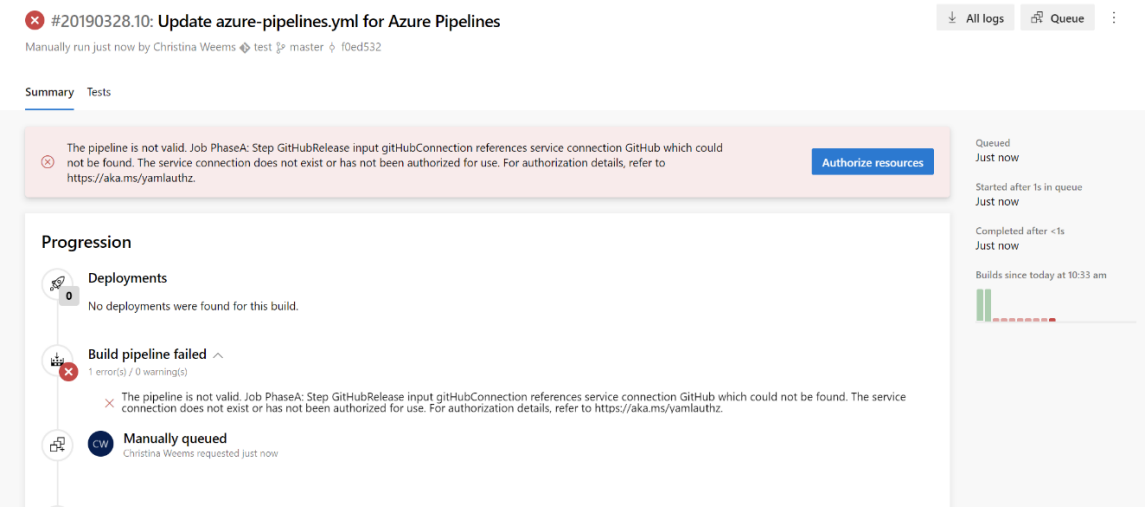
If you create a pipeline with the classic editor, then the **user creating** or **editing** the pipeline must be given **User** role on that resource in order to be able to refer to that resource in the pipeline. A **resource** administrator can add users by navigating to administration page of the corresponding resource and selecting **Security** for that resource.

If you use YAML pipelines, then the pipeline must directly be authorized to use the resource. There are multiple ways to accomplish this.

1. Navigate to the administration experience of the resource. For example, variable groups and secure files are managed in the **Library** page under **Pipelines**. Agent pools and service connections are managed in **Project settings**. Here you can authorize **all pipelines** to be able to access that resource. This is convenient if you do not have a need to restrict access to a resource - for e.g., test resources.

2. When you create a pipeline **for the first time**, all the resources that are referenced in the YAML file are **automatically authorized** for use by the pipeline, provided that you are a member of the **User** role for that resource. So, resources that are referenced in the YAML file at pipeline creation time are automatically authorized.
3. When you **make changes** to the YAML file and add additional resources (assuming that these not authorized for use in all pipelines as explained above), then the build fails with a resource authorization error that is similar to the following: Could not find a <resource> with name <resource-name>. The <resource> does not exist or has not been authorized for use.

   In this case, on the Summary page, you will see an option to authorize the resources on the failed build. If you are a member of the **User** role for the resource, you can select this option. Once the resources are authorized, you can start a new build.



Pipeline Conditions

You can specify the conditions under which each job runs. By default, a job runs if it does not depend on any other job, or if all of the jobs that it depends on have completed and succeeded.

You can customize this behavior by forcing a job to run even if a previous job fails or by specifying a custom condition.

You can specify conditions under which a step, job, or stage will run.

- **succeeded():** It's by default, only when all previous dependencies have succeeded.
- **succeededOrFailed():** Even if a previous dependency has failed, unless the run was canceled.
- **always():** Even if a previous dependency has failed, even if the run was canceled.
- **failed():** Only when a previous dependency has failed.
- Custom conditions

```
jobs:
- job: Foo
```

```
steps:
- script: echo Hello!
  condition: always() # this step will always run, even if the pipeline is canceled

- job: Bar
  dependsOn: Foo
  condition: failed() # this job will only run if Foo fails
```

Custom Condition Examples

- **Run for the master branch, if succeeding:**

  and(succeeded(), **eq**(variables['Build.SourceBranch'], 'refs/heads/master'))
- **Run if the branch is not master, if succeeding**

  and(succeeded(), **ne**(**variables['Build.SourceBranch'], 'refs/heads/master'**))
- **Run for continuous integration (CI) builds if succeeding**

  and(succeeded(), **in**(variables['Build.Reason'], 'IndividualCI', 'BatchedCI'))
- **Run if the build is run by a branch policy for a pull request, if failing**

  and(failed(), eq(variables['Build.Reason'], 'PullRequest'))
- **Run if the build is scheduled, even if failing, even if canceled**

  and(always(), eq(variables['Build.Reason'], 'Schedule'))

Use a template parameter as part of a condition

The script in this YAML file will run because parameters.doThing is false.

```
parameters:
  doThing: false

steps:
- script: echo I did a thing
  condition: and(succeeded(), eq('${{ parameters.doThing }}', false))
```

Use the output variable from a job in a condition in a subsequent job

You can make a variable available to future jobs and specify it in a condition. Variables available to future jobs must be marked as multi-job output variables.

```
jobs:
- job: Foo
  steps:
    - script: |
        echo "This is job Foo."
```

```
    echo "##vso[task.setvariable variable=doThing;isOutput=true]Yes" #The variable doThing is set to true
  name: DetermineResult
- job: Bar
 dependsOn: Foo
 condition: eq(dependencies.Foo.outputs['DetermineResult.doThing'], 'Yes') #map doThing and check if Yes
 steps:
  - script: echo "Job Foo ran and doThing is true."
```

Ref: azure-pipelines-agent/outputvariable.md at master · microsoft/azure-pipelines-agent · GitHub

Service Hooks

One of the first requests many people have when working in a system that performs asynchronous actions is to have the ability to get **notifications or alerts**.

The ability to receive Alerts and notifications is a powerful mechanism to get notified about **certain events** in your system at the moment they happen.

For example, when a build takes a while to complete, probably you do not want to stare to the screen until it has finished. But, you want to know when it does.

Almost every action in the system raises an event to which you can subscribe to. When you have made a subscription, you can then select how you want the notification to be delivered.

Service Hooks:

Service hooks enable you to perform tasks on other services when events happen in your Azure DevOps Services projects. For example, create a card in Trello when a work item is created or send a push notification to your team's mobile devices when a build fails.

In Azure DevOps these Service Hooks are supported Out of the Box:

- **Build and Release:** AppVeyor, Bamboo, Jenkins, MyGet, Clack
- **Collaborate**: Camfire, Flowdock, HipChat, Hubot
- **Customer Support**: UserVoice, Zendesk
- **Plan and track**: Trello
- **Integrate**: Azure Service Bus, Azure Storage, **WebHooks**, Zapier

Note:

If the application that you want to communicate with isn't in the list of available application hooks, you can almost always use the **Web Hooks** option as a generic way to communicate. It allows you to make an **HTTP POST** when an event occurs. So, if for example, you wanted to call an Azure Function or an Azure Logic App, you could use this option.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, INDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

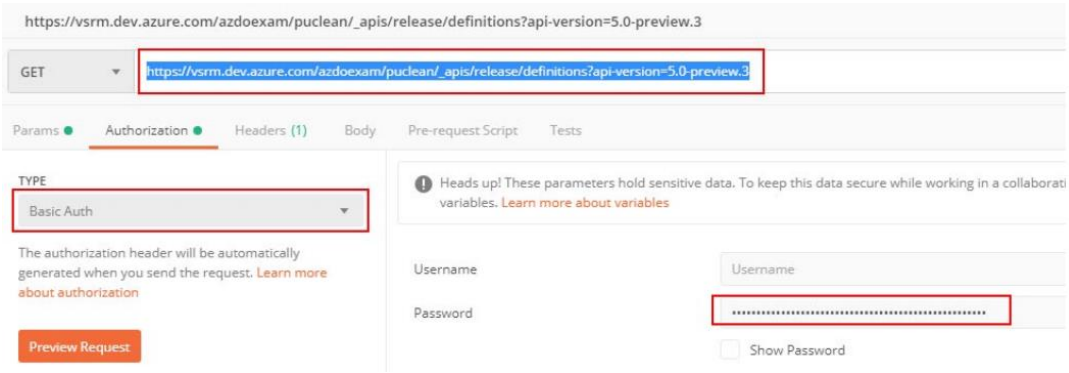
Demo 1: Setting Up Service Hooks to Monitor the Pipeline

Let's now take a look at how a release pipeline can communicate with other services by using service hooks.

1. Goto Project Settings → Service Hooks → **Create Subscription**
2. **Service**: From the list of available applications, click **Azure Storage → Next**
3. **Triggers:** Trigger on this type of event = **Release deployment completed**, then in the Release pipeline name select Release to all environments. For Stage, select Production. Drop down the list for Status and note the available options.
4. **Actions:** Capture the details from Azure Portal and provide the same.
5. Make sure that the test succeeded, then click Close, and on the Action page, click Finish.
6. Execute the Release Pipeline and Note that the message is posted in the queue.

Use the REST API

1. Get a Personal Access Token
2. Use the URL https://dev.azure.com/{organization}/{project}/_apis/release/definitions?api-version=6.1-preview.1 to get all Release Definitions.
3. Download and run **Postman**
4. In the authorization tab, select **Basic Authentication** and paste the **Personal Access token** in the **password** field



5. Press Send and look at the results.

Note: Full REST API Documentation: https://docs.microsoft.com/en-us/rest/api/azure/devops

Agenda: Continuous Deployment using Azure Pipelines

- Deploying App to App Service using YAML
- Add the deployment State to the pipeline
- Deploy Apps to Specific Environment
- Deploy Azure Functions

Deploy ASP.NET Core App to Azure App Service using YAML

Add the deployment stage to the pipeline:

Following step will be added to the YAML file. (Edit WebAppName and use variable created earlier)

```
- task: AzureRmWebAppDeployment@4
  inputs:
   ConnectionType: 'AzureRM'
   azureSubscription: 'AzureDevOpsServicePrincipal'
   appType: 'webAppLinux'
   WebAppName: $(WebAppName)
   packageForLinux: '$(build.artifactstagingdirectory)/**/*.zip'
```

To deploy a build artifact from the pipeline, you need a way to download it from the pipeline to the agent. You use the **`DownloadPipelineArtifact@2`** task to download artifacts.

This task requires a few inputs. The ones we need here are:

- **buildType**, which specifies whether we want the artifacts from the current build or a specific build. For now, we want to deploy the current build.
- **artifact**, which specifies the name of the artifact to download. We need this input to specify the name of the *.zip* file.

```
- task: DownloadPipelineArtifact@2
 inputs:
  buildType: current
  artifact: 'drop'
```

OR (The `download` task is a shortcut for the `DownloadPipelineArtifact@2` task).

```
- download: current
 artifact: drop
```

Multi-Stage YAML Pipeline with Jobs (not deployment jobs) – Doesn't support Approvals and Checks…

```
trigger:
- none
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
stages:
- stage: "Build"
 jobs:
 - job: "BuildJob"
   pool:
    vmImage: 'windows-latest'

   variables:
    buildConfiguration: 'Release'

   steps:
   - task: DotNetCoreCLI@2
    displayName: "Restore Project"
    inputs:
     command: 'restore'
     projects: '**/*.csproj'

   - task: DotNetCoreCLI@2
    displayName : "Build Project"
    inputs:
     command: 'build'
     projects: '**/*.csproj'
     arguments: '-c $(buildConfiguration)'

   - task: DotNetCoreCLI@2
    displayName : "Build Project"
    inputs:
     command: 'publish'
     publishWebProjects: true
     arguments: '-c $(buildConfiguration) -o $(build.artifactstagingdirectory)'

   - task: PublishBuildArtifacts@1
    displayName: 'Publish Artifact'
    inputs:
     PathtoPublish: '$(build.artifactstagingdirectory)'
```

```
- stage: "DeployToDev"
  dependsOn: "Build"
  jobs:
  - job:
    pool:
      vmImage: "windows-latest"
    steps:
    - download: current
      artifact: drop

    - task: AzureRmWebAppDeployment@4
      inputs:
        ConnectionType: 'AzureRM'
        azureSubscription: 'Azure Connection'
        appType: 'webApp'
        WebAppName: 'dssdemo1-dev'
        packageForLinux: '$(Pipeline.Workspace)/**/*.zip'

- stage: "DeployToQA"
  dependsOn: "DeployToDev"
  jobs:
  - job:
    pool:
      vmImage: "windows-latest"
    steps:
    - download: current
      artifact: drop
    - task: AzureRmWebAppDeployment@4
      inputs:
        ConnectionType: 'AzureRM'
        azureSubscription: 'Azure Connection'
        appType: 'webApp'
        WebAppName: 'dssdemo1-qa'
        packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
```

```
- stage: "DeployToProd"
  dependsOn: "DeployToQA"
  jobs:
  - job:
    pool:
      vmImage: "windows-latest"
    steps:
    - download: current
      artifact: drop
    - task: AzureRmWebAppDeployment@4
      inputs:
        ConnectionType: 'AzureRM'
        azureSubscription: 'Azure Connection'
        appType: 'webApp'
        WebAppName: 'dssdemo1-prod'
        deployToSlotOrASE: true
        ResourceGroupName: 'DemoRG'
        SlotName: 'staging'
        packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
    - task: AzureAppServiceManage@0
      inputs:
        azureSubscription: 'Azure Connection'
        Action: 'Swap Slots'
        WebAppName: 'dssdemo1-prod'
        ResourceGroupName: 'DemoRG'
        SourceSlot: 'staging'
```

Deploy a web application to App Service with Approval using Environment

What is an environment?

You've likely used the term *environment* to refer **to where your application or service is running**. It represents a **collection of resources** such as namespaces within Kubernetes clusters, Azure Web Apps, virtual machines, databases, which can be targeted by deployments from a pipeline.

Typical examples of environments include *Dev, Test, QA, Staging and Production.*

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

The advantages of using environments include the following.

- **Deployment history** - Pipeline name and run details are recorded for deployments to an environment and its resources. In the context of multiple pipelines targeting the same environment or resource, deployment history of an environment is useful to identify the source of changes.
- **Traceability of commits and work items** - View jobs within the pipeline run that targeted an environment and the corresponding commits and work items that were newly deployed to the environment. This allows one to track whether a code change (commit) or feature/bug-fix (work items) reached an environment.
- **Permissions** - User permissions and pipeline permissions can be used to secure environments by specifying which users and pipelines are allowed to target an environment.

Resources:

While environment at its core is a grouping of resources, the resources themselves represent actual deployment targets. The **Kubernetes resource** and **virtual machine resource** types are currently supported.

Target Environment from a deployment job

A deployment job can be used to target an entire environment (group of resources) as shown in the following **Sample** YAML snippet.

```
- stage: deploy
  jobs:
  - deployment: DeployWeb
    displayName: Deploy Web App
    pool:
      vmImage: 'Ubuntu-latest'
    # creates an environment if it doesn't exist
    environment: 'demo-dev'
    strategy:
```

Target a specific resource within an environment

```
environment: 'dssdemoapp-dev.WebServer1'
```

Security:

User permissions: Set appropriate user permissions to ensure that the users are pre-authorized to define a pipeline that targets the environment.

1. Environment → Click ⋮ on top right → Security

2. In the **User permissions** blade, click on **+Add** to add a **User or group** and select a suitable **Role**.

Pipeline permissions: Pipeline permissions can be used to authorize either all or specific pipelines for deploying to the environment of concern.

Approvals

You can manually control when a stage should run using approval checks. This is commonly used to control deployments to production environments.

Checks are a mechanism available to the *resource owner* to control if and when a stage in a pipeline can consume a resource. As an owner of a resource, such as an environment, you can define checks that must be satisfied before a stage consuming that resource can start.

Currently, manual approval checks are supported on environments.

```
trigger:
- none

stages:
- stage: 'BuildStage'
 jobs:
 - job: 'BuildJob'
   pool:
     vmImage: windows-latest
   variables:
     buildConfiguration: 'Release'
   steps:
   - task: DotNetCoreCLI@2
     displayName: "Restore"
     inputs:
       command: 'restore'
       projects: '**/*.csproj'
       feedsToUse: 'select'
   - task: DotNetCoreCLI@2
     displayName: "Build"
     inputs:
       command: 'build'
       projects: '**/*.csproj'
   - task: DotNetCoreCLI@2
```

```
    displayName: "Publish"
    inputs:
     command: 'publish'
     publishWebProjects: true
     arguments: '--output $(Build.ArtifactStagingDirectory) --configuration $(buildConfiguration)'
   - task: PublishPipelineArtifact@1
    inputs:
     targetPath: '$(Build.ArtifactStagingDirectory)'
     artifact: 'drop'
     publishLocation: 'pipeline'

- stage: 'DeploytoDev'
 dependsOn: 'BuildStage'
 jobs:
 - deployment:
  pool:
   vmImage: 'ubuntu-latest'
  environment: Development
  strategy:
   runOnce:
    deploy:
     steps:
     - task: AzureRmWebAppDeployment@4
      inputs:
       ConnectionType: 'AzureRM'
       azureSubscription: 'VS Subscription - SS1 - Automatic'
       appType: 'webApp'
       WebAppName: 'dsdemoapp-dev'
       packageForLinux: '$(Pipeline.Workspace)/**/*.zip'

- stage: 'DeploytoQA'
 dependsOn: 'DeploytoDev'
 jobs:
 - deployment:
  environment: QA
  pool:
```

```
    vmImage: 'ubuntu-latest'
  strategy:
   runOnce:
    deploy:
     steps:
     - task: AzureRmWebAppDeployment@4
       inputs:
         ConnectionType: 'AzureRM'
         azureSubscription: 'VS Subscription - SS1 - Automatic'
         appType: 'webApp'
         WebAppName: 'dsdemoapp-qa'
         packageForLinux: '$(Pipeline.Workspace)/**/*.zip'

- stage: 'DeploytoProd'
  dependsOn: 'DeploytoQA'
  jobs:
  - deployment:
    environment: Production
   pool:
    vmImage: 'ubuntu-latest'
   strategy:
    runOnce:
     deploy:
      steps:
      - task: AzureRmWebAppDeployment@4
        inputs:
          ConnectionType: 'AzureRM'
          azureSubscription: 'VS Subscription - SS1 - Automatic'
          appType: 'webApp'
          WebAppName: 'dsdemoapp-prod'
          deployToSlotOrASE: true
          ResourceGroupName: 'Demo-rg'
          SlotName: 'staging'
          packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
      - task: AzureAppServiceManage@0
        inputs:
```

```
        azureSubscription: 'VS Subscription - SS1 - Automatic'
        Action: 'Swap Slots'
        WebAppName: 'dsdemoapp-prod'
        ResourceGroupName: 'Demo-rg'
        SourceSlot: 'staging'
```

Note: Deployment Job doesn't need to have download task as it is builtin by default.

Use the below Task for Application Settings:

```
- task: AzureAppServiceSettings@1
  inputs:
    azureSubscription: 'Azure Connection'
    appName: 'dsshellowebapp'
    resourceGroupName: 'Demo-rg'
    slotName: 'staging'
    appSettings: |
      [
        {
          "name": "key1",
          "value": "valueabcd",
          "slotSetting": false
        },
        {
          "name": "key2",
          "value": "valueefgh",
          "slotSetting": true
        }
      ]
```

Deploy Apps in Virtual Machine

Demo - Environment - Virtual Machine Resource

Step1: Create an Environment

1. Azure DevOps Organization → Project → Pipelines → Environments → New Environment
2. Name = "WebServers", Select Radio Button **Virtual Machines → Next**
3. **Virtual machine resource:** Choose **Operating System = Windows**
4. **Copy PS registration script**.

Note: It is possible to create an empty environment and reference the same from deployment jobs to record the deployment history against the environment.

Step2: Install Pipeline Agent on VM

5. **RDP to VM** → Open Powershell in Administrator mode
6. Paste the PS Script copied in previous step
    a) Tag = Web
    b) User as "dssadmin" (User which has administrator priviledge)

Repeat Step2 for all the VM's you want to Deploy the Application.

Step3: Create a New Pipeline with following YAML

```
trigger:
 - '*'
variables:
 buildConfiguration: 'Release'

stages:
- stage: 'Build'
 displayName: 'Build the web application'
 jobs:
 - job: Build
  displayName: 'Build Job'
  pool:
   vmImage: 'ubuntu-16.04'
  variables:
   wwwrootDir: 'HelloWorldApp.Web/wwwroot'
   dotnetSdkVersion: '3.1.x'

  steps:
  - task: DotNetCoreCLI@2
   displayName: 'Restore project dependencies'
   inputs:
    command: 'restore'
    projects: '**/*.csproj'

  - task: DotNetCoreCLI@2
```

```
    displayName: 'Build the project - $(buildConfiguration)'
    inputs:
      command: 'build'
      arguments: '--no-restore --configuration $(buildConfiguration)'
      projects: '**/*.csproj'

  - task: DotNetCoreCLI@2
    displayName: 'Publish the project - $(buildConfiguration)'
    inputs:
      command: 'publish'
      projects: '**/*.csproj'
      publishWebProjects: false
      arguments: '--no-build --configuration $(buildConfiguration) --
output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)'
      zipAfterPublish: true

  - task: PublishBuildArtifacts@1
    displayName: 'Publish Artifact: drop'
    condition: succeeded()

- stage: 'Deploy'
 displayName: 'Deploy the web application'
 dependsOn: Build
 jobs:
 - deployment: DeployToVM
  pool:
    vmImage: 'windows-latest'
  environment:
    name: WebServers
    resourceType: VirtualMachine
    tags: web
  strategy:
    runOnce:
      deploy:
        steps:
        - task: IISWebAppManagementOnMachineGroup@0
```

```
    inputs:
      EnableIIS: true
      IISDeploymentType: 'IISWebsite'
      ActionIISWebsite: 'CreateOrUpdateWebsite'
      WebsiteName: 'default web site'
      WebsitePhysicalPath: '%SystemDrive%\inetpub\wwwroot'
      WebsitePhysicalPathAuth: 'WebsiteUserPassThrough'
      AddBinding: true
      Protocol: http
      IPAddress: 'All Unassigned'
      port: 80

  - task: IISWebAppDeploymentOnMachineGroup@0
    inputs:
      WebSiteName: 'default web site'
      Package: '$(Pipeline.Workspace)/**/*.zip'
      TakeAppOfflineFlag: true
```

7. Save and Run the Pipeline
8. **View History:** Environment → Click WebServers → Deployments

YAML for Deploying to respective Environment Based on Which Branch Build was Performed

```
trigger:
- master
- development
- release
- hotfix

variables:
  AzureConnection: 'Azure Pass 2'

stages:
- stage: "Build_Stage"
  displayName: "Building the web application"
  jobs:
  - job: 'Build_Job'
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
  displayName: "Job: Building the application"
  pool:
    vmImage: 'ubuntu-latest'
  variables:
    buildConfiguration: 'Release'
    ProjectName: '**/*.csproj'
  steps:
  - task: DotNetCoreCLI@2
    displayName: Restore
    inputs:
      command: 'restore'
      projects: '$(ProjectName)'
      feedsToUse: 'select'
  - task: DotNetCoreCLI@2
    displayName: 'Build Step'
    inputs:
      command: 'build'
      projects: '$(ProjectName)'
      arguments: '--configuration $(buildConfiguration)'
  - task: DotNetCoreCLI@2
    inputs:
      command: 'publish'
      publishWebProjects: true
      arguments: '--configuration $(buildConfiguration) -o $(Build.ArtifactStagingDirectory)'
  - task: PublishBuildArtifacts@1
    inputs:
      PathtoPublish: '$(Build.ArtifactStagingDirectory)'
      ArtifactName: 'drop'
      publishLocation: 'Container'

- stage: "DeployToDev_Stage"
  dependsOn: Build_Stage
  condition: eq(variables['Build.SourceBranch'], 'refs/heads/development')
  jobs:
  - deployment: 'DeployToDev'
    displayName: "Job: Deploying to Dev Stage"
```

```
  pool:
    vmImage: 'ubuntu-latest'
  environment: dev
  strategy:
    runOnce:
      deploy:
        steps:
        - task: AzureRmWebAppDeployment@4
          inputs:
            ConnectionType: 'AzureRM'
            azureSubscription: $(AzureConnection)
            appType: 'webApp'
            WebAppName: 'dssdemoapp-dev'
            packageForLinux: '$(Pipeline.Workspace)/**/*.zip'

- stage: "DeployToQA_Stage"
  dependsOn: Build_Stage
  condition: eq(variables['Build.SourceBranch'], 'refs/heads/release')
  jobs:
  - deployment: 'DeployToQA_Job'
    displayName: "Job: Deploying to QA Stage"
    pool:
      vmImage: 'ubuntu-latest'
    environment: QA
    strategy:
      runOnce:
        deploy:
          steps:
          - task: AzureRmWebAppDeployment@4
            inputs:
              ConnectionType: 'AzureRM'
              azureSubscription: $(AzureConnection)
              appType: 'webApp'
              WebAppName: 'dssdemoapp-qa'
              packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
```

```
- stage: "DeployToProd_Stage"
 dependsOn: Build_Stage
 condition: eq(variables['Build.SourceBranch'], 'refs/heads/master')
 jobs:
 - deployment: 'DeployToProd_Job'
  displayName: "Job: Deploying to Prod Stage"
  pool:
   vmImage: 'ubuntu-latest'
  environment: Prod
  strategy:
   runOnce:
    deploy:
     steps:
     - task: AzureRmWebAppDeployment@4
      inputs:
       ConnectionType: 'AzureRM'
       azureSubscription: '$(AzureConnection)'
       appType: 'webApp'
       WebAppName: 'dssdemoapp-prod'
       deployToSlotOrASE: true
       ResourceGroupName: 'SandeepRG'
       SlotName: 'staging'
       packageForLinux: '$(Pipeline.Workspace)/**/*.zip'

     - task: AzureAppServiceManage@0
      inputs:
       azureSubscription: $(AzureConnection)
       Action: 'Swap Slots'
       WebAppName: 'dssdemoapp-prod'
       ResourceGroupName: 'SandeepRG'
       SourceSlot: 'staging'
```

Deploy Azure Functions

Step1: Azure Portal → Create an Azure Function App

Step2: Azure DevOps → Create an Azure Repo

Step3: Visual Studio → Create an Azure Function App with HTTPTrigger

Step4: Push the Local Repo to Remote Repo

Step5: Use the below YAML File

```
trigger:
 - '*'
variables:
 buildConfiguration: 'Release'

stages:
- stage: 'Build'
 displayName: 'Build the web application'
 jobs:
 - job: Build
  displayName: 'Build Job'
  pool:
   vmImage: 'ubuntu-16.04'
  variables:
   dotnetSdkVersion: '3.1.x'
   steps:
  - task: DotNetCoreCLI@2
   displayName: 'Restore project dependencies'
   inputs:
    command: 'restore'
    projects: '**/*.csproj'

  - task: DotNetCoreCLI@2
   displayName: 'Build the project - $(buildConfiguration)'
   inputs:
    command: 'build'
    arguments: '--no-restore --configuration $(buildConfiguration)'
    projects: '**/*.csproj'

  - task: DotNetCoreCLI@2
   displayName: 'Publish the project - $(buildConfiguration)'
   inputs:
    command: 'publish'
```

```
      projects: '**/*.csproj'
      publishWebProjects: false
      arguments: '--no-build --configuration $(buildConfiguration) --
output $(Build.ArtifactStagingDirectory)/$(buildConfiguration)'
      zipAfterPublish: true

  - task: PublishBuildArtifacts@1
    displayName: 'Publish Artifact: drop'
    condition: succeeded()

- stage: 'Deploy'
 displayName: 'Deploy the azure Function'
 dependsOn: Build
 variables:
  appName: dssdemoafunc123
  azureSubscription: 'Azure Subscription Connection'
 jobs:
 - deployment: Deploy
  pool:
   vmImage: 'windows-latest'
  environment: 'Azure-function'
  strategy:
   runOnce:
    deploy:
     steps:
      - task: AzureFunctionApp@1
        displayName: Azure Function App Deploy
        inputs:
          azureSubscription: 'Azure Subscription Connection'
          appType: 'functionAppLinux'
          appName: '$(appName)'
          package: '$(Pipeline.Workspace)/drop/$(buildConfiguration)/*.zip'
```

Setting up a CI/CD pipeline for Azure Functions

https://azuredevopslabs.com/labs/vstsextend/azurefunctions/

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Azure Pipelines Template Library in GitHub

https://github.com/microsoft/azure-pipelines-yaml/tree/master/templates

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

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
- 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.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

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.

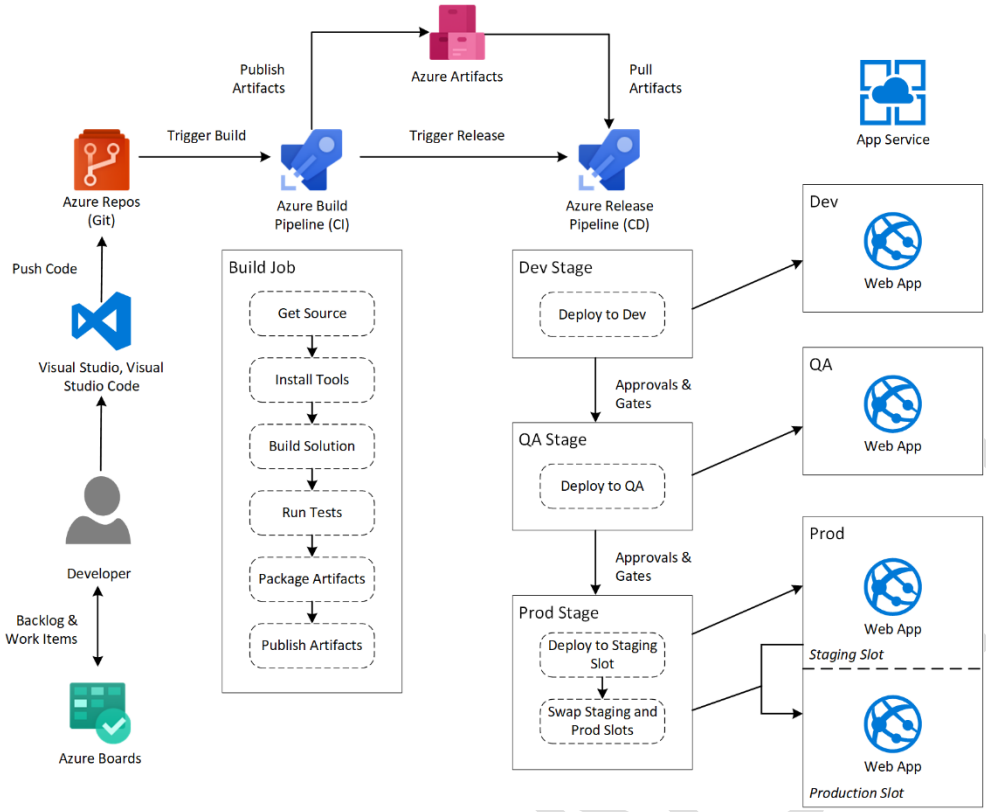
Azure Compute Options:

- 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

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Setup App Services and Connection to Azure Subscription in ADO

Create a Web App in Azure Portal

1. Login to Azure Portal, https://portal.azure.com/
2. Azure Portal → More Services → Web App → + Add
3. Select Web Apps → Create
4. 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.
5. Application Insights=Off
6. Create

Follow the above steps and create the below AppServices

1. Create an App Service = **DssDemoApp-dev**
2. Create an App Service = **DssDemoApp-qa**
3. Create an App Service = **DssDemoApp-prod / 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/Application 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. Seach 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.

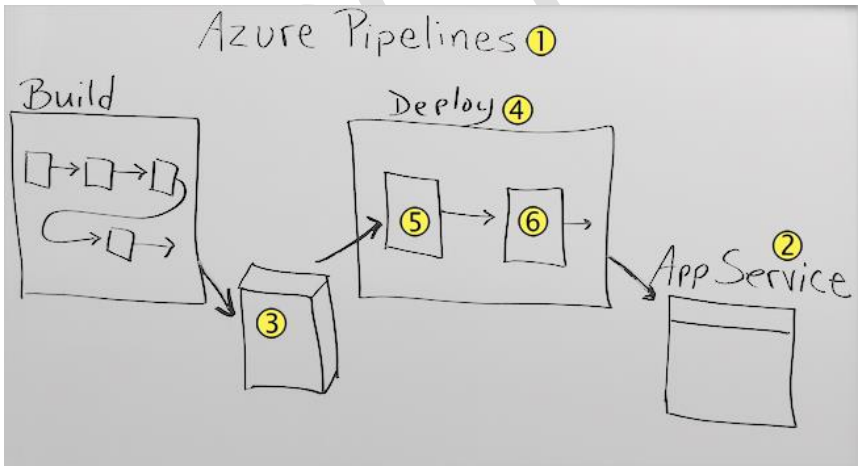
Option3: Create an Azure Resource Manager service connection to a VM with a managed service identity

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**

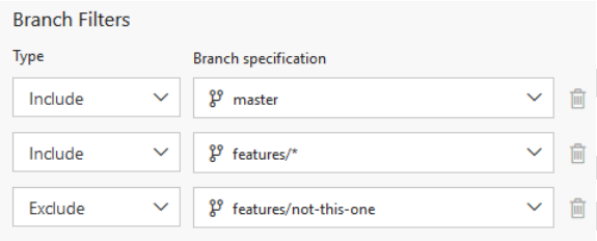
Optional: **Enable Continues deployment triggers:**

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.



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 Deplotment 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
```

OR

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

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

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
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)

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.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

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:

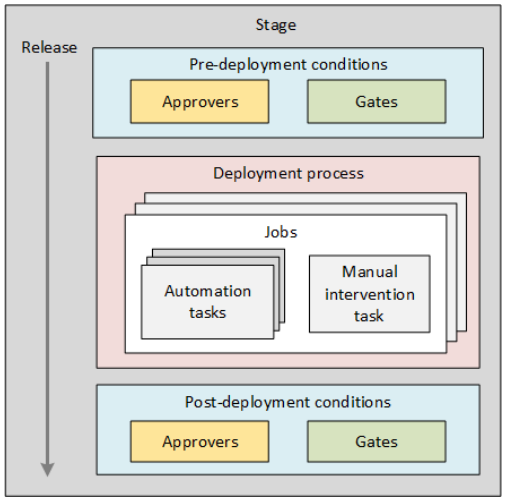
1. Choose **+ Release** and select **Create a release**.
2. In the **Create a new release** panel, check that the artifact version you want to use is selected and choose **Create**.
3. Choose the release link in the information bar message. For example: "Release **Release-1** has been created".
4. In 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**

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

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

Open **Queries**->select … on your query and add that account [{project name} Build Service ({Org name})] Eg: HelloWorldApp Build Service (Type this in search box on top) → Give READ access.

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.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Continuous Integration using YAML Pipelines

- Introduction to YAML Pipeline
- Building Azure DevOps Pipeline using YAML
- Publishing results to Artifacts
- Triggering Continuous Integration in YAML
- Filtering Tasks based on branch being built
- Using Templates to Build Multiple Configurations
- Build on Multi-Platform pipeline

Introduction to YAML Pipeline

- **YAML**: **Y**et **A**nother **M**arkup **L**anguage
- The pipeline is versioned with your code and follows the same branching structure. You get validation of your changes through code reviews in pull requests and branch build policies.
- Every branch you use can modify the build policy by modifying the **azure-pipelines.yml** file.
- A change to the build process might cause a break or result in an unexpected outcome. Because the change is in version control with the rest of your codebase, you can more easily identify the issue.

Exercise: Build Azure DevOps Pipeline using YAML

1. Pipeline → **New Pipeline**
2. Select Tab → Select **HelloWorldApp.Web** repository.
3. On the **Configure** tab, select **ASP.NET Core** (click on choose more)
4. In Visual Studio Code, modify ***azure-pipelines.yml*** as you see here:

```
trigger:
- 'none'

pool:
  vmImage: 'ubuntu-16.04'

steps:
- task: DotNetCoreCLI@2
  displayName: 'Restore project dependencies'
  inputs:
    command: 'restore'
    projects: '**/*.csproj'
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
- task: DotNetCoreCLI@2
  displayName: 'Build the project - Release'
  inputs:
    command: 'build'
    arguments: '--no-restore --configuration Release'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  displayName: 'Publish the project - Release'
  inputs:
    command: 'publish'
    projects: '**/*.csproj'
    publishWebProjects: false
    arguments: '--no-build --configuration Release --output $(Build.ArtifactStagingDirectory)/Release'
    zipAfterPublish: true

- task: PublishBuildArtifacts@1
  displayName: 'Publish Artifact: drop'
  condition: succeeded()
```

The **first task** uses the **DotNetCoreCLI@2** task to ***publish***, or package, the application's build results (including its dependencies) into a folder.

The **zipAfterPublish** argument specifies to add the built results to a .zip file.

The **second task** uses the PublishBuildArtifacts@1 task to publish the .zip file to Azure Pipelines.

Triggering Continuous Integration in YAML

You can control which branches get CI triggers with a simple syntax:

```
trigger:
- master
- releases/*
```

```
# A pipeline with no CI trigger
trigger: none
OR
trigger:
- none
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

specific branch build with batching

```
trigger:
 batch: true
 branches:
  include:
  - master
```

specific branch build

```
trigger:
 branches:
  include:
  - master
  - releases/*
  exclude:
  - releases/old*
 paths:
  include:
  - docs/*
  exclude:
  - docs/README.md
```

Example: To exclude azure-pipelines.yml

```
trigger:
  branches:
    include:
    - master
  paths:
    exclude:
    - azure-pipelines.yml
```

In addition to specifying branch names in the branches lists, you can also configure triggers based on tags by using the following format:

```
trigger:
 branches:
  include:
   refs/tags/{tagname}
  exclude:
   refs/tags/{othertagname}
```

You can specify the target branches for your pull request builds.

YAML PR triggers are only supported in GitHub and Bitbucket Cloud and for NOT for Azure Repos

```
pr:
- master
- releases/*
```

If **no pr triggers** appear in your YAML file, pull request builds are **automatically enabled** for all branches, as if you

```
pr:
 branches:
  include:
  - '*'  # must quote since "*" is a YAML reserved character; we want a string
```

Override YAML triggers

PR and CI triggers that are configured in YAML pipelines can be overridden in the pipeline settings, and by default, new pipelines automatically override YAML PR triggers. To configure this setting, select **Triggers** from the settings menu while editing your YAML pipeline.

Using Templates to Build Multiple Configurations

A *template* enables you to define common build tasks one time and reuse those tasks multiple times.

You call a template from the parent pipeline as a build step. You can pass parameters into a template from the parent pipeline.

Templates combine the content of multiple YAML files into a single pipeline.

Requirement: We need to now repeat the two tasks **Build** and **Publish** but replace **Release** with **Debug**

1. Create a *templates* directory at the root of your project:
2. Create a new File **templates\build.yml**

```
parameters:
   buildConfiguration: 'Release'


steps:
- task: DotNetCoreCLI@2
 displayName: 'Build the project - ${{ parameters.buildConfiguration }}'
 inputs:
  command: 'build'
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
    arguments: '--no-restore --configuration ${{ parameters.buildConfiguration }}'
    projects: '**/*.csproj'

- task: DotNetCoreCLI@2
  displayName: 'Publish the project - ${{ parameters.buildConfiguration }}'
  inputs:
    command: 'publish'
    projects: '**/*.csproj'
    publishWebProjects: false
    arguments: '--no-build --configuration ${{ parameters.buildConfiguration }} --
output $(Build.ArtifactStagingDirectory)/${{ parameters.BuildConfiguration }}'
    zipAfterPublish: true
```

Note the two differences

- In a template file, you use the parameters section instead of variables to define inputs.
- In a template file, you use ${{ }} syntax instead of $() to read a parameter's value. When you read a parameter's value, you include the parameters section in its name. For example, ${{ parameters.buildConfiguration }}.

3. Call the template from the pipeline
4. Place the below two task just above the last task in **azure-pipelines.yml**

```
- template: templates/build.yml
  parameters:
    buildConfiguration: 'Debug'

- template: templates/build.yml
  parameters:
    buildConfiguration: 'Release'
```

5. You see that the pipeline produces a *.zip* file for both the **Debug** configuration and the **Release** configuration.

Job templates:

In this example, a single job is repeated on three platforms. The job itself is specified only once.

 # File: jobs/build.yml

parameters:

name: ''

pool: ''

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
        sign: false
jobs:
        - job: ${{ parameters.name }}
          pool: ${{ parameters.pool }}
          steps: - script: npm install
          - script: npm test
          - ${{ if eq(parameters.sign, 'true') }}:
                - script: sign
```

 # File: azure-pipelines.yml

```
jobs:
        - template: jobs/build.yml # Template reference
          parameters:
                name: macOS
                pool:
                        vmImage: 'macOS-10.14'

        - template: jobs/build.yml # Template reference
          parameters:
                name: Linux
                pool:
                        vmImage: 'ubuntu-16.04'
        - template: jobs/build.yml # Template reference
          parameters:
                name: Windows
                pool:
                        vmImage: 'ubuntu-16.04'
                sign: true #Extra step on windows only
```

Understanding YAML File Format

YAML file Format:

```
name: string  # build numbering format
resources:
  pipelines: [ pipelineResource ]
```

```
 containers: [ containerResource ]
 repositories: [ repositoryResource ]
variables: { string: string } | [ variable | templateReference ]
trigger: trigger
pr: pr
stages: [ stage | templateReference ]
```

- If you have a single stage, you can omit stages and directly specify jobs:
- If you have a single stage and a single job, you can omit those keywords and directly specify steps:

Example:

```
name: $(Date:yyyyMMdd)
variables:
 var1: value1
jobs:
- job: One
 steps:
 - script: echo First step!
```

Stage

A stage is a logical boundary in the pipeline. It can be used to mark separation of concerns (e.g., Build, QA, and production). Each stage contains **one or more jobs**.

By default, stages run sequentially, starting only after the stage ahead of them has completed. You can manually control when a stage should run using approval checks.

Example:

```
stages:
- stage: Build
 jobs:
 - job: BuildJob
   steps:
   - script: "echo Building!"
- stage: Test
 jobs:
 - job: TestOnWindows
   steps:
   - script: echo Testing on Windows!
 - job: TestOnLinux
   steps:
```

```
  - script: echo Testing on Linux!
- stage: Deploy
 jobs:
 - job: Deploy
  steps:
  - script: echo Deploying the code!
  - script: echo Deploying the code!
```

Job

A job is a collection of **linear series of steps** to be run by an agent or on the server. It's a units of work assignable to a particular machine. More than one jobs can run in parallel.

```
jobs:
- job: MyJob
 displayName: My First Job
 continueOnError: true
 workspace:
  clean: outputs
 steps:
 - script: echo My first job
```

Step

A step is the smallest building block of a pipeline. A step can either be a **script or a task**. For example, a pipeline might consist of build and test steps.

Tasks

A task is simply a pre-created script offered as a convenience to you. This abstraction makes it easier to run common build functions.

```
steps:
- script: echo This runs in the default shell on any machine
- bash: |
   echo This multiline script always runs in Bash.
   echo Even on Windows machines!
- pwsh: |
   Write-Host "This multiline script always runs in PowerShell Core."
   Write-Host "Even on non-Windows machines!"
- task: DotNetCoreCLI@2
   displayName: 'Build the project'
   inputs:
```

```
    command: 'build'
    arguments: '--no-restore --configuration Release'
    projects: '**/*.csproj'
```

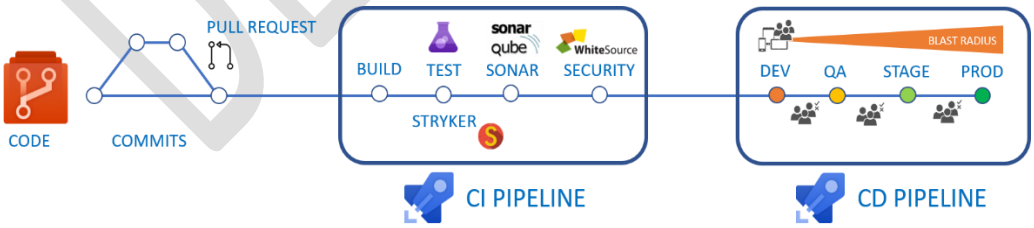

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Continuous Integration using Azure Build Pipelines

- About Azure DevOps Pipeline
- Understanding the Build Process
- Create a Pipeline using Classic Editor
- Enable Continuous Triggers for Build Pipeline
- Add a status badge to Repository
- Working with Task Groups
- Validate Pull Request based on Build Pipeline result
- Add a Widget to Dashboard

About Azure DevOps Pipeline

- Azure Pipelines is a **cloud service** that you can use to **automatically build, test, and deploy** your code project. It is a fully featured continuous integration (CI) and continuous delivery (CD) service.
- Before you use continuous integration and continuous delivery practices for your applications, you must have your **source code in a version control system**. Azure Pipelines integrates with GitHub, GitHub Enterprise, Azure Repos Git & TFVC and Bitbucket Cloud.
- You can **automate** the build, testing, and deployment of your code to Microsoft Azure, Google Cloud Platform, or Amazon Web Services or On-Premise also.
- You can use many languages with Azure Pipelines, such as Python, Java, JavaScript, PHP, Ruby, C#, C++, and Go.
- Use Azure Pipelines to deploy your code to multiple targets. Targets include container registries, virtual machines, Azure services, or any on-premises or cloud target.
- To produce packages that can be consumed by others, you can publish NuGet, npm, or Maven packages to the built-in package management repository in Azure Pipelines. You also can use any other package management repository of your choice.



Agent:

A ***build agent*** builds or deploys the code. When your build or deployment runs, the system begins one or more jobs. An agent is **installable software** that runs one build or deployment job at a time.

Microsoft Hosted Agent: Each time we run a pipeline, we'll get a fresh virtual machine. There are six virtual machine images to choose from, including Ubuntu 16.04 and windows 2019.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Pipeline:

A ***pipeline*** defines the continuous integration and deployment process for the app. It's made up of **one or more stages**. It can be thought of as a workflow that defines how your build, test, and deployment steps are run.

The pipeline runs when you submit **code** changes. You can configure the pipeline to run automatically, or you can run it manually. You connect your pipeline to a source repository like GitHub, Bitbucket, or Subversion.

Run

A run represents one execution of a pipeline. It collects the logs associated with running the steps and the results of running tests.

Trigger

A trigger is something that's set up to tell the pipeline when to run. You can configure a pipeline to run upon a push to a repository, at scheduled times, or upon the completion of another build. All of these actions are known as triggers.

Build Artifacts

An artifact is a collection of files or packages published by a run. Artifacts are made available to subsequent tasks, such as distribution or deployment.

The final product of the build pipeline is a ***build artifact***. Think of an artifact as the smallest compiled unit that we need to test or deploy the app. For example, an artifact can be:

- A Java or .NET application packaged into a .jar or .zip file.
- A C++ or JavaScript library.
- Docker image.

Benefits of Continuous CI

Integration Continuous Integration (CI) provides many benefits to the development process, including:

- Improving code quality based on rapid feedback.
- Triggering automated testing for every code change.
- Reducing build times for rapid feedback and early detection of problems (risk reduction)
- Better managing technical debt and conducting code analysis.
- Reducing long, difficult, and bug-inducing merges.
- Increasing confidence in codebase health long before production deployment.

Clone my repository for all CI and CD pipeline demos

1. D:\>git clone https://sandeepsoni@dev.azure.com/sandeepsoni/HelloWorldApp/_git/HelloWorldApp

2. delete .git folder from HelloWorldApp (use windows explorer)
3. cd HelloWorldApp
4. C:\HelloWorldApp>Code .
5. VS Code -> View -> Extensions -> Search for "gitnore" extensions -> Install (CodeZombie)
6. Press Ctrl+Shift+ P or F1 -> Select "Add gitignore" -> Search and Select "VisualStudio: VisualStudio.gitignore"
7. D:\HelloWorldApp>git init
8. D:\HelloWorldApp>git add .
9. D:\HelloWorldApp>git commit -m "Initial Commit"
10. Go to https://dev.azure.com -> Select Organization -> Create a New DevOps Project
11. Go to Azure Repos -> Copy command from section "Push an existing repository from command line"
12. D:\HelloWorldApp>git remote add origin https://<yourorg>@dev.azure.com/<yourorg>/<yourproj>/_git/<yourproj>
13. D:\HelloWorldApp>git push -u origin –all

Understanding the Build Process

Requirement:

- The build machine is running Ubuntu 16.04.
- The build machine includes build tools like:
  - The .NET Core SDK.
  - NuGet, the package manager for .NET.

Create the Application Locally

1. dotnet new sln -o HelloWorldApp
2. cd HelloWorldApp
3. dotnet new mvc -n HelloWorldApp.Web
4. dotnet sln add HelloWorldApp.Web\HelloWorldApp.Web.csproj
5. code .

Here are the steps that happen during the build process:

1. **Print** build info to the wwwroot directory to help the QA team identify the build number and date.

   echo date > ./wwwroot/buildinfo.txt

2. Run **dotnet restore** to install the project's dependencies.

   dotnet restore

3. Run **dotnet build** to build the app under both Debug and Release configurations.

   dotnet build --configuration Debug #Is used for debugging purpose so that we can get detailed messages/errors.

dotnet build --configuration Release #Is used in Production. Its performance is better.

4. Run **dotnet publish** to package the application as a .zip file and copy the results to a network share for the QA team to pick up.

   dotnet **publish** --configuration Debug --output /Debug

   dotnet **publish** --configuration Release --output /Release

This table associates the script commands with the new Azure Pipelines tasks:

Script command	Azure Pipelines task
echo `date`	CmdLine@2 or script
dotnet restore	DotNetCoreCLI@2
dotnet build	DotNetCoreCLI@2
dotnet publish	DotNetCoreCLI@2

Create a Pipeline using Classic Editor (Designer)

In Azure DevOps, you can use one of two methods to configure a pipeline:

1. **The visual designer**. Here, you drag tasks onto a form and then configure each task to do exactly what you need.
2. **A YAML file**. YAML is a compact format that makes it easy to structure the kind of data that's in configuration files. You typically maintain this YAML file directly with your application's source code.

Push the Sample Application to Repo

1. DevOps Portal → Create a Project (HelloWorldApp)
2. Project → **Repo** → Files → Copy Clone URL to Clipboard
3. D:\DevOpsDemos>git clone <Paste the URL>
4. D:\DevOpsDemos>dotnet new sln -o HelloWorldApp
5. D:\DevOpsDemos> cd HelloWorldApp
6. D:\DevOpsDemos\HelloWorldApp> dotnet new mvc -n HelloWorldApp.Web
7. D:\DevOpsDemos\HelloWorldApp> dotnet sln HelloWorldApp.sln add HelloWorldApp.Web\HelloWorldApp.Web.csproj
8. git add .
9. git commit -m "Initial Commit"
10. git push origin master
11. Switch back to the web portal and View the Committed Files.

Create a Pipeline

1. Pipeline → New Pipeline → **Use the classic editor**
2. Select your Source Code Repository and Branch → Continue

3. Select a Template: Select **ASP.NET Core** → Apply
4. This generates all the Tasks in a Job.
5. (Optional) Add Task: **File Creator** to create a file

   File path = ./HelloWorldApp.Web/wwwroot/buildinfo.txt,

   Content = "$(Build.DefinitionName), $(Build.BuildId), $(Build.BuildNumber)"
6. (Optional) Add a Task: **Command line** → Add

   Script = echo "$(Build.DefinitionName), $(Build.BuildId), $(Build.BuildNumber)"
7. Click on **Save and queue**

Once the Pipelines completes the task – Check the Artifacts or check for the error.

Download the Artifacts

Select the Pipeline → Select the Run → On top expand button **Artifacts** → drop → download

Note: Check your email. You might have already received a build notification with the results of your run. You can use these notifications to let your team members know when builds complete and whether each build passed or failed.

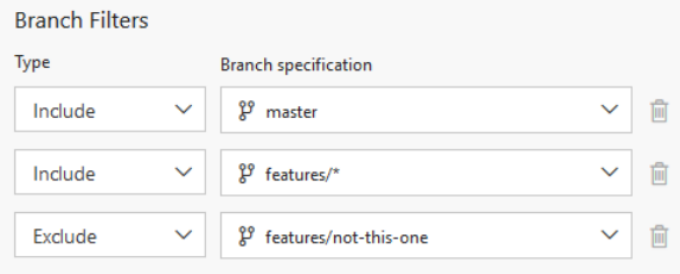
Enable Continuous Integration Triggers

Continuous integration (CI) triggers cause a build to run whenever a push is made to the specified branches or a specified tag is pushed.

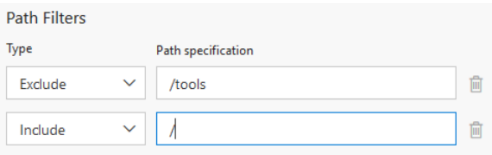
Pipelines → **Pipelines** → Select Build Pipeline → Edit → **Tab Triggers** → **Continuous integration** → **Enable continuous integration**

For example:

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



You also don't want builds to be triggered by changes to files in the tools folder.



Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Batch changes

Select this check box if you have many team members uploading changes often and you want to reduce the number of builds you are running. If you select this option, when commits are made when a build is already running, the system waits until the build is completed and then queues another build of all changes that have not yet been built.

Working with Task Groups (Classic)

A *task group* allows you to encapsulate a sequence of tasks, already defined in a build or a release pipeline, into a **single reusable task** that can be added to a build or release pipeline, just like any other task. You can choose to extract the parameters from the encapsulated tasks as configuration variables, and abstract the rest of the task information.

Create Task Group

1. For both Restore and Build unlink any parameters.
2. Select a **sequence of tasks** (restore and build tasks) in a build pipeline. Then open the shortcut menu and choose **Create task group**.
3. Specify a name and description for the new task group, and the category (tab in the Add tasks panel) you want to add it to.
4. After you choose **Create**, the new task group is created and replaces the selected tasks in your pipeline.
5. **All the '$(vars)' from the underlying tasks, excluding the predefined variables, will surface as the mandatory parameters for the newly created task group.**
6. Save your updated pipeline.

Manage task groups

7. Azure Pipelines → **Task Groups** → Select the Task Group
8. Details page opens and you can now manage Tasks and Properties as per requirement.
9. **Hover** the Task Group → Use the **Export** shortcut command to save a copy of the task group as a JSON pipeline, and the **Import** icon to import previously saved task group definitions. Use this feature to transfer task groups between projects and enterprises, or replicate and save copies of your task groups.

Add a Widget to Dashboard

1. In Azure DevOps, select **Overview**, and then select **Dashboards**.
2. Select **Add a widget**.
3. In the **Add widget** pane, search for **Build History**.
4. Drag the **Build History** tile to the canvas.
5. Select the gear icon to configure the widget.

   a. Keep the **Build History** title.

b. In the **Pipeline** drop-down list, select your pipeline.

c. Keep **All branches** selected.

6. Select **Save**.
7. Select **Done Editing**.

   The **Build History** widget is displayed on the dashboard.

8. Hover over each build to view the build number, when the build was completed, and the elapsed build time.

   Each build succeeded, so the bars on the widget are all green. If the build had failed, it would appear in red.

9. Select one of the bars to drill down into that build.

To display more widgets, select the **Extension Gallery** link at the bottom of the **Add Widget** pane.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Azure Repos and Git

- Version control using Git
- What is Git, Azure Repos and GitHub
- Install Git Locally
- Getting Started with Git Commands
- Updating to DevOps Repository
- Working with Branches
- Merging Branches
- Creating and Committing a Pull Request
- Add a rule to Require a Review
- Squash Merging during Pull Request.
- Working with Merge Conflicts
- Cherry-Picking and Rebase
- Undo Changes using Reset and Revert
- Ignoring files using gitignore
- Managing Git Branches in Azure Repos
- Branch Policies and Branch Permissions
- Branches in Folders
- Working the GitHub Repositories
- Branching Workflow Types
    - Feature Branching
    - Gitflow Branching
    - Forking Workflow
- Summary of Git Commands

Version Control using Git

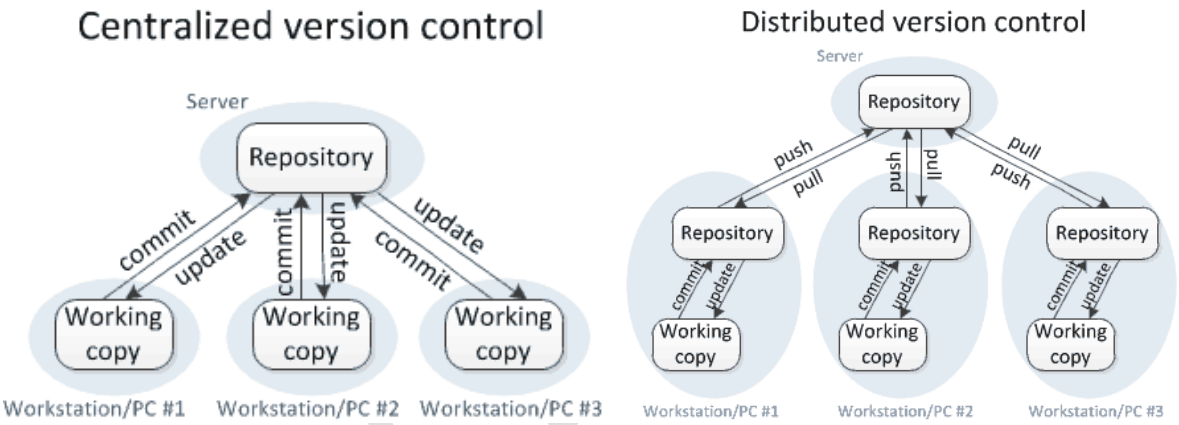
DevOps is a revolutionary way to release **software quickly and efficiently** while maintaining a high level of security. Source control (version control) is a critical part of DevOps

- Version control systems are software that help you track changes you make in your code over time. As you edit your code, you tell the version control system to take a **snapshot** of your files. The version control system saves that snapshot permanently so you can recall it later if you need it.
- Use version control to save your work and coordinate code **changes across your team**.

- Even if you're just a single developer, version control helps you stay organized as you fix bugs and develop new features. Version control keeps a history of your development so that you can review and even rollback to any version of your code with ease.

Benefits of Source Control

- Create workflows
- Work with versions
- Collaboration between developers
- Maintains history of changes
- Automate tasks



Centralized Version Control:

- There is a single central copy of your project and programmers commit their changes to this central copy.
- Common centralized version control systems are TFVC, CVS, Subversion (or SVN) and Perforce.

Strengths	Best Used for
• Easily scales for very large codebases • Granular permission control • Permits monitoring of usage • Allows exclusive file locking	• Large integrated codebases • Audit and access control down to the file level • Hard to merge file types

Distributed Version Control:

- Every developer clones a copy of a repository and has the full history of the project
- Common distributed source control systems are Mercurial, Git and Bazaar.

Strengths:	Best Used for

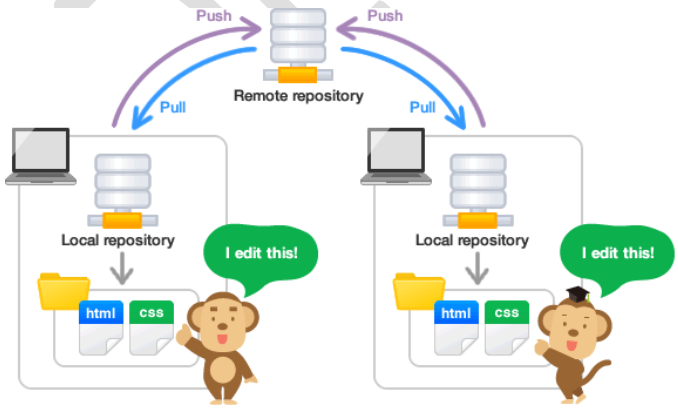
• Cross platform support. • An open source friendly code review model via pull requests. • Complete offline support. • Portable history. • An enthusiastic growing user based.	• Small and modular codebases. • Evolving through open source. • Highly distributed teams. • Teams working across platforms.

What is Git

- Git is the most commonly used **version control system**, a tool to manage your source code **history**. It is quickly becoming the standard for version control.
- Git is a **distributed** version control system, meaning that your **local copy** of code is a complete **version control repository**. These fully functional local repositories make it is easy to work offline or remotely. You commit your work locally, and then sync your copy of the repository with the copy on the server.
- You can use the clients and tools of your choice, such as Git for Windows, Mac, partners' Git services, and tools such as Visual Studio and Visual Studio Code.

Git Repository

- A Git repository, or repo, is a folder that you've told Git to help you track file changes in. You can have any number of repos on your computer, each stored in their own folder.
- Each Git repo on your system is independent, so changes saved in one Git repo don't affect the contents of another.
- A Git repo contains every version of every file saved in the repo. Git saves these files very efficiently, so having a large number of versions doesn't mean that it uses a lot of disk space. Storing each version of your files helps Git merge code better and makes working with multiple versions of your code quick and easy



Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Benefits of Git

- Development with local copy.
- Faster releases because branches allow simultaneous development.
- Strong community support being open source.
- Built-in integration in many IDE. (eg: Visual Studio Code and VS.NET)
- Setup Branch policies to ensure pull requests are reviewed.

Objections to using Git

- I can overwrite history
- I have large files (binaries)
- I have a very large repo
- I don't want to use GitHub (because its by MS)
- There's a steep learning curve

What is GitHub?

GitHub is a hosting service for **Git** repositories. **GitHub** is the service for projects that use **Git**.

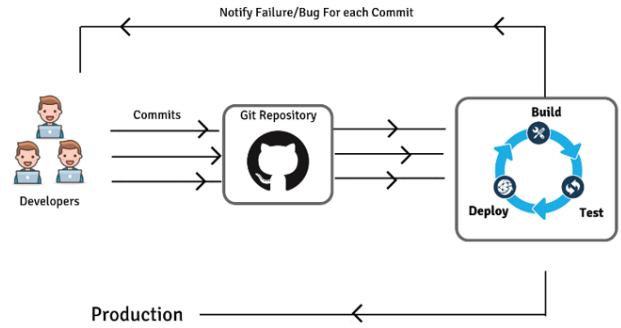
While Git is a command line tool, GitHub provides a Web-based graphical interface.

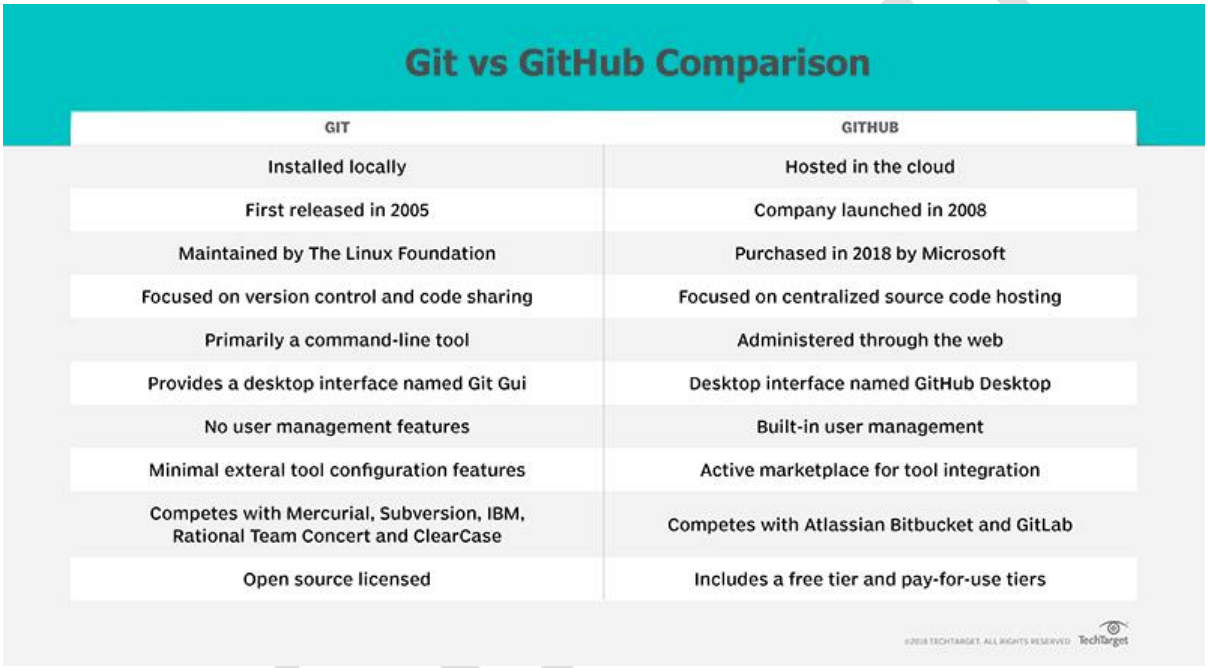
GitHub also provides access control and several collaboration features, such as a wikis and basic task management tools for every project.

GitHub essentially manages:

- Repositories
- Branches
- Commits
- Pull Requests
- Git (the version control software GitHub is built on)

The flagship functionality of GitHub is "**forking**" – copying a repository from one user's account to another. This enables you to take a project that you don't have write access to and modify it under your own account. If you make changes you'd like to share, you can send a notification called a "pull request" to the original owner. That user can then, with a click of a button, merge the changes found in your repo with the original repo.





Note: You don't *have* to use a remote service like GitHub or Azure Repos if all you want is version control - local git is just fine for that.

Azure Repos:

Azure Repos in Azure DevOps is equivalent to GitHub. It's a set of version control tools that you can use to manage your code.

Azure Repos provides two types of version control:

- **Git**
  - Distributed source control system
  - Each developer has a copy of the source repository on their dev machine
- **TFVC**
  - Centralized source control system

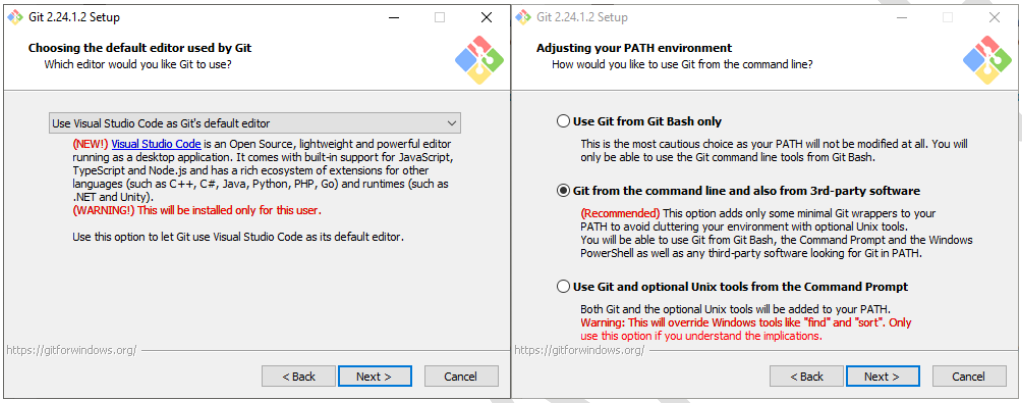
- Team members have **only one version of each file** on their dev machines.
- In the *Server workspaces* model, before making changes, team members publicly check out files.
- In the *Local workspaces* model, each team member takes a copy of the latest version of the codebase with them and works offline as needed.

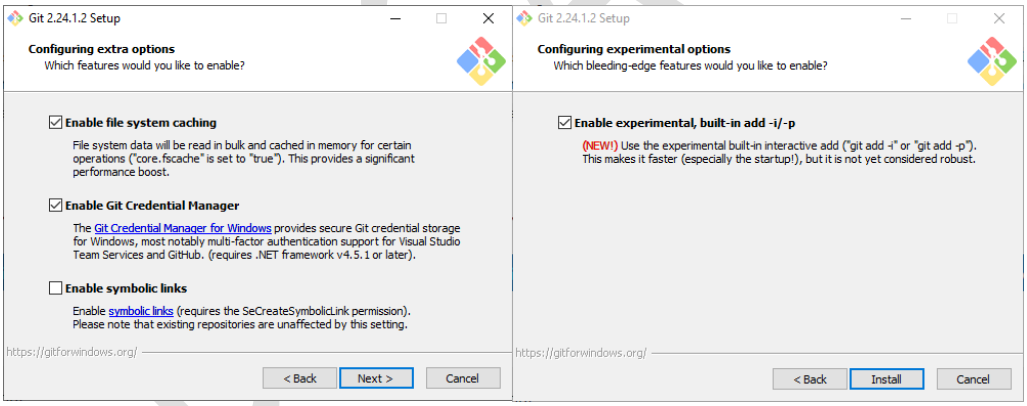
Download and install Git

Git Command line package (https://git-scm.com/downloads)

Download and run the latest Git for Windows installer, which includes the **Git Credential Manager** for Windows.

Make sure to enable the Git Credential Manager installation option.





Authenticating to Your Git Repos:

When you first connect to a Git repo (Azure Repo) the **credential manager prompts** for your Microsoft Account or Azure Active Directory credentials. Once authenticated, the credential manager creates and caches a **personal access token** for future connections to the repo.

- Git Credential Manager simplifies authentication with your Azure DevOps Services/TFS Git repos

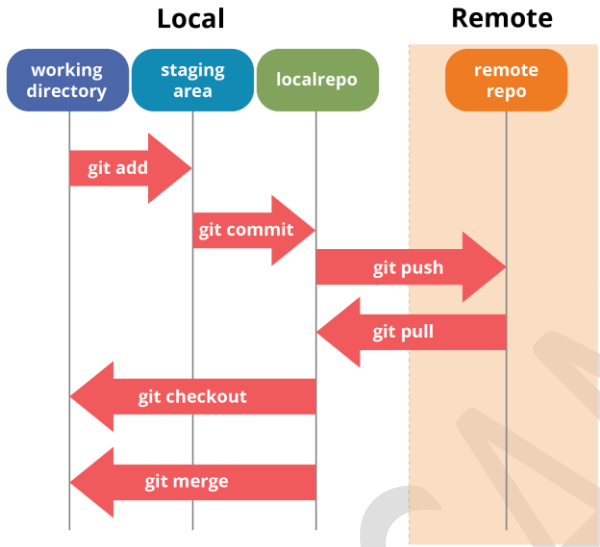
Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

- Supports MFA and two-factor authentication with GitHub repositories
- You can also configure your IDE with a Personal Access Token or SSH to connect with your repos.

Git Commands Documentation on Local System: C:\Program Files\Git\mingw64\share\doc\git-doc

Getting Started with Git Commands

Git Data Transport Commands



Solution (.sln)

- ➔ Project (.csproj) → Restore → Build → EXE/DLL → Publish
  - CS Files
  - Config Files
  - XML Files
  - HTML / CSS / JS

1. Create the ASP.NET Core Web application with Unit Test project.

```
dotnet new sln -o HelloWorldApp
cd HelloWorldApp
dotnet new mvc -n HelloWorldApp.Web
dotnet sln HelloWorldApp.sln add HelloWorldApp.Web\HelloWorldApp.Web.csproj
```

2. OPTIONAL - Now the build the created project: Type the following code in Terminal Window (Ctrl + ~)

```
dotnet dev-certs https --trust
dotnet build
cd HelloWorldApp.Web
dotnet .\bin\Debug\netcoreapp3.1\HelloWorldApp.Web.dll
cd..
```

It will restore all the packages and build the project.

Note: The project is not added to the Git yet. Let's start adding to Git.

3. First step is initializing the Git repository.

```
git init
```

This creates a hidden folder .git in the current directory

4. Add **.gitignore** file as below to the root directory where git hidden folder is present.

```
Code .
In .gitignore add following:
obj
bin
```

Note that the .gitignore file is now added to the current directory.

5. Now let's check the status of git repository.

```
git status
```

There are no commits till now, that will be reflected in the output

Stage your changes

Git does not automatically add changed files to the snapshot when you create a commit. You must first stage your changes to let Git know which updates you want to add to the next commit. Staging lets you to selectively add files to a commit while excluding changes made in other files.

6. Now add all the files to the staging

```
git add .
```

Save with commits

Commits include the following information:

a. A snapshot of the files saved in the commit. Git snapshots the contents of all files in your repo at the time of the commit—this makes switching versions very fast and helps Git merge changes.
b. A reference to the parent commit(s). Commits with multiple parents occur when branches are merged together.
c. A short and to the point message describing the changes in the commit. You enter this message when you create the commit.

7. Now commit all the changes to the master branch (default branch) in local git repository

```
OPTIONAL: git config user.name sandeep
OPTIONAL: git config user.email sandeep@dss.com
git commit -m "First commit"
```

-a is used for staging all files before committing. Commit any files you've added with git add, and also commit any files you've changed since then

The commit command runs with the -m flag, which allows you to pass a message through the command line. If you don't provide this, Git will open up an editor in the terminal so you can enter a commit message.

Updating to Azure DevOps Repository

Once you are ready to share your code, get the clone URL for the repository you want to connect to and then set up a remote relationship (in this case, origin) so your repo can push changes to a shared repo.

If the Local Repository is already Existing:

8. **DevOps Portal** → Project → Repos → Copy the Commands from second section as below
   git remote add origin https://DemoOrg@dev.azure.com/ DemoOrg /HelloWorldDemo/_git/HelloWorldDemo
   git push -u origin --all
9. Execute the above commands on your local machine.

It prompts for sign in. Sign in with the DevOps credentials.

If New Local Repository has to be created. This creates a local copy of project.

10. Open another command window on your local machine

If the local folder is Empty then use the following command to clone the remote repo to local.

```
git clone <url>
```

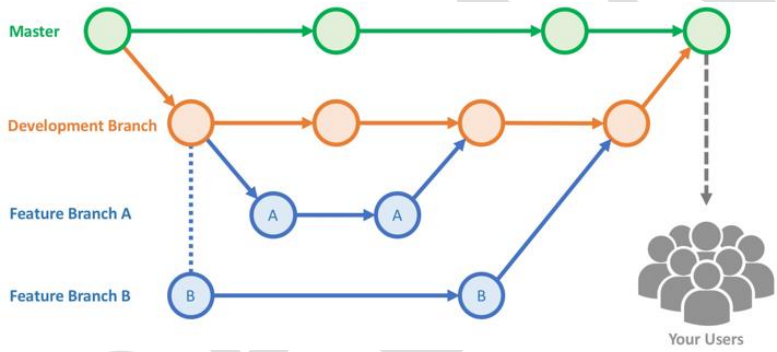
url should be copied from Azure DevOps portal eg:https:// DemoOrg @dev.azure.com/DemoOrg /HelloWorldDemo /_git/HelloWorldDemo.

In windows use "**doskey /history"** = To get the list of commands from history.

Working with Branches

Git branches aren't much more than a small reference that keeps an exact history of commits, so they are very cheap to create. Committing changes to a branch will not affect other branches, and you can share branches with others without having to merge the changes into the main project. Create new branches to isolate changes for a feature or a bug fix from your master branch and other work.

After you feel that your code is ready to be **merged** into the master branch in the main repository that's shared by all developers, you create what's called **a *pull request***. When you create a pull request, you're telling the other developers that you have code ready to review and you want it merged into the master branch. When your pull request is approved, it becomes part of the master codebase.



Some suggestions for naming your feature branches:

1. features/feature-name
2. features/feature-area/feature-name
3. users/username/description
4. users/username/workitem
5. bugfix/description
6. hotfix/description

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Merging Branches

Merging is Git's way of putting a forked history back together again.

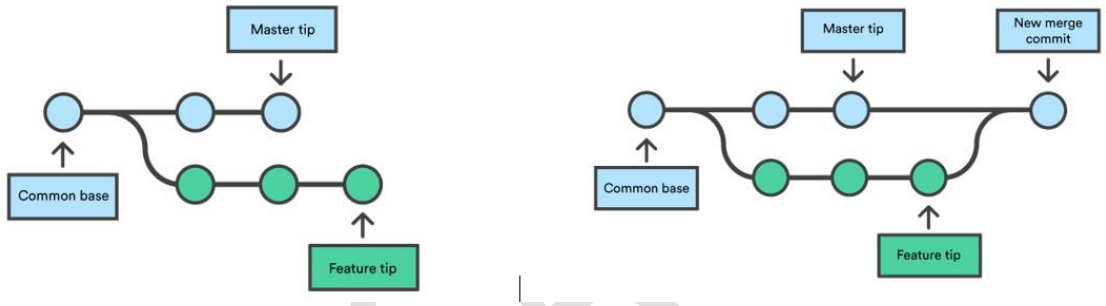
The ***git merge*** command lets you take the independent lines of development created by git branch and integrate them into a single branch.

git merge <branch name>

where <branch name> is the name of the branch that will be merged into the receiving branch.

Note that merge merges the target branch into the current branch. The current branch will be updated to reflect the merge, but the target branch will be completely unaffected. Again, this means that ***git merge*** is often used in conjunction with ***git checkout*** for selecting the current branch

Say we have a new branch feature that is based off the master branch. We now want to merge this feature branch into master.



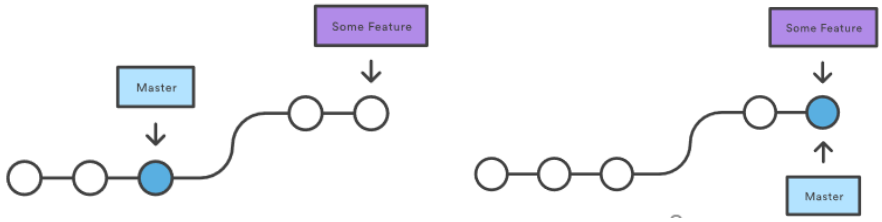
Git will determine the merge algorithm automatically (discussed below).

Two Git Merging Algorithms

Fast Forward Merge

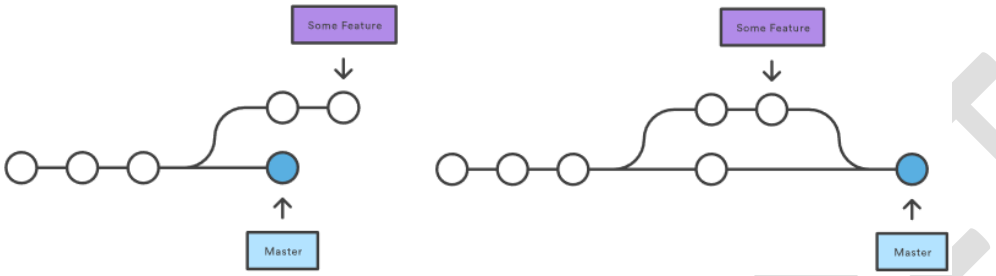
A fast-forward merge can occur when there is a linear path from the current branch tip to the target branch. Instead of "actually" merging the branches, all Git has to do to integrate the histories is move (i.e., "fast forward") the current branch tip up to the target branch tip. This effectively combines the histories, since all of the commits reachable from the target branch are now available through the current one.

For example, a fast forward merge of some-feature into master would look something like the following:



3-way merge

However, a fast-forward merge is not possible if the branches have diverged. When there is not a linear path to the target branch, Git has no choice but to combine them via a 3-way merge. 3-way merges use a dedicated commit to tie together the two histories. The nomenclature comes from the fact that Git uses three commits to generate the merge commit: the two branch tips and their common ancestor.



Merging the branch changes with Master branch

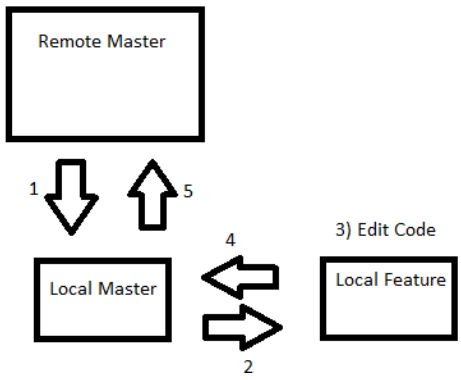
While you were busy working on your feature, there might have been changes made to the remote master branch. Before you **create a pull request**, it's common practice to get the latest from the remote master branch.

pull = fetch + merge (it is a combined command that does a fetch and then a merge)

a) **fetch** downloads the changes from your remote repo but do not apply them to your code.

b) **merge** applies changes taken from fetch to a branch on your local repo.

Working with Local Branch and Pushing Changes to Remote Master:



1. git pull origin master
2. git branch feature1

   git checkout feature1
3. Edit the Code in Index.html

   git add .

git commit -m "feature1"

4. **git checkout master**

   git merge feature1

5. git push origin master
6. **git log** **#**Press 'q' to quit

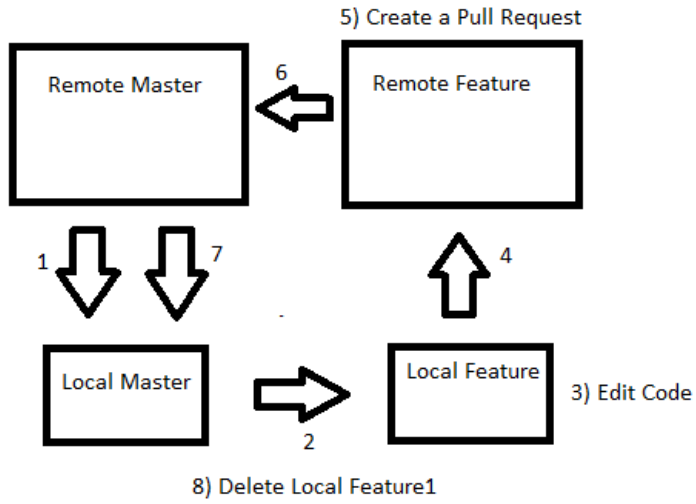
Create a Pull Request (PR) and Commit Changes to Master

Pull requests let your team give feedback on changes in feature branches before merging the code into the master branch. Reviewers can step through the proposed changes, leave comments, and vote to approve or reject the code. Azure DevOps provides a rich experience for creating, reviewing, and approving pull requests.

Pull requests will be issued from feature branch to master branch.

Workflow

1. Create a Feature Branch
2. Checkout to Feature Branch
3. Develop the feature (Edit the files)
4. Stage and Commit changes to Local Feature Branch
5. Push the Commit to Remote Feature Branch
6. Create a Pull Request (Pull from Remote Feature to Remote Master)
7. Pull Request should [Approved] or Rejected.
8. On my local - Pull from the Remote master so that Local master is updated...
9. Delete the Local feature and Remote Feature...



Step1: Create a feature branch – Edit the code and update the feature branch in remote repo

1. git pull origin master
2. git branch features/edit-home-title

   git checkout features/edit-home-title
3. Edit the code

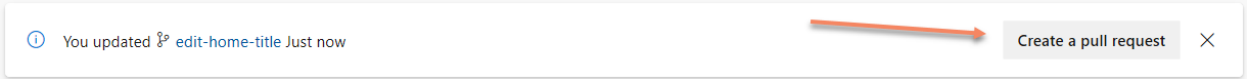
   git add .

   git commit -m "Some Comment"
4. git push origin feature/edit-home-title

Step2: Creating Pull requests in Azure DevOps portal

5. GO to the Azure repos and navigate to feature branch "**edit-home-title**"

   Click on "Create a Pull Request"



   Provide title and description and click on **Create**
6. Now login as the reviewer (email id given while creating pull request)

   Now navigate to Pull requests from Left side menu and click on displayed Pull request.

   1. Navigate to Files tab and review the code changes…

   Now click on the pull requests and check the code changes and we can approve it or reject it. The suggestions can be made in discussion board. Once we are done. We can approve the request.

   Once approved. We can **complete** the merger in the by clicking **Complete → Provide the required details → Complete merge**

Note that the branch edit-home-title is deleted and changes are now reflected in master branch.

Step3: Fetch the latest code from the remote repository and merges it into your local repository and delete the local feature branch

Login as Approver and → Go to Pull Request → Commit the PR

7. git checkout master
   git pull origin master
8. git branch -d feature/edit-home-title
9. git branch #Displays only master branch

ADD A RULE TO REQUIRE A REVIEW:

Branch policies help teams protect their important branches of development. Policies enforce your team's code quality and change management standards.

1. **Select the Branch → Select . . . icon → Branch Policies**
2. Opens configure your policies in the **Policies** page.
3. Check "**Require a minimum number of reviewers"**
4. Minimum number of reviewers = 1
5. Select **Save changes** to apply your new policy configuration.

Note: If a policy is created on a branch, direct push to that branch is not allowed.

Test the Rule

1. git checkout master
   git pull origin master
   git checkout -b bugfix/home-page-typo
2. Edit the Index.cshtml
   git add .
   git commit -m "Fixed home page content"
   git push origin bugfix/home-page-content
3. **Go to DevOps and → Create a pull request**
4. You can see that a human review is required before you can merge the change.
5. To merge the pull request, select **Merge pull request**.
6. Select the **Use your Administrator privileges to merge this pull request** check box, and then select **Confirm merge**.

Recovering a Deleted Branch

- **git reflog**
- Find the SHA-ID for the commit at the tip of your deleted branch,
- git checkout -b <branch> <SHA-ID>

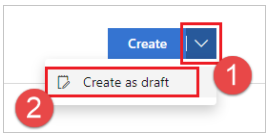
Draft Pull Request

Draft pull requests have the following differences from published pull requests:

- Build validation policies are enabled but not run automatically. They can be manually queued by selecting the **...** menu beside the build in the pull request.
- Voting is disabled while in draft mode.
- Required reviewers aren't automatically added.
- Notifications are sent while in draft mode, but only to reviewers that you explicitly add to the draft pull request.
- Draft pull requests are displayed in the pull requests list with a special badge.

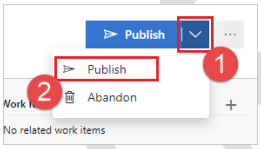
To create a draft pull request,

Choose **Create as draft** when creating the pull request.



Publish a draft pull request

When you're ready to have the pull request reviewed and completed, you can publish it.



When you publish a pull request, required reviewers are assigned and notified, policies are evaluated, and voting begins.

Revert a pull request

To undo the changes in a pull request, follow these steps:

1. Open the completed pull request and select **Revert**. When you revert a pull request in this way, you create a new branch with changes that undo the pull request for an existing target branch in your repo.
2. In **Target branch**, select the branch where you want to undo the pull request changes.

3. In **Topic branch name**, select a new branch where the reverted changes are created, then select **Revert**.
4. Select **Create pull request** to merge the newly created branch in a second pull request to complete the revert.

Squash Merging during Pull Request

Squash merging is a merge option that allows you to **condense the Git history** of topic branches when you complete a pull request. Instead of each commit on the topic branch being added to the history of the default branch, a squash merge takes all the file changes and adds them to a **single new commit** on the default branch. A simple way to think about this is that squash merge gives you just the file changes, and a regular merge gives you the file changes and the commit history.

How is a squash merge helpful?

Squash merging keeps your default **branch histories clean and easy to follow** without demanding any workflow changes on your team. Contributors to the topic branch work how they want in the topic branch, and the default branches keep a linear history through the use of squash merges. The commit history of a master branch updated with squash merges will have **one commit** for each merged branch. You can step through this history commit by commit to find out exactly when work was done.

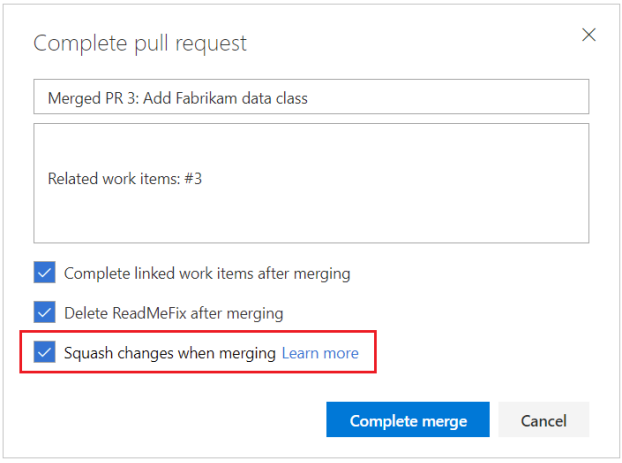
Considerations when squash merging

When squash merging, **it's a good practice to delete the source branch**. This prevents confusion as the topic branch itself does not have a commit merging it into the default branch.

Completing pull requests with squash merge

You can choose to squash merge when completing a pull request in Azure Repos. Choose **Squash changes when merging** on the **Complete pull request** dialog to squash merge the topic branch.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Working with Merge Conflicts

Example to demo AUTO RESOLVE if merge conflict

Open Command Window1: Create Local Git and also Push the same to remote repo.

D:\Demo\>**git config user.name user1**

D:\Demo\>**git config user.email user1@dss.com**

D:\Demo\>md User1

D:\Demo\>cd User1

D:\Demo\User1>git init

D:\Demo\User1>**git remote add origin <URL>**

D:\Demo\User1>Notepad demo.txt

```
This is line1
This is line2
```

D:\Demo\User1>git add .

D:\Demo\User1>git commit -m "Initial Commit"

D:\Demo\User1>git push origin master

Open Command Window2: Clone the repo and add a new line and push to remote

D:\Demo\>md User2

D:\Demo\>cd User2

D:\Demo\User1>**git clone <URL>**

D:\Demo\User1>**git pull origin master**

D:\Demo\>**git config user.name user2**

D:\Demo\>**git config user.email user2@dss.com**

D:\Demo\User2>Notepad demo.txt (Add a new line of text)

This is line1 This is line2 **This is by User2**

D:\Demo\User2>git add .

D:\Demo\User2>git commit -m "User2 Commit"

D:\Demo\User2>git push origin master

Example to demo MANUAL RESOLVE if merge conflict

Open Command Window1: Add a New line (different from User2). Auto resolves merge conflict

D:\Demo\User1>Notepad demo.txt (Add a new line of text)

This is by User1 This is line1 This is line2

D:\Demo\User1>git add .

D:\Demo\User1>git commit -m "User1 Commit"

D:\Demo\User1>git push origin master

Note that this will get rejected.

D:\Demo\User1>git pull **origin master**

Resolves Merge Conflict Automatically and does fast forward merge

D:\Demo\User1>Type Demo.txt

Note that it will auto merge the changes as they are in different lines…

This is by User1 This is Line1 This is Line2 This is by User2

D:\Demo\User1>git push origin

Switch to Command Window2: Edit the same line (First Line) which was updated by user1 without fetching the changes of User1

This is line by User2

This is Line1 This is Line2 This is by User2

D:\Demo\User2>git add .

D:\Demo\User2>

D:\Demo\User2>git push origin master

Merge Conflict will be reported…

D:\Demo\User2>git pull origin master

Merge Conflict will be reported again because auto merge will not work

Resolve the conflict manually by editing the file…

D:\Demo\User2>git add .

D:\Demo\User2>git commit -m "Update"

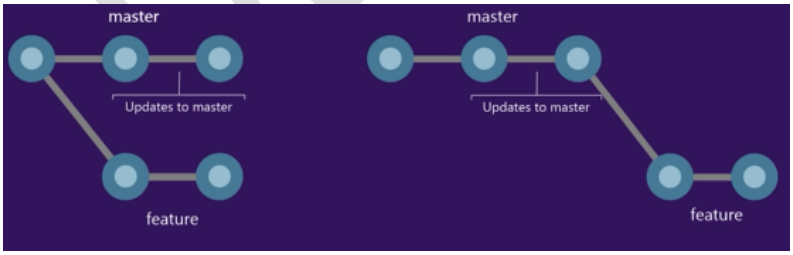
D:\Demo\User2>git push origin master

Understanding Rebase for Resolving

- Git creates history as you save your code in your **commits** and **merges** changes back into the master branch with **pull requests**.

  This generated history can get complicated when you need to update a feature branch with changes from the main branch to **catch up on work committed by others**. Your commit history will diverge from the master branch at multiple points, making it hard to follow.
- Use rebase to address the problem of updating your branch with the latest changes from the main branch. **Rebase takes the changes made in the commits in your current branch (feature branch) and replays them on the history of another branch (master branch).** The commit history of your current branch will be rewritten so that it starts from the most recent commit in the target branch of the rebase.

Before and after rebase:



Rebasing your changes in your feature branch off the latest changes in the main branch lets you test your changes on the most recent version in the main branch while keeping a **clean Git history**.

> git checkout master

> git rebase feature1

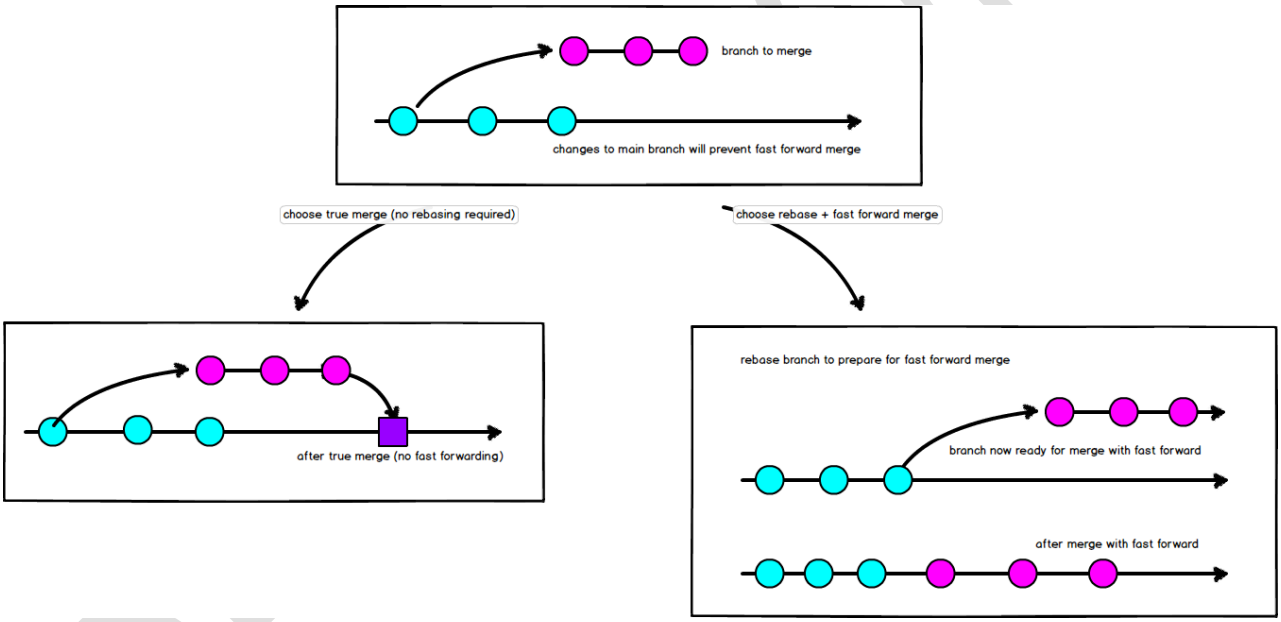
If you hit a conflict, resolve the conflicting files, do a git add to stage the merged changes, then continue the rebase with

> git rebase --continue.

After a successful rebase, your local branch will have a different history than your remote branch. You must force push your local branch to update your remote branch.

```
> git push -f users/sandeep/myfixes
```

Merge vs Rebase



Example:

Open Command Window1: Create Local Git and also Push the same to remote repo.

D:\Demo\>md User1

D:\Demo\>cd User1

D:\Demo\User1>git init

D:\Demo\User1>**git remote add origin <URL>**

D:\Demo\User1>Notepad demo.txt

```
This is line1
This is line2
```

D:\Demo\User1>git add .

D:\Demo\User1>git commit -m "Initial Commit"

D:\Demo\User1>git push origin master

Open Command Window2: Clone the repo and add a new line and push to remote

D:\Demo\>md User2

D:\Demo\>cd User2

D:\Demo\User1>**git clone <URL>**

D:\Demo\User1>**git pull origin master**

D:\Demo\User2>Notepad demo.txt (Add a new line of text)

This is line1 This is line2 **This is by User2**

D:\Demo\User2>git add .

D:\Demo\User2>git commit -m "User2 Commit"

D:\Demo\User2>git push origin master

Open Command Window1: Add a New line (different from User2). Auto resolves merge conflict

D:\Demo\User1>Notepad demo.txt (Add a new line of text)

This is by User1 This is line1 This is line2

D:\Demo\User1>git add .

D:\Demo\User1>git commit -m "User1 Commit"

D:\Demo\User1>git push origin master

Note that this will get rejected.

D:\Demo\User1>git pull **--rebase origin master**

Resolves Merge Conflict Automatically and does fast forward merge

D:\Demo\User1>Type Demo.txt

Note that it will auto merge the changes as they are in different lines…

This is by User1

This is Line1

This is Line2 This is by User2

D:\Demo\User1>git push origin

Switch to Command Window2: Edit the same line (First Line) which was updated by user1 without fetching the changes of User1

This is line by User2 This is Line1 This is Line2 This is by User2

D:\Demo\User2>git add .

D:\Demo\User2>

D:\Demo\User2>git push origin master

Merge Conflict will be reported…

D:\Demo\User2>git pull **--rebase** origin master

Merge Conflict will be reported again because auto merge will not work

Resolve the conflict manually by editing the file…

D:\Demo\User2>git add .

D:\Demo\User2>git **rebase --continue** **#Note: Commit is NOT used here…**

D:\Demo\User2>git push origin master

Cherry-Picking

Copy commits from one branch to another using cherry-pick. **Unlike a merge or rebase**, cherry-pick only brings the **changes** from the **commits you select**, instead of all the changes in a branch.

Cherry-pick is a great way to tackle these common problems:

- Accidentally committing on the wrong branch. Cherry-pick the change(s) over to the correct branch and then reset the original branch to the previous commit.
- Pulling out a set of commits made in a feature branch so you merge them back to your master branch sooner.

Steps:

1. **Edit Demo.txt in Azure Repo (in browser) and put a buggy line**

This is Line1

This is Line2

This is a buggy line

2. Commit the changes from browser itself.

From the Command Prompt get the latest version

3. D:\User1>git pull origin master

Switch to browser and login with a different User Identity (Rahul)

3. Create a New Branch – **cherrydemo**
4. Open the Demo.txt and Edit as below (resolve the bug)

This is Line1

This is Line2

This is a buggy line (Deleted)

5. Commit and provide subject as "Buggy line deleted"

Switch to command window and fetch the commit history of other user (Rahul)

6. git **remote add** hotfix <URL of Repo>
7. git **fetch** hotfix #gets all commits on my local repo
8. git log hotfix/**cherrydemo** (hotfix informed us that he fixed it on this branch)
9. copy the SHA1
10. **git cherry-pick <sha1 of commit>**

Note that the SHA1 in our commit it will be different

In Visual Studio:

1. Commit the changes in Source Branch.
2. Checkout to the branch you want to cherry-pick changes (Target Branch) into using the **Branches** view.
3. Right-click the **Source Branch → View History...**
4. Right-click the **Commit** you want to cherry-pick and select **Cherry-pick**.
5. **Repeat** this process for each commit you need to bring over to your current branch.

Cherry-pick a Pull Request (PR)

To copy changes made in a pull request to another branch in your repo, follow these steps:

1. In a completed pull request, select **Cherry-pick**, or for an active pull request, select **Cherry-pick** from the **...** menu. Cherry-picking a pull request in this way creates a new branch with the copied changes. Merge into a target branch in a second pull request.
2. In **Target branch**, enter the branch you want to merge the copied changes.
3. In **Topic branch name**, enter a new branch to contain the copied changes, then select **Cherry-pick**.
4. Select **Create pull request** to merge the topic branch into the target branch to complete the cherry-pick.

Undo changes using reset or revert

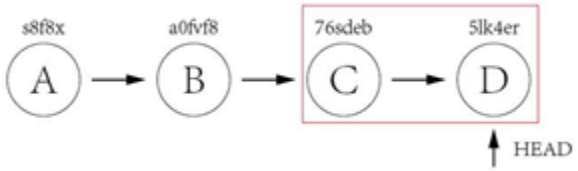
About Command: git reset

Reset your local branch to a previous commit.

git **reset** --hard HEAD
git **reset** --hard <SHA-ID-OF-COMMIT>

1. The --hard part of the command tells Git to reset the files to the state of the previous commit and **discard any staged changes**.
2. The HEAD argument tells Git to reset the local repository to the most recent commit. If you want to reset the repo to a different commit, provide the ID instead of HEAD.

Assuming we have below few commits.

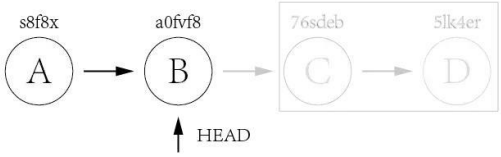


Commit A and B are working commits, but commit C and D are bad commits. Now we want to rollback to commit B and drop commit C and D. Currently HEAD is pointing to commit D *5lk4er*, we just need to point HEAD to commit B *a0fvf8* to achieve what we want.

It's easy to use git reset command.

git reset --hard a0fvf8

After executing above command, the HEAD will point to commit B.



But now the remote origin still has HEAD point to commit D, if we directly use **git push** to push the changes, it will not update the remote repo, we need to add a **-f** option to force pushing the changes.

```
git push -f origin master
```

The drawback of this method is that all the commits after HEAD will be gone once the reset is done. In case one day we found that some of the commits ate good ones and want to keep them, it is too late. Because of this, many companies forbid to use this method to rollback changes.

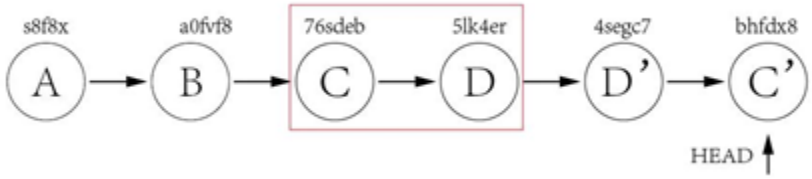
About Command: git revert

The use of **git revert** is to create a new commit which reverts a previous commit. The HEAD will point to the new reverting commit.

For the example of git reset above, what we need to do is just reverting commit D and then reverting commit C.

```
git revert 5lk4er
git revert 76sdeb
```

Now it creates two new commit D' and C',



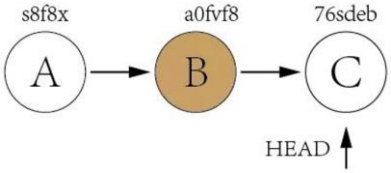
In above example, we have only two commits to revert, so we can revert one by one. But what if there are lots of commits to revert? We can revert a range indeed.

```
git revert OLDER_COMMIT^..NEWER_COMMIT
```

This method would not have the disadvantage of **git reset**, it would point HEAD to newly created reverting commit and it is ok to directly push the changes to remote without using the **-f** option.

Special Case

Now let's take a look at a more difficult example. Assuming we have three commits but the bad commit is the second commit.



It's not a good idea to use **git reset** to rollback the commit B since we need to keep commit C as it is a good commit. Now we can revert commit C and B and then use **cherry-pick** to commit C again.

Execute the following commands

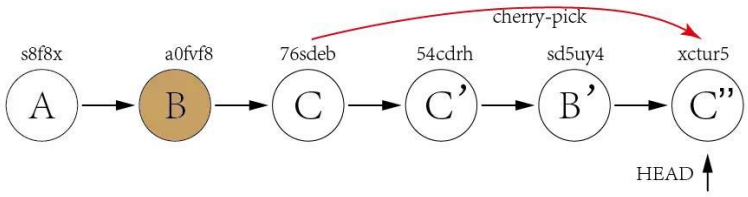
a) git **revert** 76sdeb

Head is at 54cdrh

b) git **revert** a0fvf8

Head is at sd5uy4

c) git **cherry-pick** 76sdeb



From above explanation, we can find out that the biggest difference between **git reset** and **git revert** is that git reset will reset the state of the branch to a previous state by dropping all the changes post the desired commit while git revert will reset to a previous state by creating new reverting commits and keep the original commits. It's recommended to use git revert instead of git reset in enterprise environment.

To temporarily Store uncommitted changes and revert to last commit:

- git stash

To apply the changes to working directory

- git stash pop

Ignore file changes with Git

Use .gitignore to prevent tracking of files

1. **VS Code** → View → **Extensions** → Search and Install visual studio extension: .**gitignore**.
   Once the extension is installed, go to **Command Palette** (View → Command Palette or Ctrl+Shift+P for windows), you can find option to **Add .gitignore** → Enter → Select **VisualStudio.gitignore**.

You can check gitignore in root directory.

2. Edit the .gitignore file as below

```
.vscode/*
!.vscode/settings.json
!.vscode/tasks.json
```

```
!.vscode/launch.json
!.vscode/extensions.json
*.code-workspace

bin/*
obj/*
```

Ignore files only on your system in a given repo:

Your .gitignore is shared across team members as a file committed and pushed to the Git repo.

To exclude files only on your system without pushing the changes to the rest of your team, edit the **.git/info/exclude** file in your local repo. Changes to this file will not be shared with others and only apply to the files in that repo. The syntax for this file is the same as the one used in .gitignore.

Ignore files across all repos on your system (Create a file: .gitignore_global)

```
git config --global core.excludesfile C:\Users\<current user>\.gitignore_global
```

This is particularly useful for ignoring entire file types you don't want to ever commit, such as compiled binaries.

Ignore changes to already committed files

```
git update-index --skip-worktree <file>
```

Resume tracking files with:

```
git update-index --no-skip-worktree <file>
```

Permanently ignore changes to a file

If a file is already tracked by Git, adding that file to your .gitignore is not enough to ignore changes to the file. You also need to remove the information about the file from Git's index:

These steps will not delete the file from your system. They just tell Git to ignore future updates to the file.

1. Add the file in your .gitignore.
2. Run the following:

```
git rm --cached <file/directory> -r
```

3. Commit the removal of the file and the updated .gitignore to your repo.

Managing Git Branches in Azure Repos

Repos → Branches

- The default **Mine** tab on the branches page shows branches you've created, pushed changes to, or set as a favorite, along with the default branch for the repo, such as master.
- The **All** tab lists all branches in the repo
- The **Stale** tab lists branches in the repo that haven't had any commits in **three months** or long

Review updates to your branches

Each branch name has the **name of the last contributor** to the branch and a link to the **latest commit** on the branch.

The branch view also shows the number of commits the branch is **ahead of and behind** the branch labeled **Compare**.

If the branch has a **pull request open**, you can see its pull request ID. Select the pull request to open it up so you can review the details.

Change the compare branch

The **ahead** and **behind** numbers listed for each branch are in comparison with the branch currently labeled **Compare** on the Branches page. Update your compare branch to see how far ahead or behind your branches shown on the page are to another branch in your repo:

1. Select the **...** next to the branch you want to set as the baseline for comparison.
2. Select **Set as compare branch**.

View Branch files / history

Select the … icon next to branch name → View Files or View History

- **View files** opens up the **Files** view on the web so you can browse the files based on the most recent commit on the branch.

- **View history** shows each commit in the branch history. Select a commit from this history to see the file changes made in that commit.

Rename old branches

```
git branch --move {old_branch_name} {new_branch_name}
git push origin {new_branch_name}
git push origin --delete {old_branch_name}
```

Change your default branch

Configure your Git repo to use a different default branch to merge code into when your team creates new pull requests. This is useful when you want to use a branch other than master for new changes or need to change your main line of development in your repo.

Project settings → **Code section** → Repositories → Select the desired repository and expand the branches. → Select the **...** beside the desired branch and choose **Set as default branch**.

Lock a branch

- Prevent updates to a Git branch by locking the branch.
- Locking a branch prevents other users from changing the existing commit history.
- Locking also blocks any new commits from being added to the branch by others.
- Only the user who locked the branch or a user with **Remove Others' Locks permissions** for the branch can remove the lock.

Azure Repo → Repos → Select the Branch → Select . . . icon → Lock / Unlock

Recover Deleted Branches:

Azure DevOps Server now supports **searching** for deleted branches. This helps you understand who deleted it and when, the interface also allows you to recreate the branch it if you wish. To cut out the noise from the search results, deleted branches are only shown if you search for them by their exact name. To search for a deleted branch, enter the full branch name into the branch search box. It will return any existing branches that match that text. If a match is found, you will see who deleted it and when. You can also restore the branch. Restoring the branch will re-create it at the commit to which is last pointed. However, it will not restore policies and permissions.

Branch Permissions (Security)

Set up permissions to control who can read and update the code in a branch on your Git repo. You can set permissions for individual users and groups, and inherit and override permissions as needed from your repo permissions.

Select the Branch → Select . . . icon → Branch Security

Permission	Description
Contribute	Users with this permission can push new commits to the branch and lock the branch.
Edit Policies	Can edit branch policies.
Bypass policies when completing pull requests	Users with this permission are exempt from the branch policy set for the branch when completing pull requests and can opt-in to override the policies by checking **Override branch policies and enable merge** when completing a PR.
Bypass policies when pushing	Users with this permission can push to a branch that has branch policies enabled. Note that when a user with this permission makes a push that would override branch policy, the push automatically bypasses branch policy with no opt-in step or warning.
Force Push (Rewrite History and Delete Branches)	Can force push to a branch, which can rewrite history. This permission is also required to delete a branch.
Manage Permissions	Can set permissions for the branch.
Remove Others' Locks	Can remove locks set on branches by other users.

Note: Git doesn't support to set security permissions at file level.

Working with GitHub Repositories

GitHub.com

1. Create a New Repository in GitHub
2. Copy the clone URL

Local command prompt

1. git clone <URL>
2. dotnet sln HelloWorld.sln add HelloWorldApp\HelloWorldApp.csproj
3. dotnet build
4. dotnet run -p HelloWorldApp\HelloWorldApp.csproj
5. git add .

6. git commit -a -m "solution and project added"
7. git push origin

Branching Workflow Types

When evaluating a workflow for your team, it's most important that you consider your team's culture. You want the workflow to enhance the effectiveness of your team and not be a burden that limits productivity.

Some things to consider when evaluating a Git workflow are:

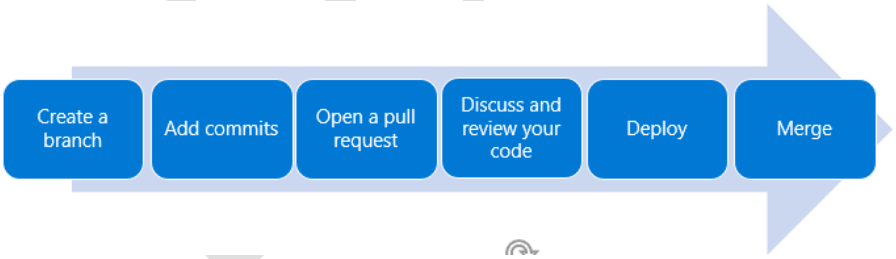
- Does this workflow scale with team size?
- Is it easy to undo mistakes and errors with this workflow?
- Does this workflow impose any new unnecessary cognitive overhead to the team?

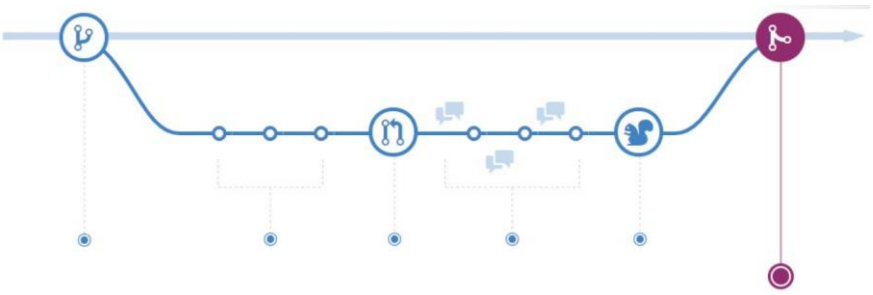
Common branch workflows

1. Trunk Based Development / Feature branching
2. Gitflow branching
3. Forking Workflow

Trunk Based Development / Feature Branch Workflow

- All feature development should take place in a dedicated feature branch instead of the master branch.
- Encapsulating feature development leverages pull requests, which are a way to initiate discussions around a branch.
- Share a feature with others without touching any official code.
- Trunk-based development is a logical extension of Centralized Workflow.





Let's cover the principles of what is being proposed:

The master branch:

- The master branch is the only way to release anything to production.
- The master branch should always be in a ready-to-release state.
- Protect the master branch with branch policies.
- Any changes to the master branch flow through pull requests only.
- Tag all releases in the master branch with Git tags.

The feature branch:

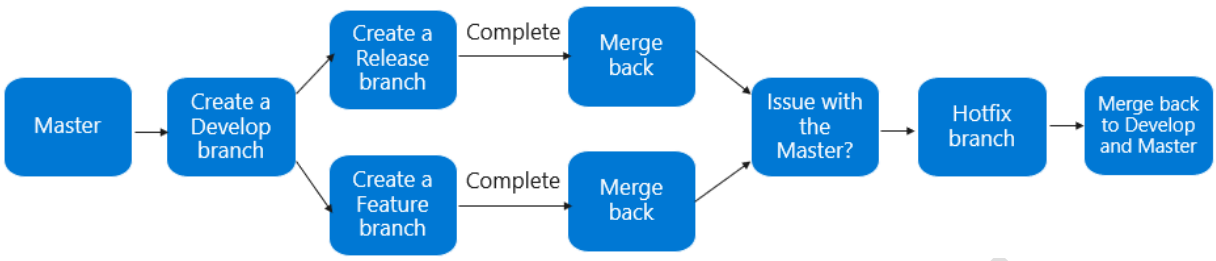
- Use feature branches for all new features and bug fixes.
- Use feature flags to manage long-running feature branches.
- Changes from feature branches to the master only flow through pull requests.
- Name your feature to reflect their purpose.

Pull requests:

- Review and merge code with pull requests.
- Automate what you inspect and validate as part of pull requests.
- Track pull request completion duration and set goals to reduce the time it takes.

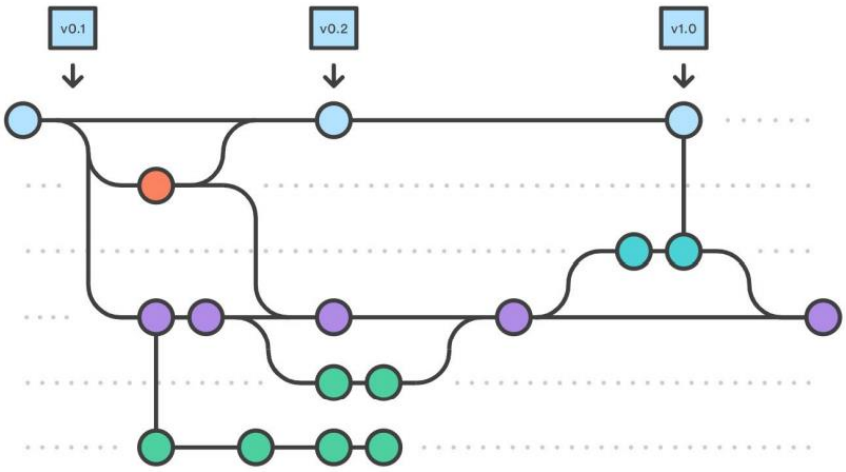
GitFlow Branch Workflow:

- GitFlow is great for a release-based software workflow.
- This provides a robust framework for managing larger projects.
- GitFlow offers a dedicated channel for hotfixes to production.



The overall flow of Gitflow is:

- A develop branch is created from master
- A release branch is created from develop
- Feature branches are created from develop
- When a feature is complete it is merged into the develop branch through a Pull request.
- When the release branch is done it is merged into develop and master
- If an issue in master is detected a hotfix branch is created from master
- Once the hotfix is complete it is merged to both develop and master



Note: Using a dedicated branch to prepare releases makes it possible for one team to polish the current release while another team continues working on features for the next release. Once the release is ready to ship, it will get merged it into master and develop, then the release branch will be deleted.

Recommendation of Code Map to Environment:

- Develop branch code -> Dev, QA1 Environment
- Release branch code -> QA2 + UAT Environment
- Master branch code -> Staging, Prod Environment

Forking Branch Workflow:

Forked repositories use the standard **git clone** command.

- Forking Branch workflow gives every developer their **own server-side repository**.
- Most often seen in public **open source projects**.
- Contributions can be integrated without the need for everybody to push to a single central repository.
- Typically follows a branching model based on the GitFlow Workflow.

The following is a step-by-step example of this workflow.

1. A developer 'forks' an 'official' server-side repository. This creates their own server-side copy.
2. The new server-side copy is cloned to their local system.
3. A Git remote path for the 'official' repository is added to the local clone. (git remote add upstream <official url>
4. A new local feature branch is created.
5. The developer makes changes on the new branch.
6. New commits are created for the changes.
7. The branch gets pushed to the developer's own server-side copy.
8. The developer opens a pull request from the new branch to the 'official' repository.
9. The pull request gets approved for merge and is merged into the original server-side repository.

Mono vs Multiple Repo

	Advantages
Mono-repo - source code is kept in a single repository	• Clear ownership • Better scale • Narrow clones
Multiple-repo – each project has its own repository	• Better developer testing • Reduced code complexity • Effective code reviews • Sharing of common components • Easy refactoring

Git Hooks

- Git hooks allow you to run custom scripts whenever certain important events occur in the Git life cycle, such as committing, merging, and pushing. For example, one could have a hook into the **commit-msg** event to validate that the commit message structure follows the recommended format.

Practical use cases for using GitHooks:

- Verifying work Item Id association in your commit message.
- Preventing you & your team from committing faulty code.
- Sending notifications to your team's chat room (Teams, Slack, HipChat, etc).

The hooks can be any sort of **executable code**, including PowerShell, Python, or any other scripts. Or they may be a binary executable. The only criteria is that hooks must be stored in the .**git/hooks** folder in the repo root, and that they must be named to match the corresponding events.

Note:

If you are on Windows, simply renaming (removing extension sample) the file won't work…Git will fail to find shell in the designated path as specified in the script. The problem is in the first line of the script, the shebang declaration: ***#!/bin/sh***

Fix it by providing the path to the sh executable on your system. I'm using the 64-bit version of Git for Windows, so my shebang line looks like this: ***#!C:/Program\ Files/Git/usr/bin/sh.exe***

Edit **pre-commit**

```
#!C:/Program\ Files/Git/usr/bin/sh.exe
matches=$(git diff-index --patch HEAD | LC_ALL=en_US.utf8 grep '^+' | LC_ALL=en_US.utf8 grep -
Pi 'password|keyword2|keyword3')
echo $matches
if [ ! -z "$matches" ]
then
        cat <<\EOF
Error: Words from the blacklist were present in the diff:
EOF
        echo $matches
        exit 1
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
fi
```

Edit any file and if the new changes include the words: **password or keyword1 or keyword2** – The commit will fail

Overwrite Git Hooks:

If you ever need to overwrite the Git hooks you have set up on the client side, you can do so by using the no-verify switch:

```
git commit --no-verify
```

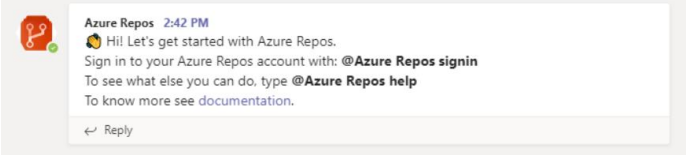
Server side GitHooks with Azure Repos

Project Settings → Service Hooks

Integration with Microsoft Teams

If you use Microsoft Teams, you can use the Azure Repos app for Teams to easily monitor your repositories. Set up and manage subscriptions to receive notifications in your channel whenever code is pushed/checked in or when a pull request (PR) is created, updated or merged.

1. Open Microsoft Teams → Add the Azure Repos app to your team in Microsoft Teams: Visit the **App store** in Microsoft Teams and search for the **Azure Repos app**. Upon installing, a welcome message from the app displays as shown in the following image. Use the @azure repos handle to start interacting with the app.



2. **Connect the Azure Repos app to your repositories:** Once the app has been installed in your team, authenticate yourself to Azure Repos using the **@azure repos signin** command. Use **Sign in with different email** if your Microsoft Teams and Azure Boards are in different tenants.
3. To start monitoring all Git repositories in a project

```
@Azure Repos subscribe https://dev.azure.com/myorg/myproject
```

4. Monitor a specific repository.

```
@Azure Repos subscribe https:// dev.azure.com/myorg/myproject/_git/myrepository
```

5. To view, add, and remove subscriptions for a channel, use the subscriptions command:

```
@azure repos subscriptions
```

For more details visit: https://docs.microsoft.com/en-us/azure/devops/repos/integrations/repos-teams

Summary of Git Commands

Git task	Notes	Git commands
Tell Git who you are	Configure the author name and email address to be used with your commits.	git config --global user.name "Sam Smith" git config --global user.email sam@example.com
Create a new local repository		git init
	Create a working copy of a local repository:	git clone /path/to/repository
	For a remote server, use:	git clone username@host:/path/to/repository
Add files	Add one or more files to staging (index):	git add <filename> git add *
	Commit changes to head (but not yet to the remote repository):	git commit -m "Commit message"
	Commit any files you've added with git add, and also commit any files you've changed:	git commit -a
	Send changes to the master branch of your remote repository:	git push origin master
	Push a specific branch to your remote repository	git push origin <branch_name>
	Push all branches to your remote repository	git push --all origin
Status	List the files you've changed and those you still need to add or commit:	git status
	Takes your uncommitted changes (both staged and unstaged), saves them away for later use, and then reverts them from your working copy.	git stash
	Popping your stash removes the changes from your stash and reapplies them to your working copy.	git stash pop

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

	You can reapply the changes to your working copy and keep them in your stash with git stash apply:	git stash apply
	If you haven't connected your local repository to a remote server, add the server to be able to push to it:	git remote add origin <server>
	List all currently configured remote repositories:	git remote -v
	Create a new branch and switch to it:	git checkout -b <branchname>
	Switch from one branch to another:	git checkout <branchname>
	List all the branches in your repo, and also tell you what branch you're currently in:	git branch
	Delete the feature branch:	git branch -d <branchname>
	Fetch and merge changes on the remote server to your working directory:	git pull
	To merge a different branch into your active branch:	git merge <branchname>
	View all the merge conflicts: View the conflicts against the base file: Preview changes, before merging:	git diff git diff --base <filename> git diff <sourcebranch> <targetbranch>
	After you have manually resolved any conflicts, you mark the changed file:	git add <filename>
	You can use tagging to mark a significant changeset, such as a release:	git tag 1.0.0 <commitID>
	CommitId is the leading characters of the changeset ID, up to 10, but must be unique. Get the ID using:	git log
	Push all tags to remote repository:	git push --tags origin
	If you mess up, you can replace the changes in your working tree with the last content in head: Changes already added to the index, as well as new files, will be kept.	git checkout -- <filename>

	Instead, to drop all your local changes and commits, fetch the latest history from the server and point your local master branch at it, do this:	git fetch origin git reset --hard origin/master

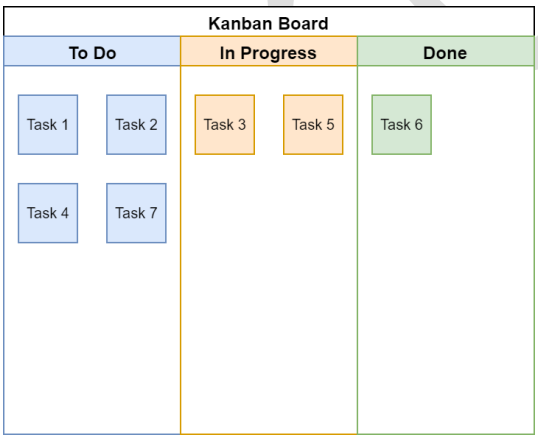
Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: Azure Boards

- Working with Work Items
- Epic, Feature, User Story, Task, Bug and Test Cases
- Linking Items
- Collaborate with Team members
- Follow a Work Item
- Dashboards
- Sprints
- List work items using Queries

Azure Boards Introduction

- Azure Boards is a tool in Azure DevOps to help teams plan the work that needs to be done.
- Essentially, Boards is your ultimate, built in agile **project planning and management tool** and has all of the functionality you would expect to find in a standalone piece of software with the ability to link work items and tasks with your pipelines.
- Team will use this tool to get a better idea of what work needs to be done and how to prioritize it.
- You can quickly and easily start **tracking user stories, backlog items, task, features, and bugs** associated with your project. You do this by adding **work items** based on the process and work item types available to your project.
- They mimic the functionality of Atlassian's Issue & Project Tracking software, **Jira**.



Boards Process Supported by Azure DevOps

In Azure DevOps, you customize your work tracking experience through a process. A process defines the building blocks of the work item tracking system as well as other sub-systems you access through Azure DevOps. Whenever you create a team project, you select the process which contains the building blocks you want for your project. The default processes differ mainly in the work item types (WITs) they provide for planning and tracking work.

a. **Basic Process:** This template is flexible for any process and great for teams getting started with Azure DevOps. Choose Basic when your team wants the simplest model that uses Epics, Issues and Tasks to track work.
b. **Agile Process:** This template is flexible and will work great for most teams using Agile planning methods, including Scrum and tracks development and test activities separately. This process works great if you want to track **user stories** and **bugs** on the Kanban board, or track bugs and tasks on the task board.
c. **Scrum Process:** Choose Scrum when your team practices Scrum. This process works great if you want to track **product backlog items** (PBIs) and **bugs** on the Kanban board, or break PBIs and bugs down into tasks on the task board.
d. **Capability Maturity Model Integration (CMMI) Process**: This template is for more formal projects requiring a framework for process improvement and an auditable record of decisions. With this process, you can track **requirements, change requests, risks, and reviews**..

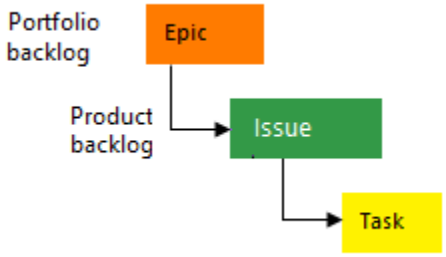
Azure DevOps supports two process types:

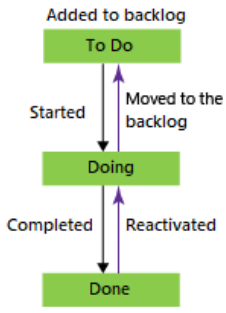
1. **Core System Processes:** You cannot customize these.
2. **Inherited Processes**: You create from a core system process. These processes you can customize.

Basic Process

Basic provides three work item types—**epics**, **issues**, and **tasks**—and a very simple workflow.

As works progresses from **not started** to **completed**, you update the State workflow field from **To Do**, **Doing**, and **Done**.





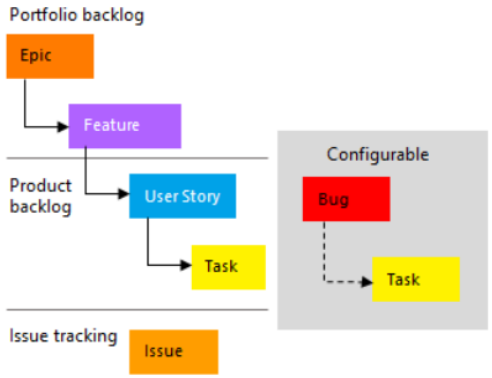
- Add **epics** to track significant features or requirements.
- Use **issues** to track user stories, bugs, or other smaller items of work.

- Use **tasks** to track even smaller amounts of work for which you want to track time either in hours or days.

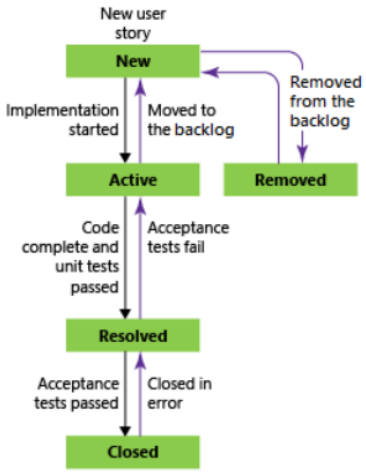
Agile Process

The Agile process provides several work item types—user stories, bugs, features, epics, and tasks. As works progresses from not started to be completed, you update the State workflow field from New, Active, Resolved, and Closed.

Work item types



Workflow



- Add **epics** to track significant business initiatives.
- Add **features** to track specific applications or set of work.
- Define **user stories** to track work that you'll assign to specific team members,
- Add **bugs** to track code defects.
- Use **tasks** to track even smaller amounts of work for which you want to track time either in hours or days.
- Use **Issues** to track obstacles to progress
- Use **Test Case** for Server-side data for a set of steps to be tested.

Try These:

Organization Settings → Boards → Process →

1. **View all Process:** Processes Tab
2. **Build-In Fields**: Fields Tab
3. **List Work Item Type:** Click on Process → Work Items types Tab

Create a new process that inherits from Agile.

1. Organization Settings → Process
2. Select Agile → click . . . → Create inherited process → Name = Custom Agile → Create process
3. Click and Open Custom Agile → Select Bug → **New Field** →
    a. **Definition Tab: Name = Bug Reason**
    b. Layout Tab: Select existing group / Create a new group
    c. Add field
4. **Switching the Process:** Organization Settings → Process → Click and Select Agile → Projects Tab → For the Selected Project → Click . . . → Change process → Select Custom Agile

Project Settings → Boards → **Team Configuration**

→ Backlog navigation levels → Check **Epic**

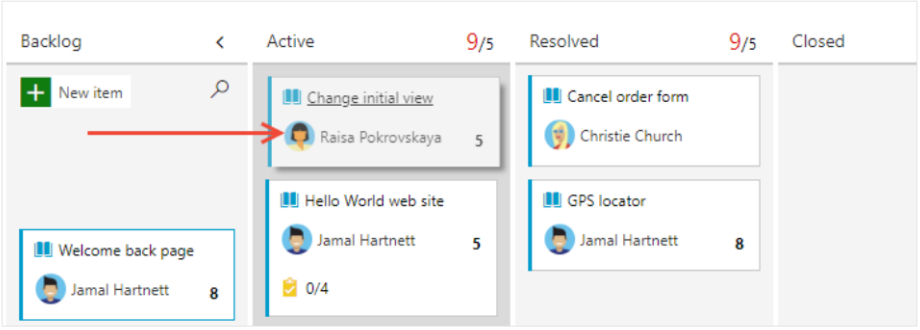
Working with Work Items

Pre-requisite:

- To add or modify work items, you must be granted **Stakeholder** access or higher.
- To view or modify work items, you must have your **View work items in this node** and **Edit work items in this node** permissions set to **Allow**. By default, the **Contributors** group has this permission set.
- To create new tags to add to work items, you must have **Basic** access or higher and have the project-level **Create new tag definition** permissions set to **Allow**. By default, the **Contributors** group has this permission set. Even if the permission is explicitly set for a **Stakeholder**, they won't have permission to add new tags, as they are prohibited through their access level.

About Kanban Board

a. You can add and update the status of work using the Kanban board.
b. You can also assign work to team members and tag with labels to support queries and filtering.
c. Share information through descriptions, attachments, or links to network shared content.
d. Prioritize work through drag-and-drop.



- We can configure Kanban boards by clicking on Gear icon.
- We can create or edit new status of the work item
- We can add or edit the column names (Active, Resolved, Closed)
- We can add new tags to the Kanban boards and specify the colors
- We can change the color codes based on tags
- We can prioritize work through drag-and-drop on your team backlog.

Sample WorkItems

Type	Title	State
Epic	Shopping Cart	Resolved
Epic	Loyalty Membership	Active
Epic	Customer Management	New
Epic	Payments	New
Epic	Vendor Management	New
Feature	Product Catalog	Resolved
Feature	Price Catalog	Active
Feature	Membership signup	New
Feature	Purchase	New
Feature	Credit Card Payments	New
Feature	Redemption	New
Feature	Loyalty points	New
Feature	Gift cards	New
Feature	Newsletter	New
Feature	Shipping & Delivery	New
User Story	As a User I want to view products from a certain category	Closed

User Story	As a User, I want to add items to my shopping cart	Active
User Story	As a User I want to view my user profile	Active
User Story	As a User I want to view product details	Active
User Story	As a User, I want to be able to modify my particulars	Active
User Story	As developer, I want to use Azure Machine Learning to provide a recommendations engine behind the website.	New
User Story	As a admin, I should be able to update prices on ad-hoc condition	New
User Story	As a user, I want to be able to update the frequency and type of newsletters to receive	New
User Story	As an admin, I want to generate coupons for customers	Resolved
User Story	As a User, I want to change quantity of items in my shopping cart	Closed
User Story	As a User I want to view my discounts coupons	Resolved
User Story	As a user, I want to choose to unsubscribe newsletters	Resolved
User Story	As a User, I should be able to select different shipping option	New
User Story	As a User, I want to remove items from my shopping cart	New
User Story	As a user, I want to subscribe to newsletters	Closed
User Story	As an admin, I want to set order limit for free shipping	New
User Story	As a User, I want to view my past orders	New
Task	As a User I want to view a public landing page with static information - JS	Closed
Task	As a User I want to see a list of popular products in the landing page - Layout	Closed
Task	As a User I want to see a list of recommended product categories in the landing page	Closed
Task	Update DB schema	Active
Task	Route to login screen if not signed in or upon timeout	New
Task	Remove item if qty is set to zero	Closed
Task	Shows the total price of all items	New
Task	Create a new route and view for cart	Active
Task	As a user, I want to choose to unsubscribe newsletters	Closed
Task	Update ARM template for deployment	Closed
Task	Create docker image	Closed

Task	Create Azure environment	Closed
Bug	The link to "about us" on the bottom of the main page missing	New
Test Case	PeakLoad-Consistent-100K-50M	Closed
Test Case	Peak Load Test - 500K users- Rampup - 30M	Ready
Test Case	Check discount is applied correctly	Design
Test Case	Check that user can select payment method	Design
Test Case	Check the delivery address details can be input	Design
Test Case	Check subtotal adds up correctly	Design
Test Case	Remove product from shopping cart	Ready
Test Case	Check display of product information, including image, is correct in shopping cart	Design
Test Case	Check user can signup for newsletters	Design
Test Case	Check that existing customers can login with username and password.	Design
Test Case	Check discount is applied correctly	Design
Test Case	Check that user can select payment method	Design
Test Case	Check the delivery address details can be input	Design
Test Case	Check subtotal adds up correctly	Design
Test Case	Remove product from shopping cart	Design
Test Case	Check display of product information, including image, is correct in shopping cart	Design
Test Case	Check user can signup for newsletters	Design
Test Case	Check that existing customers can login with username and password.	Design

Adding Work Items

Options1: Boards → **Work items → + New Work Item**

- Epic → Shopping Cart
- Epic → Loyalty Management
- Features → Product Catalog
- Features → Price Catalog
- User Story → As a User I want to see a list of popular products in the landing page – Layout
- User Story → As a User I want to view products from a certain category

Option2: Boards → **Boards**

Switch to **Epics** → +New Item

- Customer Management

Switch to **Feature** → + New Item

- Purchase
- Redemption

Switch to **Stories → + New item**

As an admin, I should be able to update prices on ad-hoc condition

Option3: Boards → **Backlogs** → Select Team →

Switch to **Epics → + New Work Item** (Add epics from above list)

- Payments

Switch to **Feature** → + New Work Item

- Loyalty points

Switch to **Stories → + New item**

- As a User, I want to add items to my shopping cart

Adding Columns in Kanban Boards

As your team updates the status of work as it progresses from one stage to the next, it helps that they agree on what **done** means. By specifying the **Definition of done** criteria for each Kanban column, you help share the essential tasks to complete before moving an item into a downstream stage.

Azure Boards → Boards → Settings (Gear Icon) → Columns → + Column

Add a **Definition of done** using markdown, such as **"Passes **all** tests."**. Click **Save and close**.

Linking Items (Parent / Child)

Click and edit Work item → **Related Work → Add link** → Existing Item

- Link type = Parent and select the parent work item
- Link type = Child and select the child work item

Sample Parent / Child Relationship

Epic: Shopping Cart

- **Features:** Product Catalog
  - **User Story:** As a User I want to view product details

        - Create an PSD of the Layout
        - Create an HTML of the Layout
        - Create database table for Product and Category
        - Write Stored Procedure
    - **User Story:** As a User I want to view products from a certain category
        - Update the existing Stored Procedure
        -
    - **User Story:** As a User I want to see a list of popular products in the landing page – Layout
        - **Task:** Create docker image
        - **Task:** Create Azure environment
- **Features:** Price Catalog
    - **User Story:** As an admin, I should be able to update prices on ad-hoc condition
    - **Bug:** The link to "about us" on the bottom of the main page missing
- **Features:** Purchase
    - **User Story:** As a User, I want to add items to my shopping cart
        - **Task:** As a User I want to view a public landing page with static information - JS
        - **Task:** As a User I want to see a list of recommended product categories in the landing page
        - **Task:** Shows the total price of all items
        - **Task:** Create a new route and view for cart
    - **User Story:** As developer, I want to use Azure Machine Learning to provide a recommendations engine behind the website.
    - **User Story:** As a User, I want to remove items from my shopping cart
    - **User Story:** As a User, I want to view my past orders
    - **User Story:** As a User, I want to change quantity of items in my shopping cart
        - **Task:** Remove item if qty is set to zero

Epic: Loyalty Management

- **Features:** Redemption
- **Features:** Loyalty points
    - **User Story:** As an admin, I want to generate coupons for customers
    - **User Story:** As a User I want to view my discounts coupons

Epic: Customer Management

- **Features:** Membership signup

- **User Story:** As a User I want to view my user profile
  - **Task:** Update DB schema
  - **Task:** Route to login screen if not signed in or upon timeout
- **User Story:** As a User, I want to be able to modify my particulars

- **Features:** Newsletter
  - **User Story:** As a user, I want to be able to update the frequency and type of newsletters to receive
  - **User Story:** As a user, I want to subscribe to newsletter
    - **Task:** As a user, I want to choose to unsubscribe newsletters

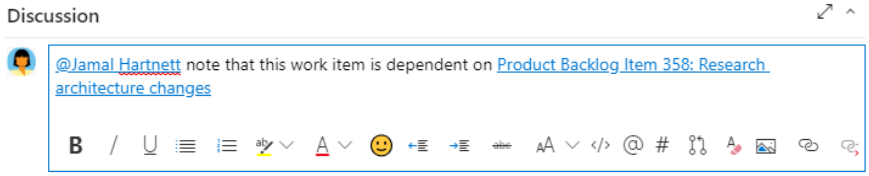
Epic: Payments

- **Features:** Credit Card Payments
- **Features:** Gift cards
- **Features:** Shipping & Delivery
  - **User Story:** As a User, I should be able to select different shipping option
  - **User Story:** As an admin, I want to set order limit for free shipping

Epic: Vendor Management

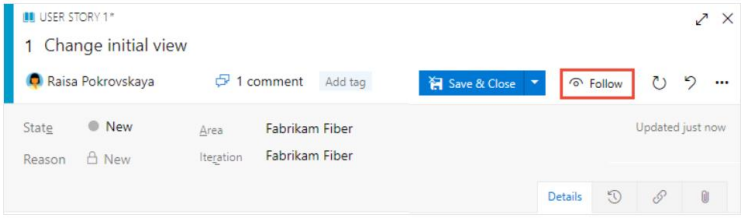
Collaborate with Team Members:

- You can collaborate with others through the **Discussion** section to add and review comments made about the work being performed.
- Use **@mentions** and **#ID** controls to quickly include others in the conversation or link to other work items. Choose to follow specific issues to get alerted when they are updated.
- You can also choose the **Attachments (icon)** tab and drag-and-drop a file to share the file with others.



Follow a Work Item

When you want to track the progress of a single work item, choose the Follow follow icon. This signals the system to notify you when changes are made to the work item.



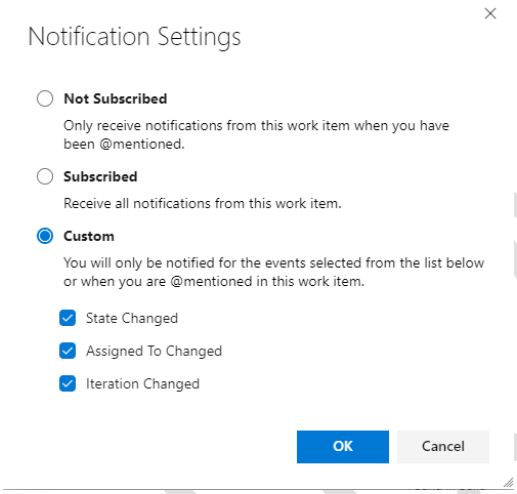
You can review and manage all the work items you've selected to follow:

Open **Boards → Queries**, choose **All**, and under **My Queries**, choose **Followed work items**.

OR

You can also view and manage work that you're following from **Boards → Work Items** and change Dropdown Following (Left most menu)

If you want to **specify conditions** on when you'll get notified of changes, choose the gear icon next to Follow on WorkItem dialog and choose from the options provided.



Capacity Planning with Sprints

Your team builds the sprint backlog during the sprint planning meeting, typically held on the first day of the sprint. Each sprint corresponds to a time-boxed interval which supports your team's ability to work using Agile processes and tools. During the planning meeting, your product owner works with your team to identify those stories or backlog items to complete in the sprint.

Planning meetings typically consist of two parts. In the first part, the team and product owner identify the backlog items that the team feels it can commit to completing in the sprint, based on experience with previous sprints. These items get added to the sprint backlog. In the second part, your team determines how it will develop and test

each item. They then define and estimate the tasks required to complete each item. Finally, your team commits to implementing some or all the items based on these estimates.

Plan sprints by assigning work to current or future sprints. Forecast work that can get completed based on effort estimates. Determine how much work can be done within a sprint. Bulk assign issues and tasks to team members and sprints.

1. In the left-side column, select **Sprints**.
2. Select **Set dates** from the upper right.
3. Leave the name as **Sprint 1**.
4. In the **Start** date field, select the calendar and pick today's date.
5. In the **End** date field, select the calendar and pick the date two weeks from today.
6. Select **Save and Close**.

Assign tasks and set the iteration

An *iteration* is another name for a sprint.

You have an initial set of work items and a timeline for your first sprint. Here you'll connect work items to your sprint and assign the tasks to yourself.

1. Under **Boards**, select **Work items**.
2. Select User Story: **As a User I want to view product details. (Under Feature: Product Catalog)**
3. In the **Iteration** drop-down list, select **Sprint 1**.
4. From the same window, select **Unassigned** and set yourself as the task owner.
5. Repeat the process for the other two work items.

Demo on Capacity Planning

Create Tasks and for each task assign Activity and Hours required

1. 5 Days Sprint
2. Two Developers each working for 8 hours = Total 80 Hours
3. One Developer is on Leave for One day
4. Task1 = 22, Task2 = 8, Task3 = 16, Task4 = 4, Task5 = 20, Task6 = 4

Note: One user can perform multiple Activities.

Dashboards

You can share progress and status with your team using configurable team dashboards.

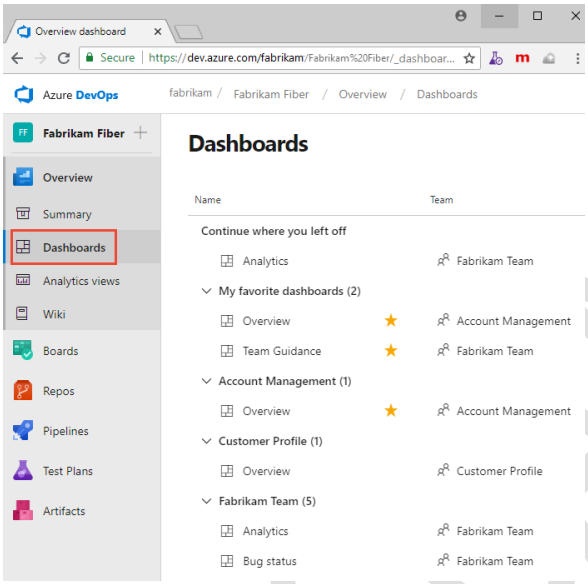
Dashboards provide easy-to-read, easy access, real-time information. At a glance, you can make informed decisions without having to drill down into other parts of your project.

The Overview page provides access to a default team dashboard which you can customize by adding, removing, or rearranging the tiles. Each tile corresponds to a widget that provides access to one or more features or functions.

All dashboards are associated with a team.

Project → Overview → **Dashboards**.

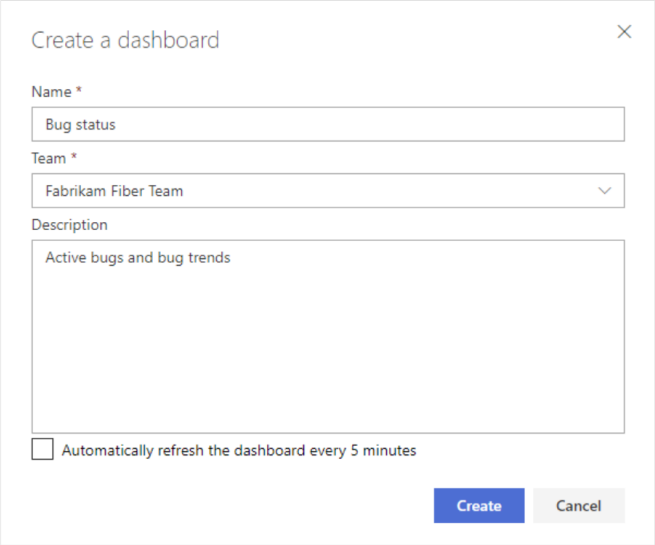
The dashboard directory page opens.



Add a dashboard

Add a new dashboard as needed to support your team's needs. You can also edit and rename any existing dashboards associated with your team.

1. From the Dashboards directory, choose **New Dashboard**. Or, when viewing a dashboard, open the selector and choose the plus icon New Dashboard option.
2. Enter the name of the dashboard and other information you want to capture.



3. The widget catalog opens. You can add one or more widgets to the dashboard. You can then configure and resize each widget as needed.
4. You can move the widgets around the dashboard to place them where you want them.

Listing Work Items using Queries

Using queries you can List bugs, user stories, or other work items based on field criteria. You can then review these lists with your team, triage work, or perform bulk work item updates.

To view, run, or email a query, you must be granted **Stakeholder** access or higher.

Task	Stakeholders	Readers	Contributors	Project admins
View and run managed queries	✓	✓	✓	✓
Create and save managed My queries	✓		✓	✓
Create and save managed Shared queries *(Stakeholders can't save Shared queries even if granted permissions)*				✓
View query charts		✓	✓	✓
Create query charts			✓	✓
Powerful semantic work-tracking search	✓	✓	✓	✓

With queries, you can perform these functions:

- List items to perform bulk updates, assign or reassign
- Review work that's in progress or recently closed

- Triage work (set priority, review, update)
- Create a chart and add it to a dashboard
- Create a chart to get a count of items or sum a field
- Create a chart that shows a burndown or burnup over time
- View a tree of parent-child related work items

My Queries, Shared Queries, and Favourites

- Only you can view and run queries that you save under **My Queries** with the queries directory. Also, you can favourite one of these queries to have it appear within your query selector.
- Queries you and others save under **Shared Queries** can be viewed by everyone with access to the project. Shared queries can be organized within folders and favourited by you or for a team. Also, you can set permissions on the folders and queries to prevent others from moving or editing them.

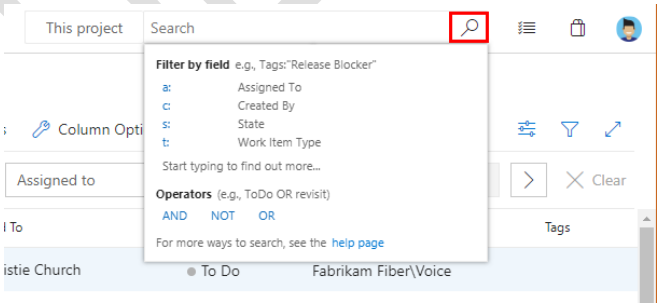
You have two ways to perform work item searches

1. **Semantic or ad hoc searches** using the work item search box

   Choose any **Boards** page, enter a keyword or phrase in the search box, and press *Enter* or choose the start search icon.

   Fine tune Semantic search results:

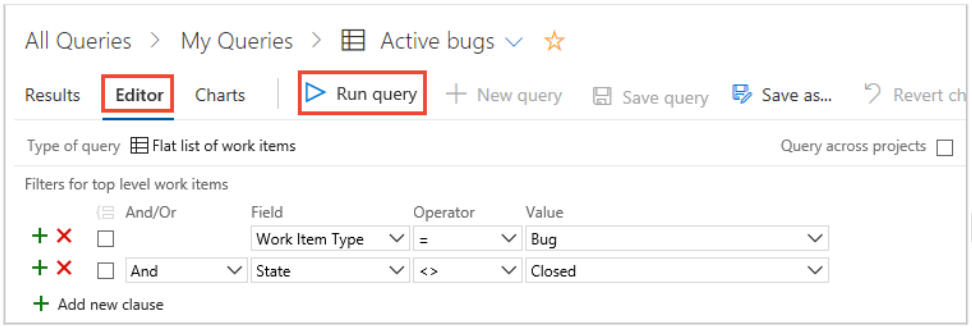
   - a: for **Assigned to:**
   - c: for **Created by:**
   - s: for **State**
   - t: for **Work item type**



Use @Me or @Today macros

2. **Managed queries** generate a list of work items based on the filter criteria you provide.

   You add and run managed queries using the built-in query-editor available from the **Boards → Queries** page.



Recommended Lab by Microsoft:

https://www.azuredevopslabs.com/labs/azuredevops/agile/

Managing Delivery Plans with Azure DevOps

https://azuredevopslabs.com/labs/azuredevops/deliveryplans/

Managing GitHub Projects with Azure DevOps

https://azuredevopslabs.com/labs/vstsextend/github-azureboards/

Which chart widgets should you recommend for each metric? To answer, drag the appropriate chart widgets to the correct metrics. Each chart widget may be used once, more than once, or not at all. You may need to drag the split bar between panes or scroll to view content. NOTE: Each correct selection is worth one point.

Chart Widgets	Answer Area	
Burndown	The elapsed time from the creation of work items to their completion:	
Cycle Time		
Lead Time	The elapsed time to complete work items once they are active:	
Velocity	The remaining work:	

Box 1: Lead time: Lead time measures the total time elapsed from the creation of work items to their completion.

Box 2: Cycle time: Cycle time measures the time it takes for your team to complete work items once they begin actively working on them.

Box 3: Burndown: Burndown charts focus on remaining work within a specific time period.

Velocity provides a useful metric for these activities:

– Support sprint planning

– Forecast future sprints and the backlog items that can be completed

– A guide for determining how well the team estimates and meets their planned commitments

Agenda: About Azure DevOps

- What is Azure DevOps
- Version History
- Azure DevOps Features
- Azure DevOps Tools and Project Life Cycle
- Create DevOps Account
- Create Organization
- Create Project and Get Started
- Create Users and invite teams members

What is Azure DevOps?

- Azure DevOps (formerly Visual Studio Team Services) is a hosted **suite of service** providing development and collaboration tools for anyone who wants an enterprise-grade DevOps tool chain.
- Azure DevOps can help your team release code in a more efficient, cooperative, and stable manner.
- Azure DevOps has a lot of **inbuilt functionality** that allows teams to get up and running with managing their project and automating their workflows to increase productivity with a very short initial learning curve.

You can quickly get up and running with the many tools available.

- Agile tools covering Kanban/scrum project methodologies
- Git repositories for source control.
- Build and Release pipelines for CI/CD automation.
- Many pre-built deployment tasks/steps to cover the most common use cases and the ability to extend this with your own tasks.
- Hosted build/release agents with ability to additionally run your own
- Custom dashboards to report on build/release and agile metrics.
- Built in wiki.

Azure DevOps is available in two different forms:

- **Azure DevOps Server**, collaboration software for software development formerly known as Team Foundation Server (TFS) and Visual Studio Team System (VSTS)
- **Azure DevOps Services**, cloud service for software development formerly known as Visual Studio Team Services and Visual Studio Online.

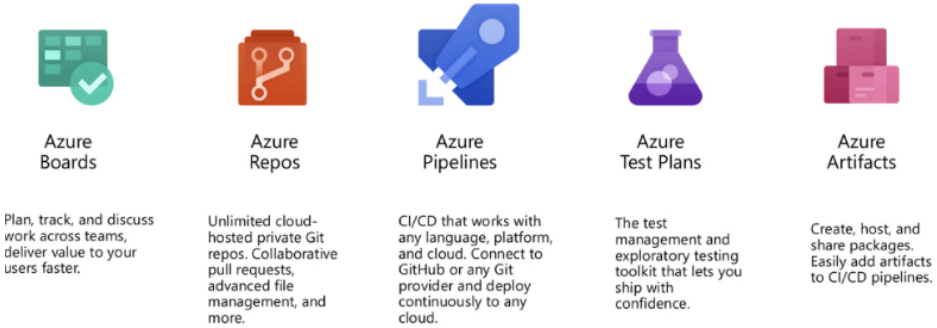
History: This first version of Team Foundation Server was released March 17, 2006.

Product name	Form	Release year
Visual Studio 2005 Team System	On-premises	2006
Visual Studio Team System 2008	On-premises	2008
Team Foundation Server 2010	On-premises	2010
Team Foundation Service Preview	Cloud	2012
Team Foundation Server 2012	On-premises	2012
Visual Studio Online	Cloud	2013
Team Foundation Server 2013	On-premises	2013
Team Foundation Server 2015	On-premises	2015
Visual Studio Team Services	Cloud	2015
Team Foundation Server 2017	On-premises	2017
Team Foundation Server 2018	On-premises	2017
Azure DevOps Services	**Cloud**	**2018**
Azure DevOps Server 2019	**On-premises**	**2019**

Azure DevOps Features

You can use one or more of the following features based on your business needs:

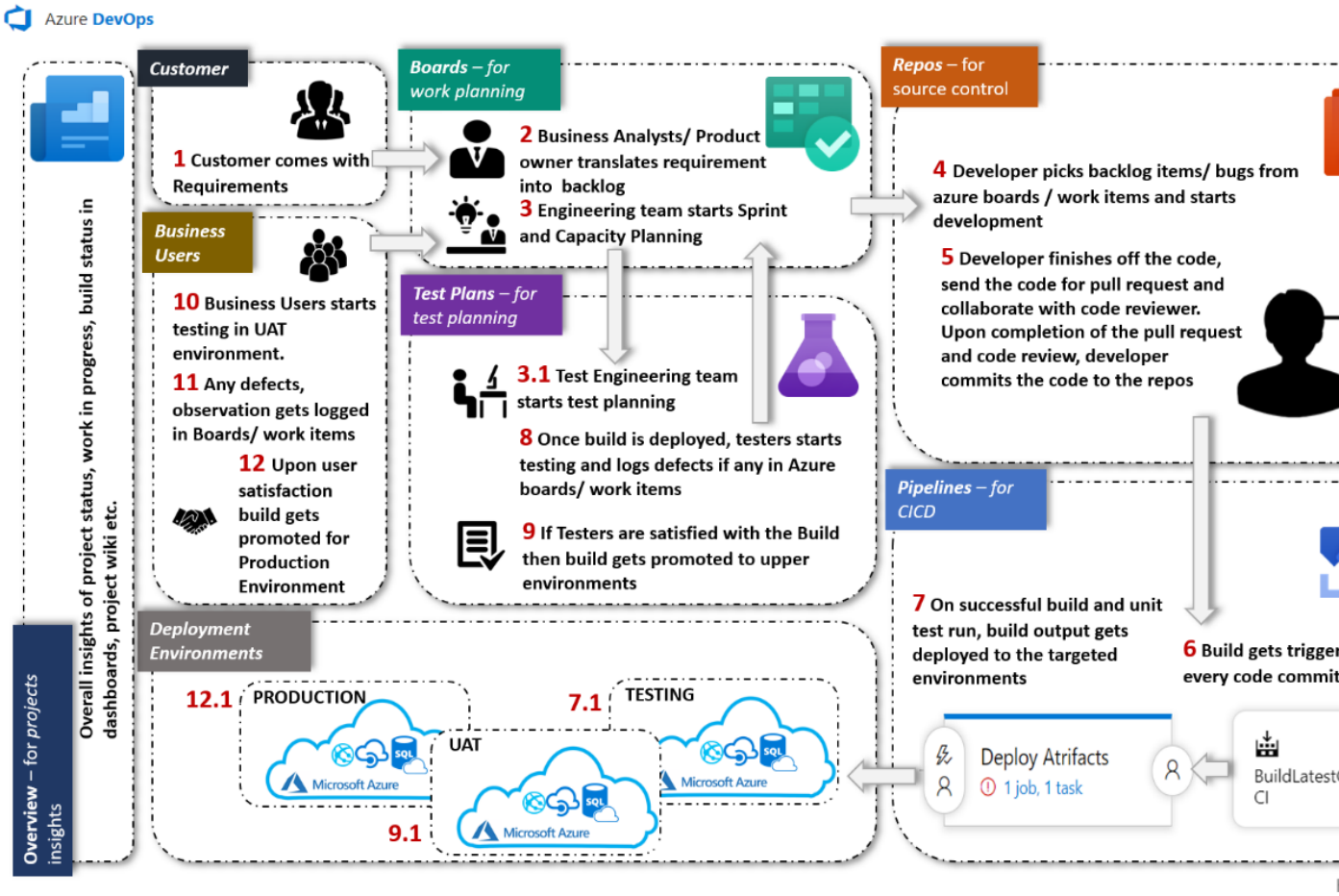
1. **Azure Boards** delivers a suite of Agile tools to support planning and tracking work, code defects, and issues using Kanban and Scrum methods.
2. **Azure Repos** provides Git repositories or Team Foundation Version Control (TFVC) for source control of your code.
3. **Azure Pipelines** provides build and release services to support continuous integration and delivery of your apps.
4. **Azure Test Plans** provides several tools to test your apps, including manual/exploratory testing and continuous testing.
5. **Azure Artifacts** allows teams to share Maven, npm, and NuGet packages from public and private sources and integrate package sharing into your CI/CD pipelines.



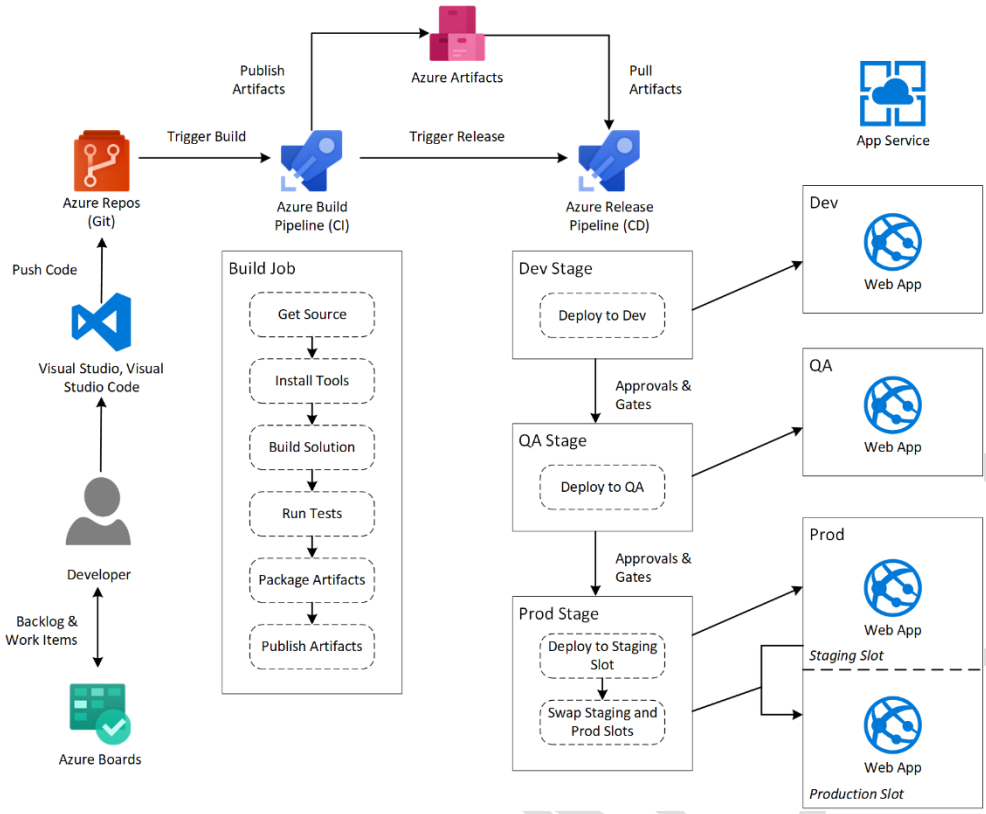
Azure DevOps supports **adding extensions** and integrating with other popular services, such as: Campfire, Slack, Trello, UserVoice, and more, and developing your own custom extensions.

Azure DevOps provides extensive integration with industry and community tools. It is far from the **closed-off single vendor** solution that was the early version of TFS. As noted above, there is a **marketplace** which makes hundreds of extensions available, so if Azure Develops doesn't do something out of the box, odds are a tool exists in the market which does.

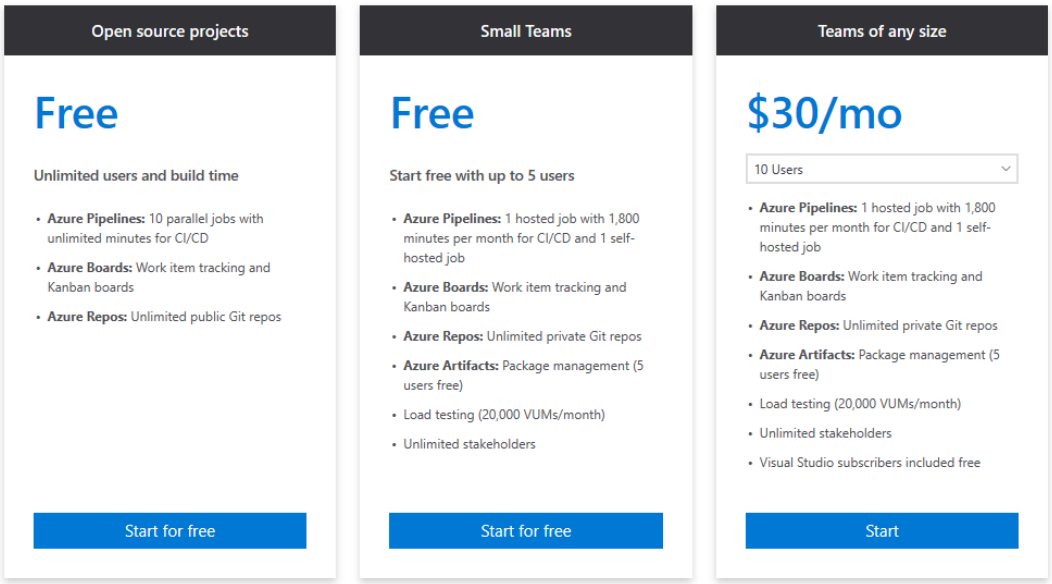
Azure DevOps Tools and Project Life Cycle



Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Designing a License Management Strategy:



https://azure.microsoft.com/en-in/pricing/details/devops/azure-devops-services/

To Create an DevOps Account / Organization

Sign up with a personal Microsoft account

1. Visit https://azure.microsoft.com/services/devops/

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

2. Click Start free
3. Either Login with Existing Microsoft Account or Create a New one.

An organization is created based on the account you used to sign in. Sign in to your organization at any time, (https://dev.azure.com/{yourorganization}).

Create a Project to get started

- If you signed up for Azure DevOps with a newly created Microsoft account (MSA), your project is automatically created and named based on your sign-in.
- If you signed up for Azure DevOps with an existing MSA or GitHub identity, you're automatically prompted to create a project. You can create either a public or private project.
- A **public GitHub repository** is accessible to everyone, whereas a **private repository** is accessible to you and the people you share it with. In both cases, only collaborators can commit changes to a GitHub repository.

Create Users

1. Create an outlook ID
2. Activate your Subscription (FREE account or Azure Pass Sponsorship or get owner rights other users subscription)
3. Visit https://portal.office.com
4. Go to Azure Active Directory and create users user1@orgname.onmicrosoft.com and user2@orgname.onmicrosoft.com

Invite team members

1. Create couple of outlook ids
2. Use primary email id and visit https://dev.azure.com/.
3. Start Free
4. Create a project to get started
    a. Project Name = "Demo Project"
    b. Description = "For Demo"
    c. Visibility = "Private"
    d. Expand Advanced, Version Control = Git, Work Item process = "Agile"
    e. Create Project
5. **Invite Users**
    a. Use bread crump and navigate to Organization
    b. Select **Organization settings**.
    c. Select **Users → Add new users**.

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

d. Users = <Email Id of another User), Access level = Basic, Add to project = <project created before>, Azure DevOps Group=Project Contributors

Note: You will have to now login as another use and accept the invitation. Do the same in different users…

Issue about Parallel Jobs:

https://devblogs.microsoft.com/devops/change-in-azure-pipelines-grant-for-private-projects/

Create DevOps Organization

==========

First create an outlook.com or any Microsoft Account (sandeepsoni@deccansoft.com)

https://dev.azure.com -> Create a Organization

Organization Settings --> Billing --> Parallel Jobs --> 0 Parallel Jobs

Visit:

https://forms.office.com/pages/responsepage.aspx?id=v4j5cvGGr0GRqy180BHbR63mUWPlq7NEsFZhkyH8jChUMlM3QzdDMFZOMkVBWU5BWFM3SDI2QlRBSC4u

To activate Your Microsoft Azure Pass Subscription

==========

Visit: https://www.microsoftazurepass.com in InPrivate Mode

Login with an outlook.com or any Microsoft Account (sandeepsoni@deccansoft.com)

***(Please ensure that you are not logged in with your organization id)

Fill in all basic details

WAIT WAIT WAIT till it automatically redirects you https://myapplication.microsoft.com or https://portal.azure.com

OR

Active FREE Trial Account

Link Azure Active Directory and DevOps Organization

==========

https://dev.azure.com/<OrganizationName> -> Organization Settings --> Azure Active Directory --> Connect directory --> Select the AD Tenant (Default Directory) --> Connect

it will logout and you login again

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Agenda: DevOps Introduction

- Traditional Software Development Life Cycle
- Waterfall Model
- About Agile Methodology.
- What is DevOps?
- DevOps Practices?
- The Challenge
- Benefits of DevOps over Traditional IT
- DevOps Tools
- What is CI and CD?
- DevOps as a profession – DevOps Engineer

Traditional Software Development Life Cycle

The **developers** create applications and the **operations teams** deploy them to an infrastructure they manage.

Responsibility of Developers

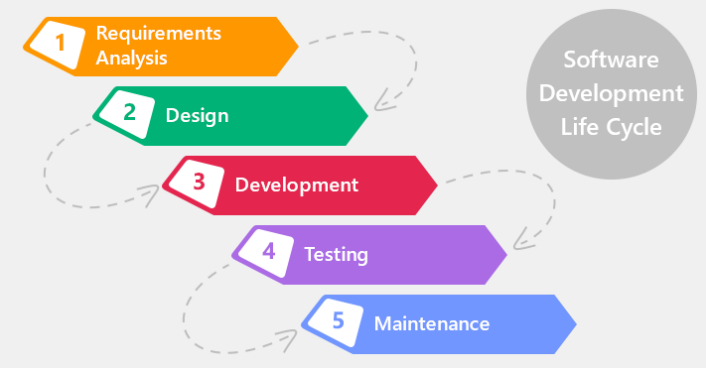
1. Develop Software Applications
2. New Features Implementation
3. Collaborate with other Developers in Team.
4. Maintain Source Repos and deal with versions.
5. Pass on the code to the operations team.

Responsibility of IT Operations

1. IT Operations determine how the software and hardware are managed.
2. Plan and Provide the required IT Infrastructure for Testing and Production of Applications.
3. Deploy the Application and Database.
4. Validate and Monitor performance.

Waterfall Model

In the below diagram you will see the phases it will involve:



Requirement Gathering stage	*During this phase, detailed requirements of the software system to be developed are gathered from client*
Design Stage	Plan the programming language, for Example Java, PHP, .net or database like MSSQL, Oracle, MySQL, etc. Or other high-level technical details of the project
Development Stage	After design stage, it is built stage, that is nothing but coding the software.
Test Stage	In this phase, you test the software to verify that it is built as per the specifications given by the client.
Deployment stage	Deploy the application in the respective environment.
Maintenance stage	Once your system is ready to use, you may later require change the code as per customer request

How the traditional Systems worked:

Tasks would be divided into different groups based on specialization

1. Group to write specification
2. Group that Develop application.
3. Group that Test the application.
4. Group to configure and manage VM.
5. Group to that hands over VM to another group to install database.
6. and so on…

A system / process is created for each action and each group operations in isolation from others. Groups communicate with each other in a very formal way, such as using ticketing system.

Drawback:

- This requires handoffs from one group to another. This can introduce significant delays, inconsistencies and inaccuracies.
- Hard to accurately define requirements, which can change over time. After delivery requires change requests and additional cost.
- Lack of a common approach among the groups contributes to the problems of long build times and errors.

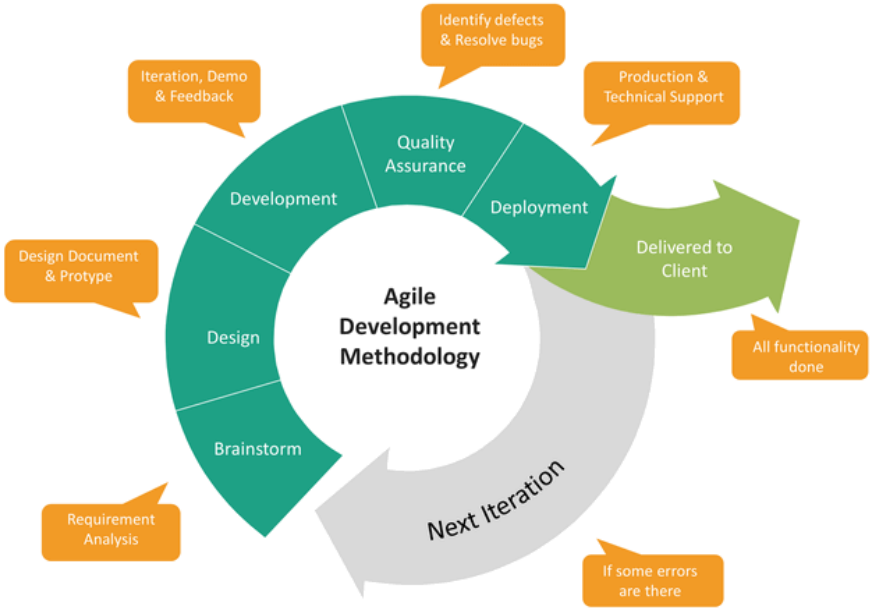
And blame game begins…

What is Agile Methodology

Agile is a process by which a team can manage a project by **breaking** it up into **several stages** and involving constant **collaboration** with stakeholders and **continuous** improvement and iteration at every stage.

- Emphasizes constantly adaptive planning, and early delivery with continual improvement.
- Development methods are based on releases and iterations.
- At the end of each iteration, there should be tested working code.
- Focused on these shorter-term outcomes.

There are no surprises. **Continuous collaboration is key**, both among team members and with project stakeholders, to make fully-informed decisions.



Principles of Agile Development

1. Satisfy the customer.
2. Welcome changing requirements.
3. Deliver working software frequently.
4. Work together throughout the project
5. Build projects around motivated individuals.
6. Use face-to-face conversation.
7. Measure progress through working software.
8. Agile processes promote sustainable development.
9. Continuous attention to technical excellence and good design
10. Simplicity--the art of minimizing the amount of work not done
11. Use self-organizing teams
12. Reflect on how to become more effective

Mentoring Team Members on Agile Practices

- Many teams hire external Agile coaches or mentors.
- Agile coaches have teaching and mentoring skills.
- Agile coaches tend to be both trainers and consultants.
- Some coaches are technical experts.
- Some coaches are focused on Agile processes.

Enabling In-Team and Cross-Team Collaboration

- **Cultural changes** – more open work spaces, meeting etiquette, outsourcing, better communication.
- **Cross-functional teams** – collaboration with others, diversity of opinion, rewarding collective behavior.
- **Collaboration tooling** – Skype, Slack, Teams, Google Meet.

What is DevOps (Development and Operations)**?**

DevOps Lessons from Formula 1

https://www.devopsgroup.com/blog/devops-lessons-formula-1-part-2/

The word "DevOps" was coined in 2009 by Patrick Debois, who became one of its gurus.

What DevOps is not

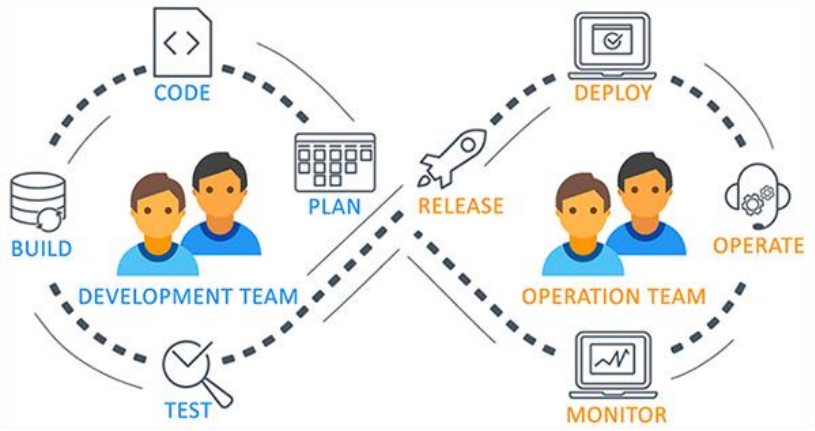
- DevOps isn't a Software
- DevOps isn't a Job Title (like developer or tester or administrator)
- DevOps isn't a process or a technology.

What is DevOps?

- DevOps is an Idea / thinking to work in a certain way that could solve the problem of Developers and IT Operation teams within the organization.
- **At Microsoft:** DevOps is the union of **people, process, and products** to enable continuous delivery of value to **our end users**.
- **DevOps is a continuous journey**

- The contraction of **"Dev" and "Ops"** refers to replacing siloed Development and Operations to create multidisciplinary teams that now work together with shared and efficient practices and tools. Sometimes, these two teams are merged into a single team where the engineers work across the entire application lifecycle.
- DevOps is a **culture** and **technical** movement that emphasizes using of **various Automation tools** for collaboration and communication of both software developers and other information-technology (IT) professionals.

How DevOps Works?

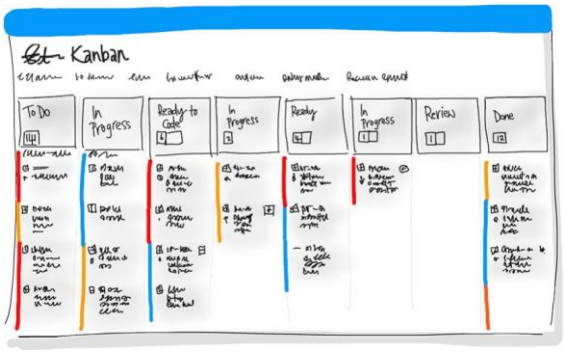


Starting from design and development to testing **automation** and from **continuous integration** to **continuous delivery**, the team works **together** to achieve the desired goal. People having both development and operations skill sets work **together** and use various tools for CI-CD and Monitoring to respond **quickly** to customers need and fix issues and bugs.

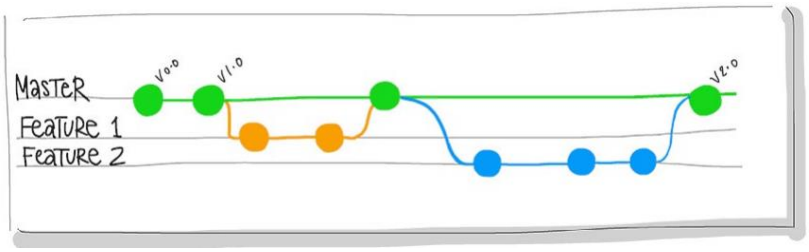
DevOps Practices

When you adopt DevOps practices, you **shorten your cycle** time by working in smaller batches, using more automation, hardening your release pipeline, improving your telemetry, and deploying more frequently.

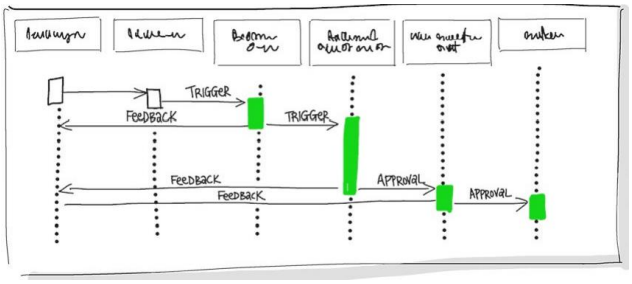
- **Agile planning**. Together, we'll create a backlog of work that everyone on the team and in management can see. We'll **prioritize** the items so we know what we need to work on first. The backlog can include user stories, bugs, and any other information that helps us.



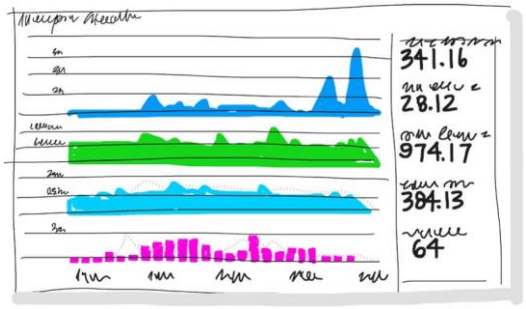
- **Version Control,** Usually With Git, enables teams located anywhere in the world to communicate effectively during daily development activities as well as to integrate with software development tools for monitoring activities such as deployments.



- **Continuous Integration (CI)**. We'll automate how we **build** and **test** our code. We'll run that every time a team member commits changes to version control. This leads to finding defects early. Other benefits include less time wasted on fighting merge issues and rapid feedback for development teams.
- **Continuous Delivery (CD)**. CD is how we test, configure, and deploy from a build to a QA or production environment.



- **Continuous Monitoring**. We'll use telemetry to get information about an application's performance and usage patterns. We can use that information to improve as we iterate.



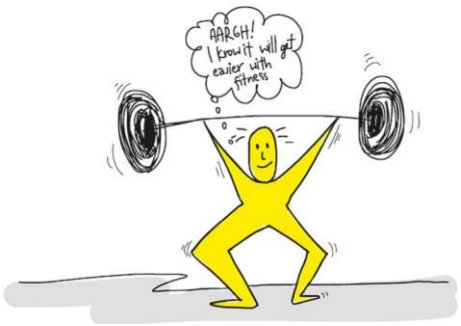
When you adopt DevOps practices, you **shorten your cycle time** by working in smaller batches, **using more automation**, hardening your release pipeline, improving your telemetry, and deploying more frequently.

The Challenge!!!

The biggest challenge is breaking down a process based on decades of rules, regulations, and frustrating areas of comfort: "It is how we have always done it; why change?"

DevOps May Hurt at First

If it hurts, do it more often. Just like going to the gym, adopting new practices is likely to hurt at first. The more often you exercise the new practices, the easier they will become. And just like training at the gym, where you exercise large muscles before small muscles, adopt practices that have the greatest impact first and cross-train to develop synergy.



There are several challenges when creating teams:

- Availability of staff.
- Disruption of current procedures.

Separating Transformation Teams

- For DevOps transformations, the **separate team** should be made up of staff members, all of whom are focused on and measured on the transformation outcomes, and **not involved in the operational day-to-day work**.
- The team might also include some **external experts** that can **fill the knowledge gaps** and help to advise on processes that are new to the existing staff members.
- Ideally the staff members who were recruited for this should already be **well-regarded** throughout the organization and as a group they should **offer a broad knowledge base** so they can think outside the box.

Ideal DevOps team members

- They think there is a **need to change** and have shown an ability to innovate.
- They are **well-respected** and have broad knowledge of the organization and how it operates.
- Ideally, they already **believe** that DevOps practices are what is needed.

Defining Shared Goals

One of the key aims of DevOps is to provide **greater customer value**, so outcomes should have a customer value focus.

Projects must have a clearly-defined set of **measurable outcomes**, like:

- Reduce the time spent on fixing bugs by 60%.
- Reduce the time spent on unplanned work by 70%.
- Reduce the out-of-hours work required by staff to no more than 10% of total working time.
- Remove all direct patching of production systems.

Measurable goals should have timelines that challenging yet achievable.

Benefits of DevOps over Traditional IT

Speed + Rapid Delivery + Reliability + Scale + Improved Collaboration + Security

With DevOps, teams:

1. **Deploy more frequently**

   In fact, some teams deploy up to dozens of times per day.

   Practices such as monitoring, continuous testing, database change management, and integrating security earlier in the software development process help elite performers deploy more frequently, and with greater predictability and security.

2. **Reduce lead time from commit to deploy**

   Lead time is the time it takes for a feature to make it to the customer. By working in smaller batches, automating manual processes, and deploying more frequently, elite performers can achieve in hours or days what once took weeks or even months.

3. **Reduce change failure rate**

   A new feature that fails in production or that causes other features to break can create a lost opportunity between you and your users. As high-performing teams mature, they reduce their change failure rate over time.

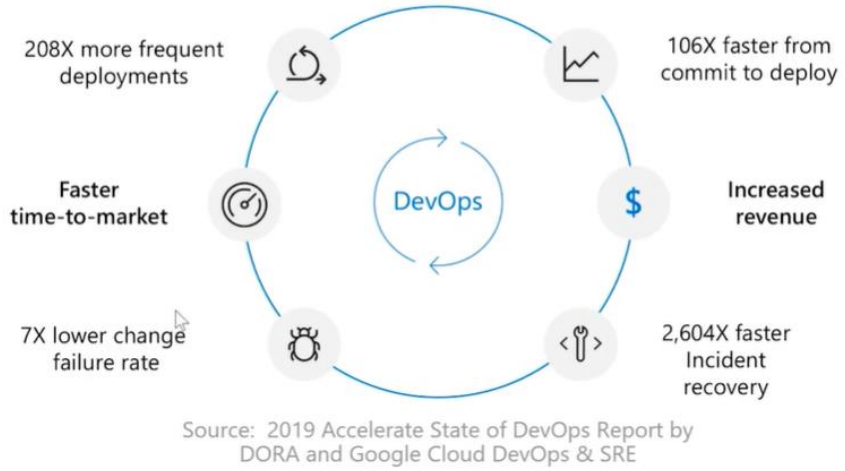
4. **Recover from incidents more quickly**

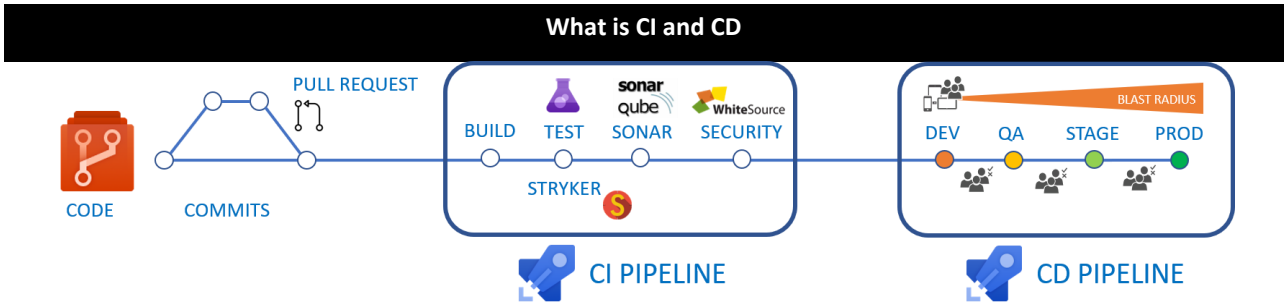
   When incidents do occur, elite performers are able to recover more quickly. Acting on metrics helps elite performers recover more quickly while also deploying more frequently.

How you implement cloud infrastructure also matters. The cloud improves software delivery performance, and teams that adopt essential cloud characteristics are more likely to become elite performers.

The information here is based on DevOps research reports and surveys conducted with technical professionals worldwide.

Organizations that have implemented DevOps saw these benefits:





- **Continuous integration** is a software development practice where developers regularly merge their code changes into a central repository, after which automated builds and tests are run. The key goals of continuous integration are to find and address bugs quicker, improve software quality, and reduce the time it takes to validate and release new software updates.
- **Continuous delivery** is a software development practice where code changes are automatically built, tested, and prepared for a release to production. It expands upon continuous integration by deploying all code changes to a testing environment and/or a production environment after the build stage. When continuous delivery is implemented properly, developers will always have a deployment-ready build artifact that has passed through a standardized test process.

CI and CD enables agile teams to **increase deployment frequency** and **decrease lead time for change**, change-failure rate, and mean time to recovery key performance indicators (KPIs), thereby **improving quality and delivering value faster**. The only prerequisites are a solid development process, a mindset for quality and accountability for features from ideation to deprecation, and a comprehensive pipeline.

Release Management Tools:

- Azure DevOps Pipelines
- Jenkins
- Circle CI

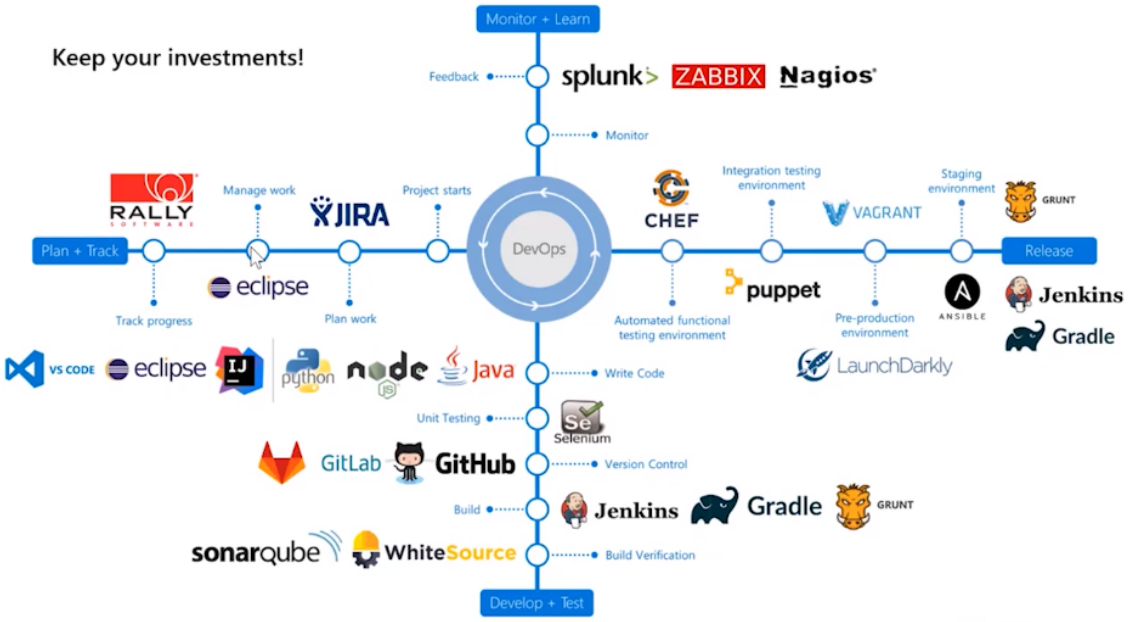
- GitLab Pipelines
- Atlassian Bamboo
- XL Deploy/XL Release

DevOps Tools

To expedite and actualize DevOps process apart from culturally accepting it, one also needs various DevOps tools like Puppet, Jenkins, GIT, Chef, Docker, Selenium, Azure/AWS etc to achieve automation at various stages which helps in achieving **Continuous Development, Continuous Integration, Continuous Testing, Continuous Deployment, Continuous Monitoring** to deliver a quality software to the customer at a very fast pace.

Tools Required for

- Version Control System
- Configuration management
- Ticketing System
- Resource Monitoring
- Provisioning



DevOps as a profession - DevOps Engineer

When the company's management decides to shift to DevOps, the need arises to train IT department specialists to master certain practices and use new tools. In this case, either developers or system administrators need to assume new job responsibilities.

A better alternative may be hiring a professional with a clear understanding of the DevOps approach and an ability to set all the necessary processes properly.

After getting software requirements specifications, a DevOps engineer starts setting up the IT infrastructure required for the development. When the IT infrastructure is ready and provided to developers, testers, and other specialists involved in the development cycle, a DevOps engineer ensures that the development and testing environments are aligned with the production environment.

If you ask the DevOps engineer what exactly they do, the answer will likely mention *"**automation**."* What they actually mean is the following:

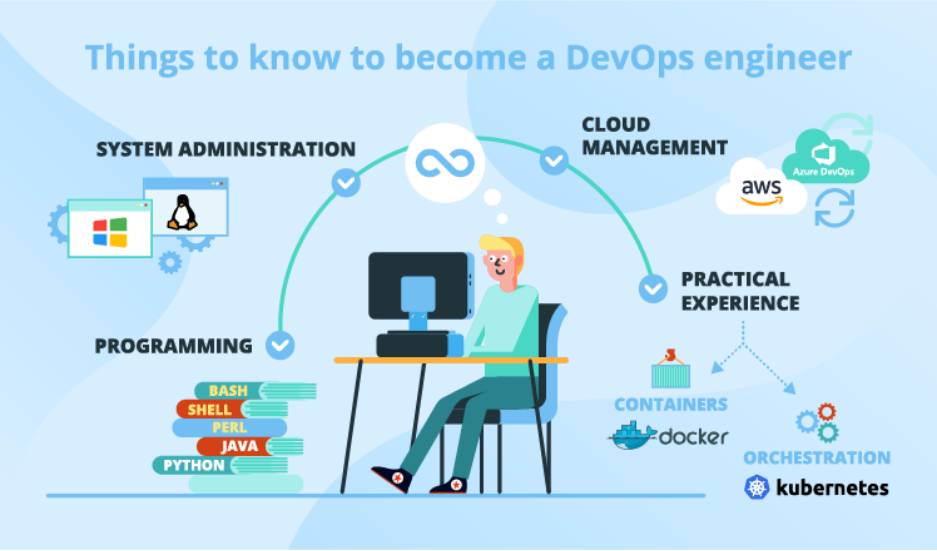
- Automating software delivery from the testing environment to the production.
- Managing physical and virtual servers and their configurations.
- Monitoring the IT infrastructure's state and the application's behavior.

Things to know to become a DevOps engineer

1. Linux and/or Windows Administrator
2. Programming Skills
3. Cloud Management Skills
4. Practical Experience with Containers and orchestration.

Below is a more comprehensive list of tools most commonly required to get a job in DevOps:

- Version control systems (**Git**, Team Foundation Server (TFS), Apache Subversion, etc.).
- Continuous integration tools (**Azure DevOps, Jenkins**, Travis CI, TeamCity, and others).
- Software deployment automation platforms (Chef, Puppet, **Ansible**, etc.).
- Systems monitoring tools (Grafana, Zabbix, **Prometheus**, and the like).
- Cloud infrastructure (**AWS, Microsoft Azure**, Google Cloud Platform (GCP), Alibaba Cloud, and more).
- Container orchestration tools (such as **Kubernetes**, Docker Swarm, Apache Mesos, OpenShift, Google Kubernetes Engine (GKE), Azure Kubernetes Service (AKS), AWS EKS).



DevOps engineers' average salary in the US is twice as high as that of a system administrator. The reason is quite simple — a competent DevOps engineer can greatly increase the efficiency of software development and operations.

More: https://www.scnsoft.com/blog/how-to-become-a-devops-engineer

Agenda: Kubernetes

- About Kubernetes
- Azure Kubernetes Service (AKS)
- Deploy Service to AKS Cluster.

What is Kubernetes

Containerization has brought a lot of flexibility for developers in terms of managing the deployment of the applications. However, the more granular the application is, the more components it consists of and hence requires some sort of management for those.

Nearly all applications nowadays need to have answers for things like

- Replication of components
- Auto-scaling
- Load balancing
- Rolling updates
- Logging across components
- Monitoring and health checking
- Service discovery
- Authentication

One still needs to take care of scheduling the deployment of a certain number of containers to a specific node, managing networking between the containers, following the resource allocation, moving them around as they grow and much more.

The process of organizing multiple **containers and managing them as needed** is known as **container orchestration**.

Kubernetes, a container orchestration technology used to orchestrate the deployment and management of hundreds and thousands of containers in clustered environment.

- Kubernetes is an **open source project** was started by Google in the year 2014. **Redhat** is the second major contributor along with Microsoft, HP, VMWare etc…
- The name **Kubernetes** originates from Greek, meaning *helmsman (who steers ship or boat)* or *pilot*, and is the root of *governor* and cybernetic (theory or study of communication and control). **K8s** is an abbreviation derived by replacing the 8 letters “ubernete” with “8”.

- It is a **platform** designed to completely manage the life cycle of **containerized applications and services** using methods that provide **predictability, scalability, and high availability**.
- It orchestrates **computing**, **networking**, and **storage** infrastructure on behalf of user workloads.
- It is easier to **deploy**, **scale**, and **manage** applications.
- Kubernetes aims to support an extremely diverse variety of workloads, including **stateless**, **stateful**, and **data-processing** workloads. If an application can **run in a container**, it should run great on Kubernetes.

Kubernetes is:

- a container platform.
- a microservices platform
- a portable cloud platform.

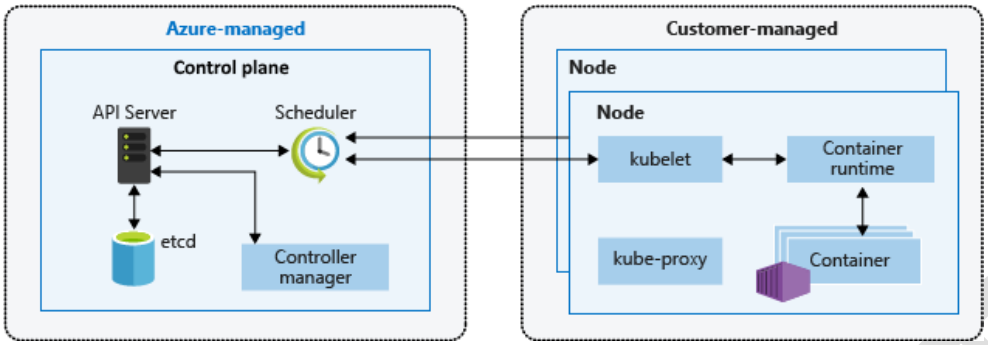
Following are some of the important features of Kubernetes.

- Simplifies Application Deployment
- Continues development, integration and deployment
- Application-centric management and better hardware utilization.
- Modern tooling, have CLI and management REST API support.
- Highly Scalable infrastructure with Autoscaling support
- Kubernetes Monitors Health of each node and does self-healing of docker containers.
- Rolling updates

Azure Kubernetes Service (AKS)

- AKS makes it quick and easy to deploy and manage containerized applications without container orchestration expertise.
- AKS reduces the complexity and operational overhead of managing Kubernetes by offloading much of that responsibility to Azure.
- As a hosted Kubernetes service, Azure handles critical tasks like provisioning, upgrading, scaling, health monitoring and maintenance for you.
- In addition, the service is free, you only pay for the agent nodes within your clusters, not for the masters.
- Master node(s) is managed by AKS

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)



Service quotas and limits

Resource	Default Limit
Max node pools per cluster	100
Max nodes per cluster (Standard LB)	1000
Max pods per node (basic networking with Kubenet)	110
Max pods per node (advanced networking with Azure CNI)	301
Max cluster per subscription	5000

Walkthrough

1. Package your Application in to one or more containers
2. Push those images to an image registry like Docker Hub or ACR
3. Post App Descriptor to the Kubernetes API Server

Walkthrough:

Step 1: Develop and Test the application locally

Step2: Deploy the docker images in ACR

Step3: Create Kubernetes Cluster

Step4: Run the application developed in step 1

Step 1: Develop and Test the application locally

1. Execute the following commands to create an ASP.NET Core MVC project.

```
D:\>md HelloWebApp
D:\>cd HelloWebApp
D:\HelloWebApp:\> dotnet new mvc
D:\HelloWebApp:\> Code .
```

2. Add the following docker file to the project

```
FROM mcr.microsoft.com/dotnet/aspnet:6.0 AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443

FROM mcr.microsoft.com/dotnet/sdk:6.0 AS build
WORKDIR /src
COPY ["HelloWebApp.csproj", ""]
RUN dotnet restore "./HelloWebApp.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "HelloWebApp.csproj" -c Release -o /app/build

FROM build AS publish
RUN dotnet publish "HelloWebApp.csproj" -c Release -o /app/publish

FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "HelloWebApp.dll"]
```

3. Build the docker image

```
D:\HelloWebApp>docker build -t sandeepsoni/hellowebapp .
```

4. Test the image locally

```
D:\HelloWebApp>docker run -p  "8080:80" sandeepsoni/hellowebapp
```

Visit: http://localhost:8080

Step2: Create Container Registry and Push the Image

5. Login to Azure

```
az login
az account set --subscription "Azure Training - SS1"
```

6. Create Resource Group

```
az group create --name DemoKRG --location eastus
```

7. Create Azure Container Registry

```
az acr create --resource-group DemoKRG --name dssregistry --sku Standard --location eastus
```

8. Goto ACR → Under **Admin user**, select **Enable**. Take note of the following values:
   - Login server
   - Username
   - password
9. Local Machine: Login to ACR

```
docker login --username dssregistry --password oZ2+N4QW+DmhwDvLe5pDFF/9oMB0PS/r

dssregistry.azurecr.io
```

10. Tag Local Images

```
docker image tag sandeepsoni/hellowebapp dssregistry.azurecr.io/hellowebapp
```

11. Push images to ACR

```
docker push dssregistry.azurecr.io/hellowebapp
```

Step3: Create Service Principal and Assign AcrPull Role over the Container Registry

12. Create Service Principal in Azure Active Directory.

```
Azure Active Directory → App Registration → + Register App

Name = KubernetesServicePrincipal,

OK.

Note the service principal ClientID and Secret.
```

OR

```
az ad sp create-for-rbac --skip-assignment

output

{
  "appId": "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4",
  "displayName": "azure-cli-2018-09-18-10-52-15",
  "name": "http://azure-cli-2018-09-18-10-52-15",
  "password": "ffb12981-f5f1-44d4-8fa1-e3b0aa609936",
  "tenant": "82d8af3b-d3f9-465c-b724-0fb186cc28c7"
}
```

13. **Assign Role AcrPull to Service Principal to read images from Azure Container Registry**

    Go to Azure Contaner Registry → Access Control (IAM) → Add Role Assignment

Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Role = AcrPull

Service Principal = KubernetesServicePrincipal

Assign.

OR

1. Get the ACR Id

az acr show --resource-group DemoKRG --name dssregistry --query "id" --output tsv

output:

/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry

2. Assign contributor Role to Service Principal in Azure AD

az role assignment create --assignee "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --scope "/subscriptions/24784a25-4b3b-4fbe-bd67-045821454fda/resourceGroups/DemoKRG/providers/Microsoft.ContainerRegistry/registries/dssregistry" **--role AcrPull**

Step3: Create Azure Kubernetes (AKS) Cluster

14. Create **Azure Kubernetes Service** with associated **Service Principal**

az aks create --resource-group DemoKRG --name **dssdemocluster** --node-count 1 --generate-ssh-keys --service-principal "c2a21be3-f9bc-4234-ae43-49cedf8a3cb4" --client-secret "ffb12981-f5f1-44d4-8fa1-e3b0aa609936"

OR

PS: New-AzAksCluster -ResourceGroupName DemoKRG -Name dssdemocluster -NodeCount 2 -GenerateSshKey

Note: Delete key related files in this folder if existing: C:\Users\<Current User>\.ssh

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

15. Connect to Kubernetes Cluster

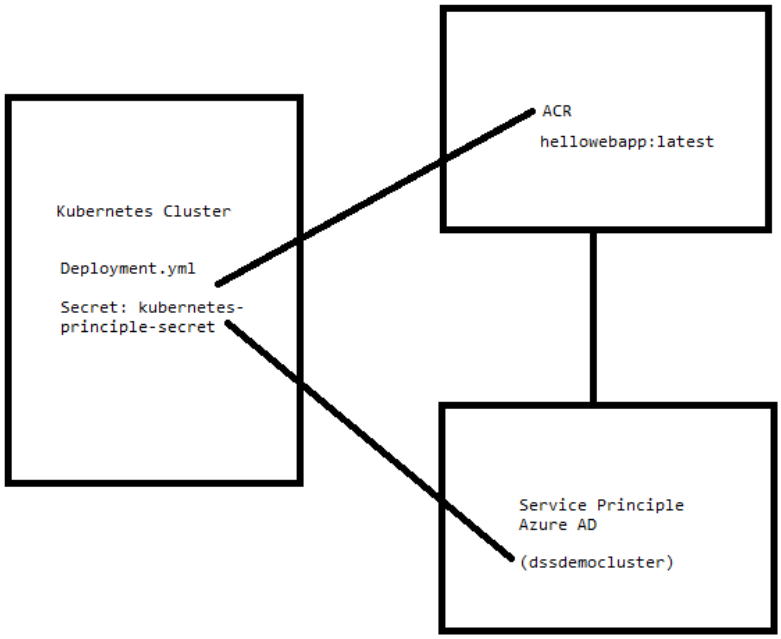
```
az aks get-credentials --resource-group DemoKRG --name dssdemocluster
```

16. Configure Kubernetes to use your ACR

When creating deployments, replicasets or pods, kubernetes will try to use docker images already stored locally or pull them from the public docker hub. To change this, you need to specify the custom docker registry as part of your k8s object configuration (yaml or json).

Instead of specifying this directly in your configuration, we'll use the concept of k8s **secrets**. You decouple the k8s object from the registry configuration by just referencing the secret by it's name. But first, let's create a new k8s secret.

```
kubectl create secret docker-registry kubernetes-principle-secret --docker-server dssregistry.azurecr.io --docker-username="<Service Principal CLIENTID>" --docker-password "<Service Pricipal Secret>"
```



Kubernetes manifest files define a desired state for a cluster, including what container images should be running.

17. Create a file DeploymentAndService.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
 name: hellowebapp
 labels:
  app: hellowebapp
spec:
 replicas: 1
 selector:
   matchLabels:
    app: hellowebapp
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

```
 template:
  metadata:
   labels:
    app: hellowebapp
  spec:
   containers:
   - name: hellowebapp
    image: dssdemo.azurecr.io/hellowebapp:latest
    imagePullPolicy: Always
    ports:
    - containerPort: 80
   imagePullSecrets:
   - name: kubernetes-principle-secret


---


apiVersion: v1
kind: Service
metadata:
 name: hellowebapp
spec:
 type: LoadBalancer
 ports:
 - port: 80
 selector:
  app: hellowebapp
```

18. Use the kubectl create command to run the application.

```
kubectl apply -f deployment.yaml
```

Test the application

19. To monitor progress, use the the below command.

```
kubectl get service azure-vote-front –watch
```


Deccansoft Software Services H.No: 153, A/4, Balamrai, Rasoolpura, Secunderabad-500003 TELANGANA, NDIA. http://www.deccansoft.com | http://www.bestdotnettraining.com
Phone: +91 40 2784 1517 OR +91 8008327000 (INDIA)

Once the *EXTERNAL-IP* address has changed from *pending* to an *IP address*, use `CTRL-C` to stop the kubectl watch process.

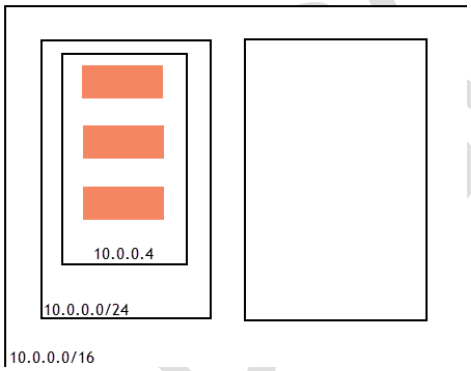
20. Open a web browser to the external IP address of your service

AKS Networking and Node Pools

Types of Networks:

- **Kubenet:** By default, AKS clusters use kubenet, and an Azure virtual network and subnet are created for you. With *kubenet*, nodes get an IP address from the Azure virtual network subnet. Pods receive an IP address from a logically different address space to the Azure virtual network subnet of the nodes. Network address translation (NAT) is then configured so that the pods can reach resources on the Azure virtual network. The source IP address of the traffic is NAT'd to the node's primary IP address. This approach greatly reduces the number of IP addresses that you need to reserve in your network space for pods to use.
- **Azure Container Networking Interface (CNI):** Every pod gets an IP address from the subnet and can be accessed directly. Each node has a configuration parameter for the maximum number of pods that it supports. The equivalent number of IP addresses per node are then reserved up front for that node. This approach requires more planning, and often leads to **IP address exhaustion** or the need to rebuild clusters in a larger subnet as your application demands grow.

Note: To use Windows Server node pools, you must use Azure CNI. The use of kubenet as the network model is not available for Windows Server containers.



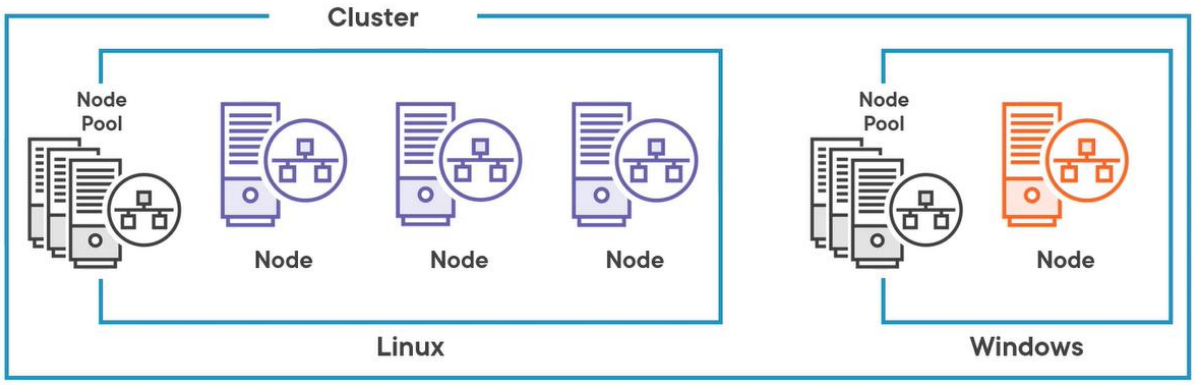
About AKS Node Pools

In Azure Kubernetes Service (AKS), nodes of the same configuration are grouped together into *node pools*.

- **System node pools:** They serve the primary purpose of hosting **critical system pods** such as CoreDNS and tunnelfront on underlying VMs. At least **one system node** pool is required for a cluster. The size of the VMs cannot be changed after creating the cluster. These are Linux Nodes only.

- **User Node Pool:** To support applications that have different compute or storage demands, you can create additional User node pools. User node pools are where you place your application-specific pods. Can be Windows and Linux both.

Note: Application pods can be scheduled on system node pools if you wish to only have one pool in your AKS cluster.



Upgrade an AKS cluster

To check which Kubernetes releases are available for upgrade.

```
az aks get-upgrades --name dssdemocluster --resource-group DemoRG --output table
```

Output:

Name	ResourceGroup	MasterVersion	NodePoolVersion	Upgrades
default	DemoRG	1.8.10	1.8.10	1.9.1, 1.9.2, 1.9.6

We have three versions available for upgrade: 1.9.1, 1.9.2 and 1.9.6.

To Upgrade:

```
az aks upgrade --name dssdemocluster --resource-group DemoRG --kubernetes-version 1.9.6
```

When upgrading an AKS cluster, Kubernetes minor versions cannot be skipped. For example, upgrades between 1.8.x -> 1.9.x or 1.9.x -> 1.10.x are allowed, however 1.8 -> 1.10 is not. To upgrade, from 1.8 -> 1.10, you need to upgrade first from 1.8 -> 1.9 and then another do another upgrade from 1.9 -> 1.10

During the upgrade process, AKS will add a new node to the cluster, then carefully cordon and drain one node at a time to minimize disruption to running applications