Webben vNext

Lab 2 – Azure och Web Deployment

# Intro

In this you will learn how to use Azure Blob Storage as the backend storage for your application, and you will also learn how to create and deploy Azure Web Apps using Visual Studio Team Services.

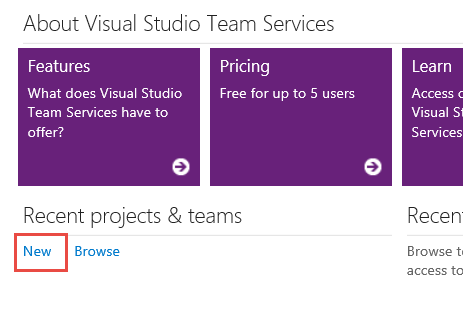
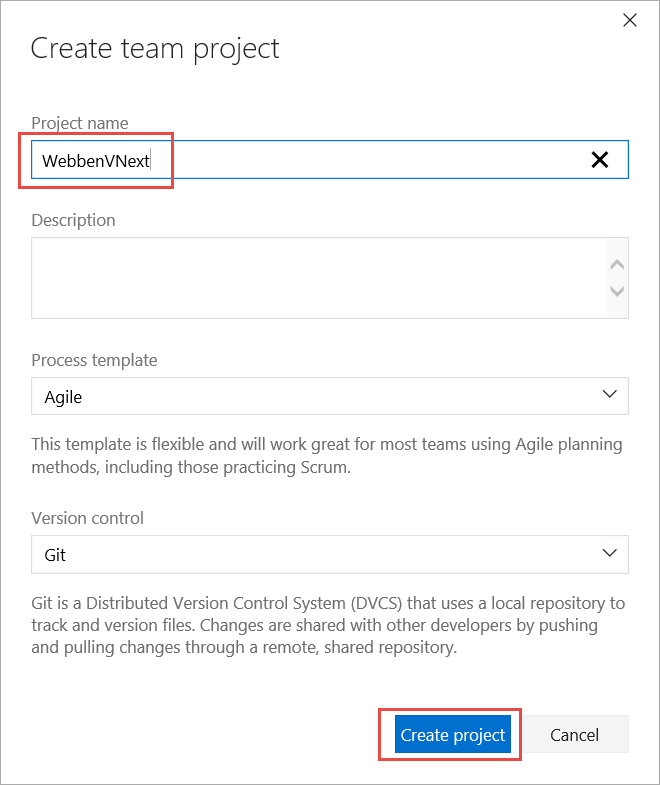
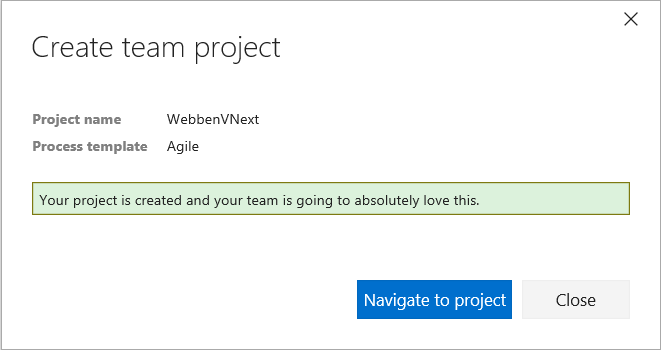
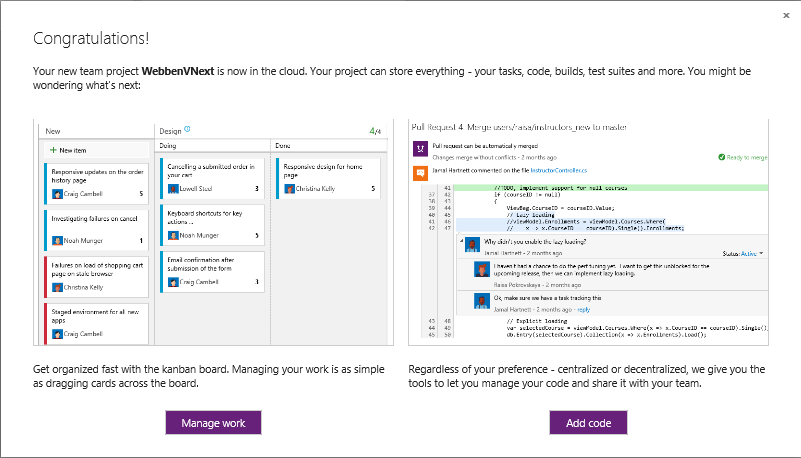
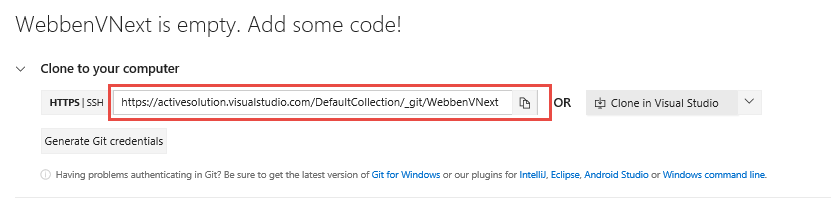
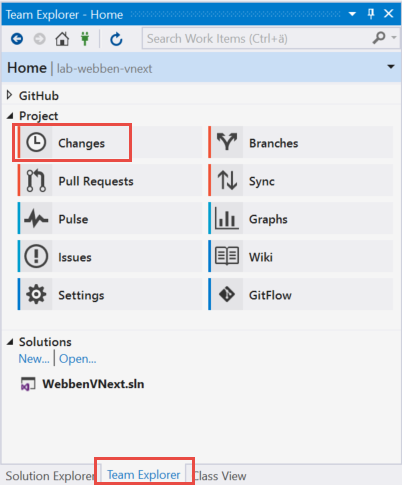
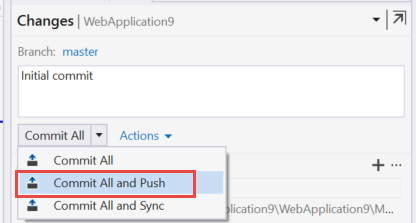
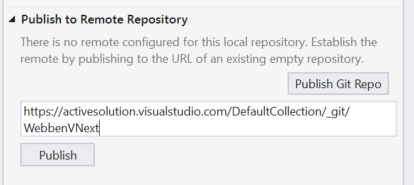
Techniques and concepts you will use:

* Visual Studio Team Services (VSTS)
* Azure Web App
* Azure Blob Storage
* Azure App Service Plans
* Create a build- and release pipeline using VSTS

# Lab Instructions

## Create a new Team Project in VSTS

In this step you will create a new team project where you will store your source code. You will later in this lab also create a release pipeline that will build and deploy your web application to Microsoft Azure.

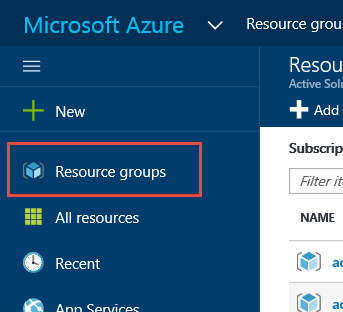
1. Connect to your Visual Studio Team Services Account (https://<youraccount>.visualstudio.com)
2. On the home screen, locate the **Recent projects & teams** section and click on **New**  
     
   
3. Give the team project a name, and make sure that you select **Git** as version control:  
     
   
4. When the project is created, navigate to the new team project:  
     
   
5. Press **Add Code** to navigate straight to the Code hub in VSTS  
     
   
6. In the repository screen, copy the Git URL for the repository:  
     
   
7. You will now push your local project to the Git repository in VSTS. In Visual Studio, open Team Explorer and navigate to the Changes section:  
     
   
8. Enter a commit message and select **Commit All and Push**  
     
   
9. In the **Publish to Remote Repository** section, paste the Git URL and select **Publish**:  
     
   
10. Your source code is now safely stored in VSTS!

## Create Web Application resources in Azure

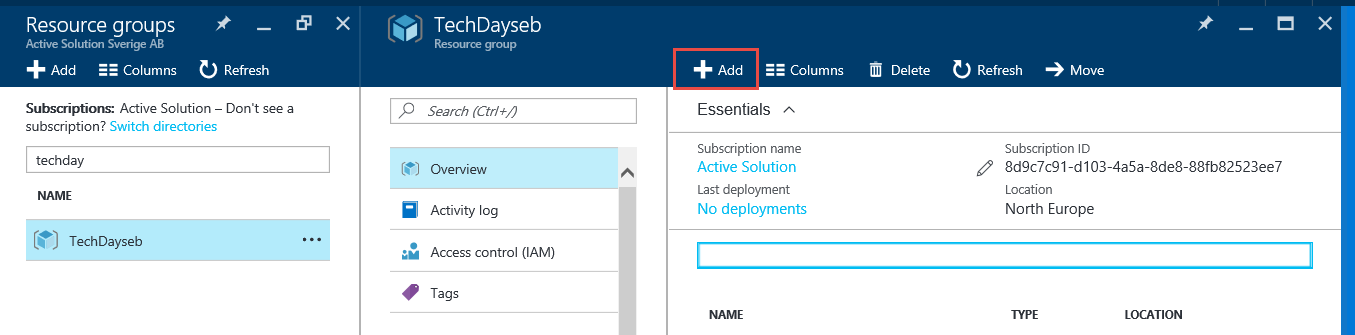
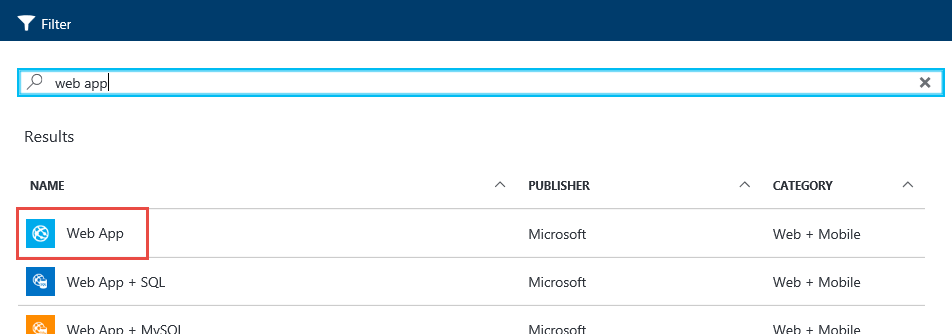
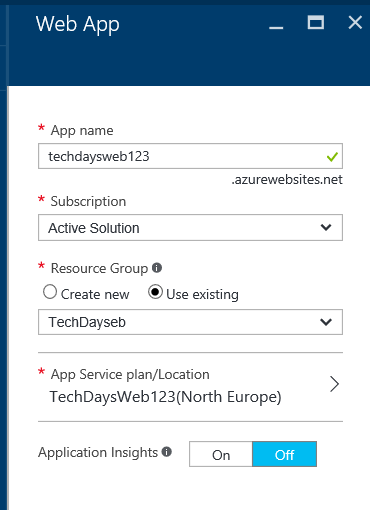
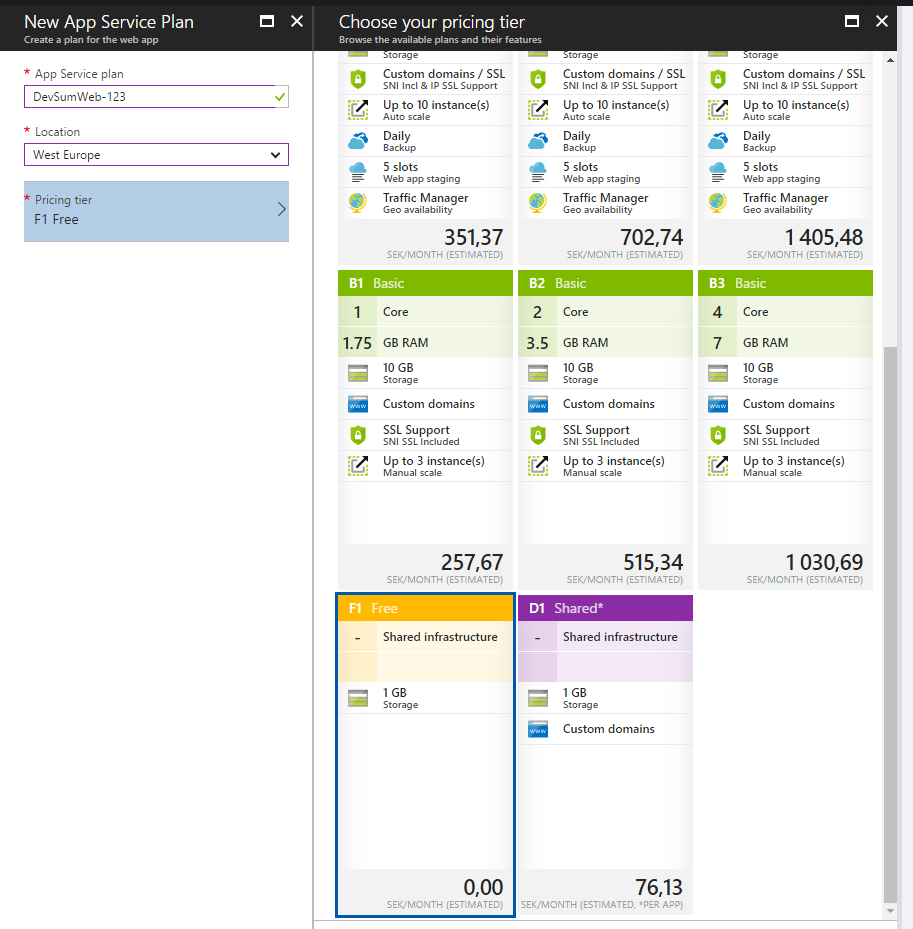
For this lab we will create a couple of resources in Azure:

* A resource group that will contain the other resources
* An Azure Web App where we will deploy our application to
* A Storage Account where we will store the images

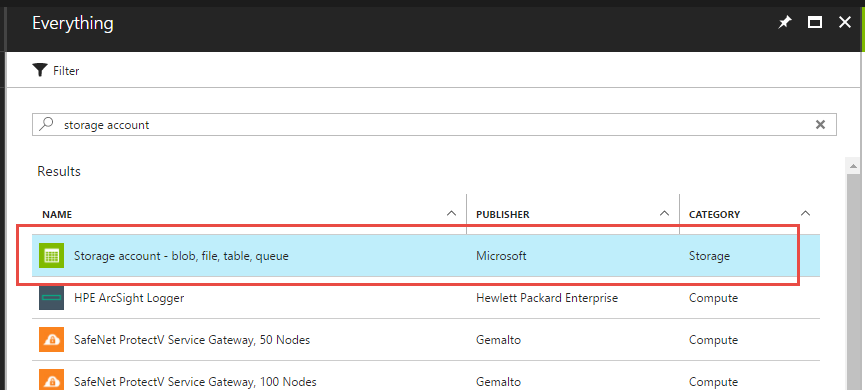
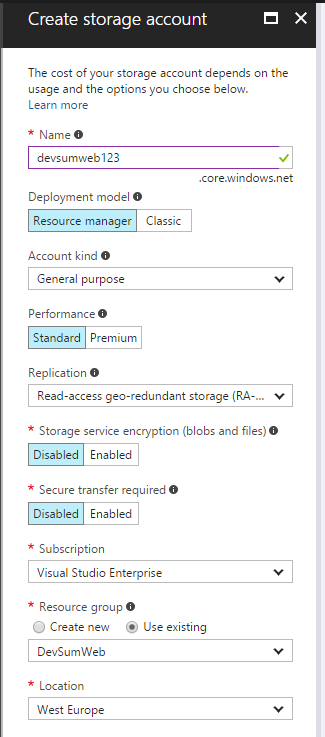
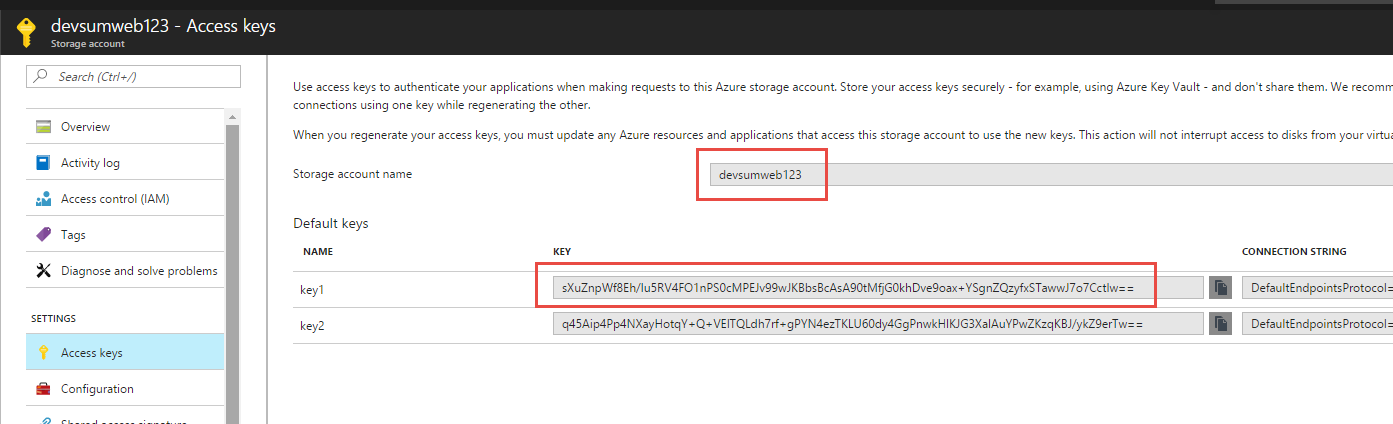
### Create the resource group

1. Log into the Azure Portal (<https://portal.azure.com>)
2. Select the **Resource groups** node to the left to show any existing resource groups  
     
   
3. Click the **+Add** button
4. Enter a name for the resource group (for example DevSumWeb)
5. Select **West Europe** as the **Resource group** location
6. Click **Create** to create the new resource group

### Create an Azure Web App resource

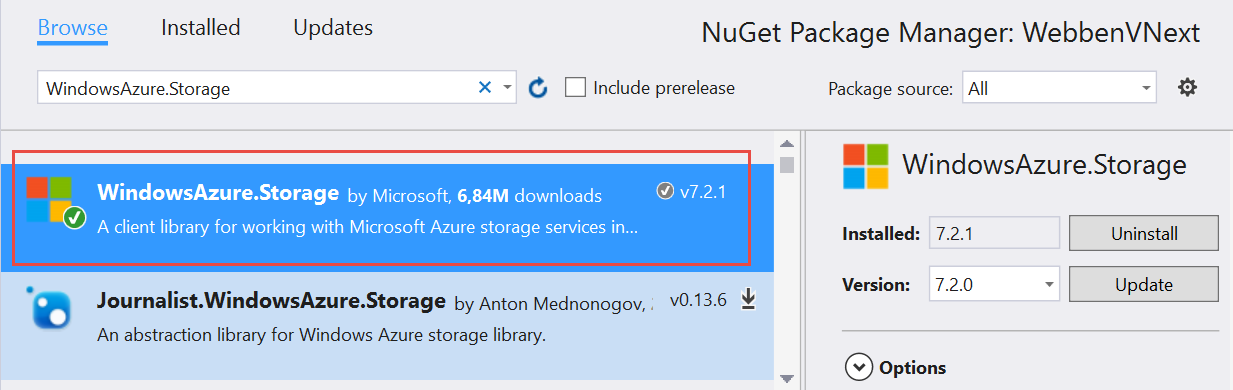
1. Select the new resource group from the list
2. Click on the +Add button in the toolbar  
     
   
3. Type **web app** in the filter to search for the Azure Web App resource  
     
   
4. Select the Web App resource
5. Give the web app a (globally) unique name, and select the existing resource group  
     
   
6. As part of creating the new web app, you need to create a new App Service Plan. Give the app service plan the same name as the web app, and select the **Free** pricing tier:  
     
   
7. Click **Create** to create the new web application

### Create a Storage Account

1. On the resource groupo, select the +Add button again and create a Storage Account. Search for “storage account” and select it:  
     
     
     
   
2. Give the storage account a unique name (**Note**: it can only contain letters and digits), and leave the rest of the properties with their default settings.  
     
   
3. Click **Create** to create the storage account
4. After the account is created, select it from the resource group and click on **Access Keys** in the **Settings** section
5. Make a note of the storage account name and the first access key (key1). You will need this later in this lab  
     
   

## Store Images in Azure Storage

In this first part of the lab we will add code to the web app that will allow us to read and store our images in Azure Blob Storage instead.

1. Install the WindowsAzure.Storage NuGet package:  
     
   
2. Add a new class in the **Storage** folder called **AzureBlobsOptions**:

|  |
| --- |
| public class AzureBlobsOptions  {  public string StorageConnectionString { get; set; }  } |

1. Add a new class in the **Storage** folder called **AzureBlobs** that implements the **IBlobs** interface.

|  |
| --- |
| using System.Collections.Generic;  using System.IO;  using System.Linq;  using System.Threading.Tasks;  using Microsoft.Extensions.Options;  using Microsoft.WindowsAzure.Storage;  using Microsoft.WindowsAzure.Storage.Blob;  namespace WebbenVNext.Storage  {  public class AzureBlobs : IBlobs  {  private readonly CloudBlobContainer \_container;  public AzureBlobs(IOptions<AzureBlobsOptions> options)  {  var storageAccount = CloudStorageAccount.Parse(options.Value.StorageConnectionString);  var blobClient = storageAccount.CreateCloudBlobClient();  \_container = blobClient.GetContainerReference("uploads");  }  public async Task Save(string name, Stream file)  {  await \_container.CreateIfNotExistsAsync(BlobContainerPublicAccessType.Blob, null, null);  var blob = \_container.GetBlockBlobReference(name);  await blob.UploadFromStreamAsync(file);  }  public async Task<IEnumerable<string>> GetAllBlobUrls()  {  await \_container.CreateIfNotExistsAsync(BlobContainerPublicAccessType.Blob, null, null);  var blobs = await \_container.ListBlobsSegmentedAsync(null);  return blobs.Results.Select(x => x.Uri.AbsoluteUri);  }  }  } |

### Prepare App for Deployment

To be able to deploy the application to multiple environment we need to manage the configuration for it. To do this, we will add a tokenized appsettings file to the project that is when deployment the app to test and production environments.

1. Add a new JSON file to the project, call it ***appsettings,json***
2. Change the file to look like this:

|  |
| --- |
| {  "AzureBlobConnectionString": "\_\_AZURECONNECTIONSTRING\_\_"  } |

1. Add the following NuGet package to your projects:  
   1. *Microsoft.Extensions.Configuration.UserSecrets*
2. We also need to install the tooling for user secrets which is also a NuGet package, but it’s a special
3. Right-click on the web project and select *Edit WebbenVNext.csproj*
4. In the project file, add the following at the bottom, right before the </Project> element

|  |
| --- |
| <ItemGroup>  <DotNetCliToolReference Include="Microsoft.Extensions.SecretManager.Tools" Version="1.0.1" />  </ItemGroup> |

1. Save and compile the solution
2. Right-click the web project and select *Manage User Secrets*
3. This will create and open a file called *secrets.json*. Add the following line within the brackets:  
     
   userSecretsId": "aspnet-webbenvnext"
4. In the file Startup.cs, add a new property:  
     
   public IConfigurationRoot Configuration { get; }
5. Add the following using statement at the top of the file to bring in the IConfigurationRoot definition  
     
   using Microsoft.Extensions.Configuration;
6. Add a constructor to the Startup

|  |
| --- |
| public Startup(IHostingEnvironment env)  {  var builder = new ConfigurationBuilder()  .SetBasePath(env.ContentRootPath)  .AddEnvironmentVariables();  if (env.IsDevelopment())  {  builder.AddUserSecrets<Startup>();  }  else  {  builder.AddJsonFile($"appsettings.json", optional: true);  }  Configuration = builder.Build();  } |

1. In the method **ConfigureServices**, change so that you are now using the AzureBlobs implementation instead of the LocalBlobs:  
     
    services.AddTransient<IBlobs, AzureBlobs>();
2. Add the following lines at the end of the **ConfigureServices** method:

|  |
| --- |
| services.Configure<AzureBlobsOptions>(options =>  {  options.StorageConnectionString = Configuration.GetValue<string>("AzureBlobConnectionString");  }); |

1. Now you need to add a **user secret** for the Azure storage connection string.   
     
   Open a developer command prompt for Visual Studio 2017 and browse to the directory of the web application.  
     
   Run the following command:

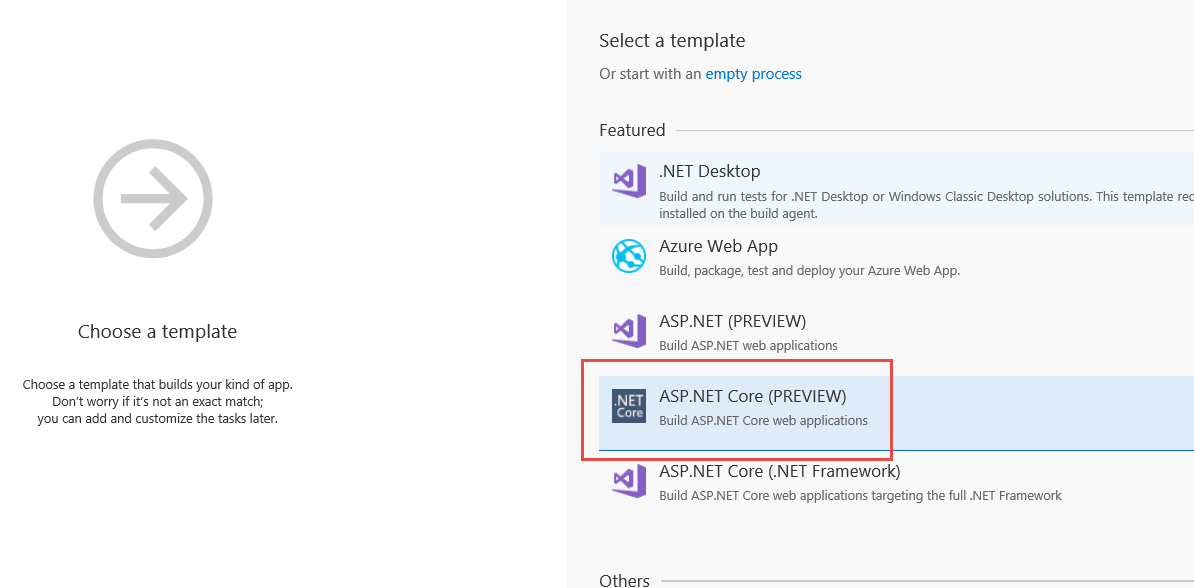
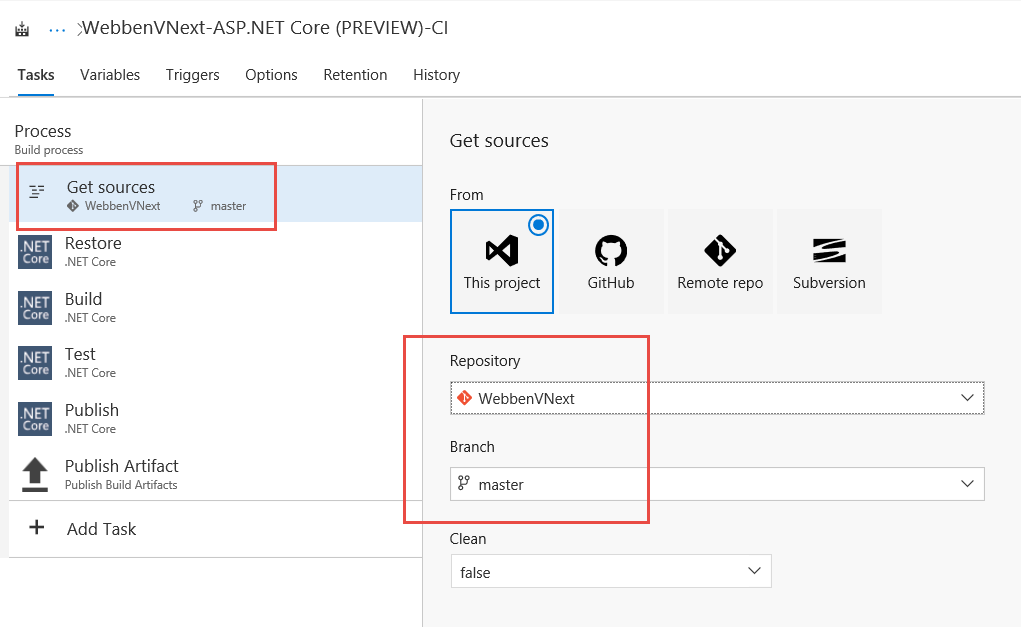
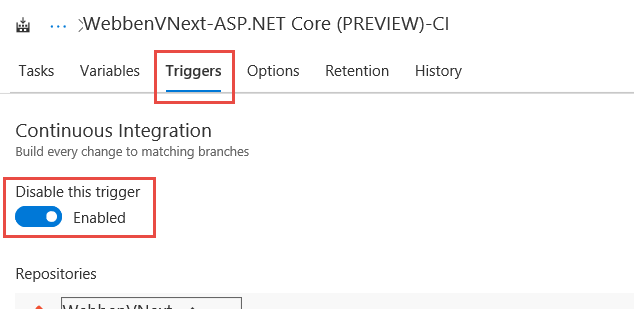
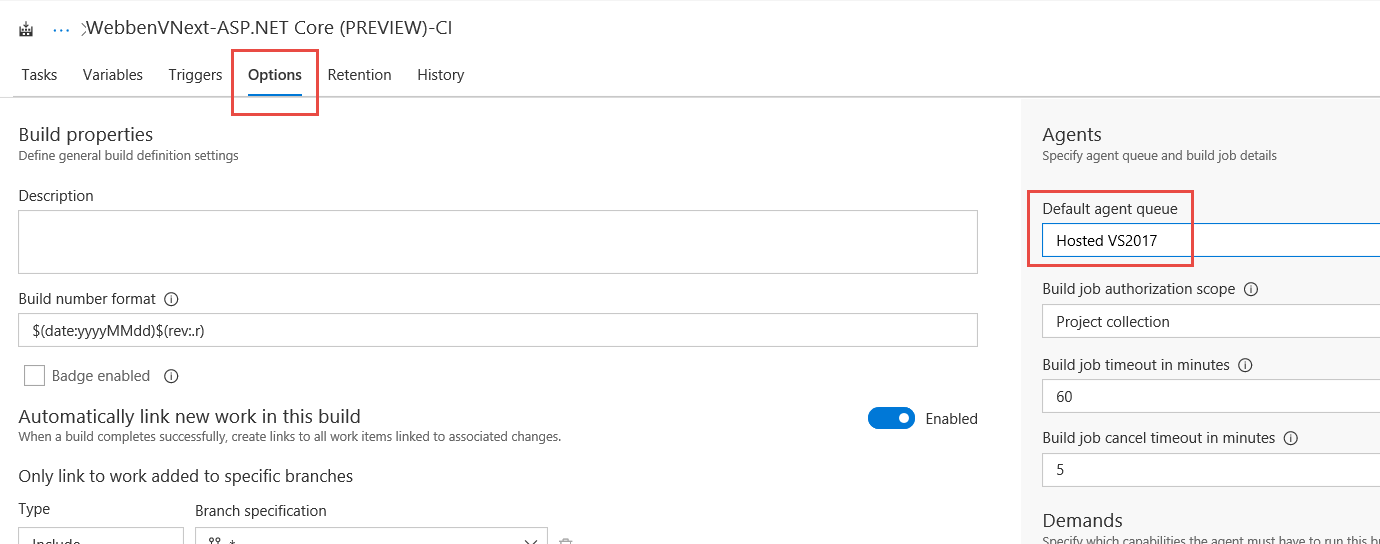
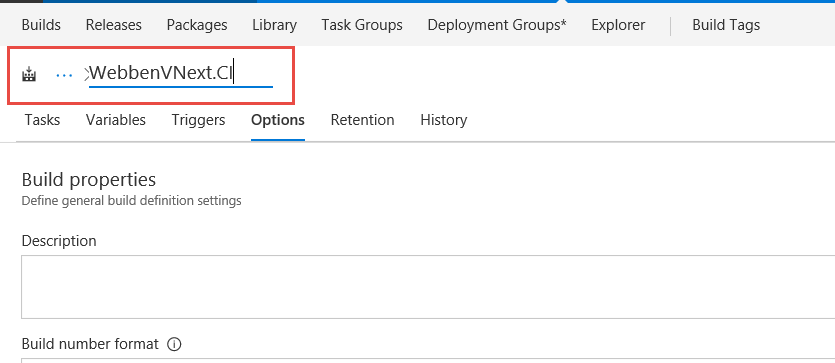
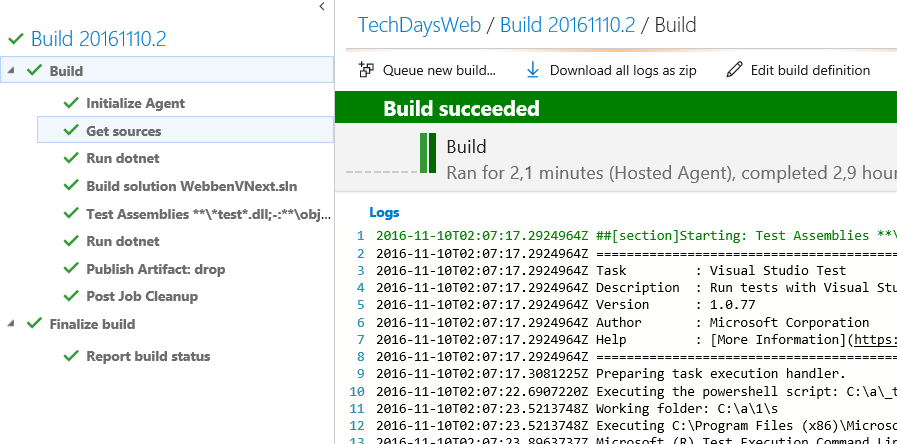
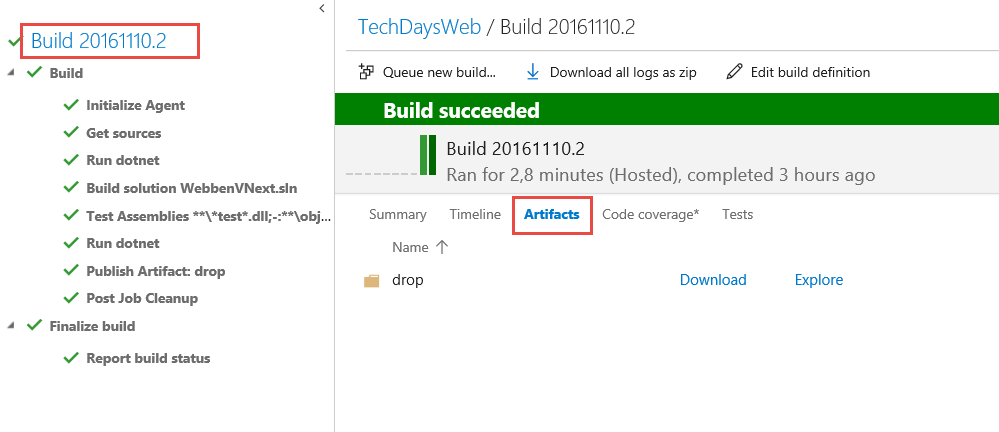
|  |
| --- |
| dotnet user-secrets set "AzureBlobConnectionString" "DefaultEndpointsProtocol=https;AccountName=STORAGEACCOUNT;AccountKey=ACCOUNTKEY" |

Where *STORAGEACCOUNT* and *ACCOUNTKEY* is replaced with the values that you noted before when creating the storage account.  
  
For example:  
  
***dotnet user-secrets set "AzureBlobConnectionString" "DefaultEndpointsProtocol=https;AccountName=devsumweb123;AccountKey= sXuZnpWf8Eh/Iu5RV4FO1nPS0cMPEJv99wJKBbsBcAsA90tMfjG0khDve9oax+YSgnZQzyfxSTawwJ7o7CctIw=="***

1. Compile and run the application locally, and make sure that it still works
2. Commit and push the changes.

## Create a Build Definition

In this step you will create a build definition in VSTS, that will build and test your web application.

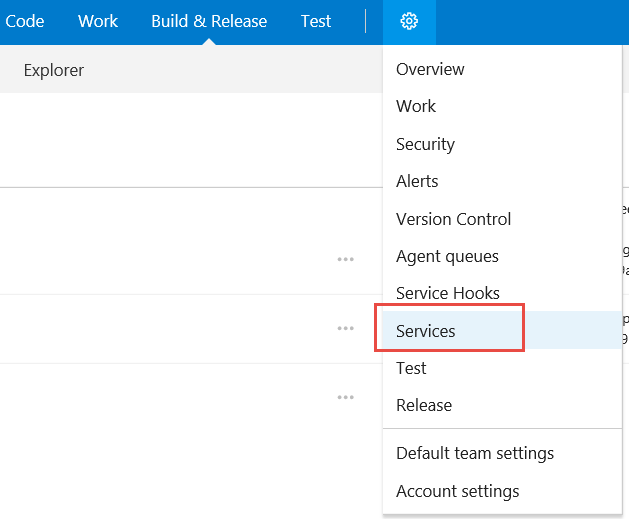
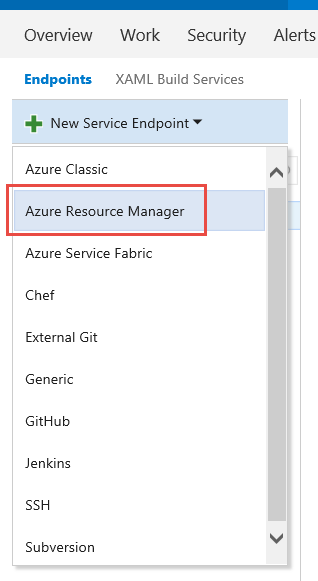
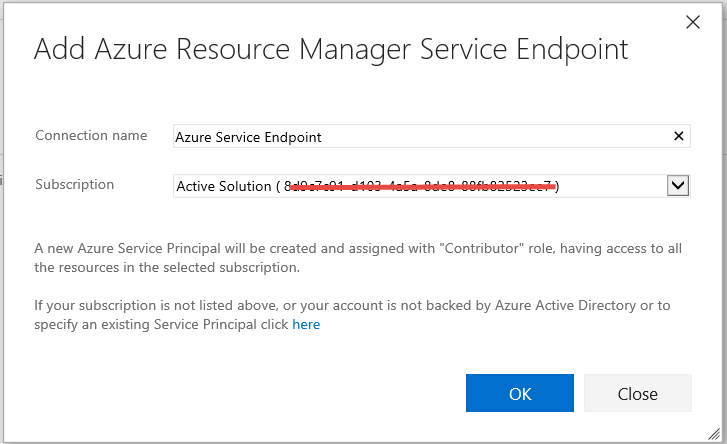
1. In Visual Studio Team Services, select your team project and go to the Build hub
2. Select +New to create a new build definition. Select the **ASP.NET Core (PREVIEW)** template  
     
   
3. Press Apply.
4. Select the Get Sources task
5. Select the team project and Git repository that you pushed your code to  
     
   
6. Go to the Triggers tab and enabled the Continuous Integration trigger  
     
   
7. On the **Options** tab, select the **Hosted VS2017** agent queue  
     
   
8. Change the name of the build definition to (for example) **WebbenVNext.CI**  
     
   
9. Select Save & Queue to start a new build and make sure that it completes successfully  
     
   
10. Open the artifacts of the build and verify that the published web application is located there.  
      
    

## Create a Release Pipeline

In this step you will create a release definition that takes the output from the build (e.g. “artifacts”) and deploys the application to Microsoft Azure.

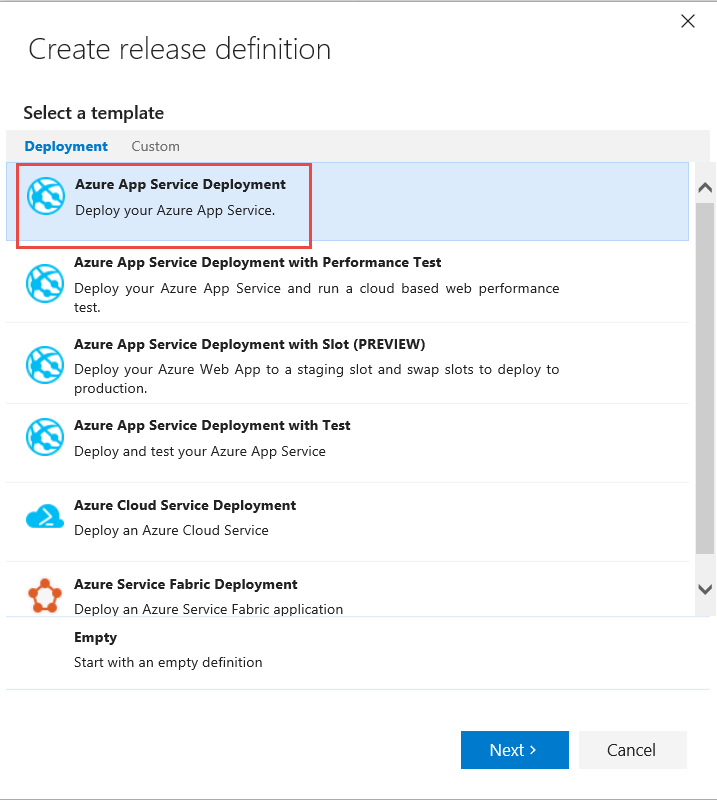
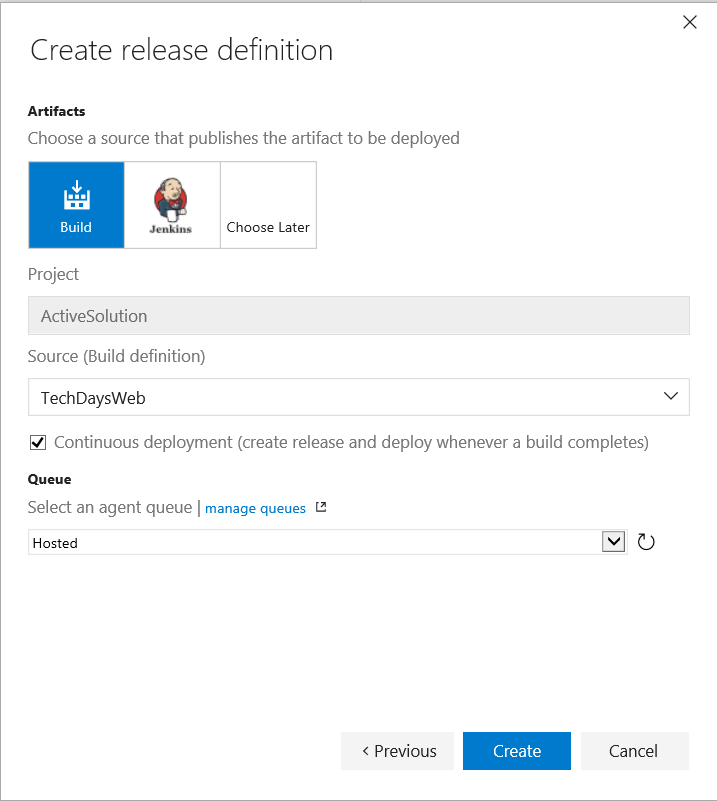
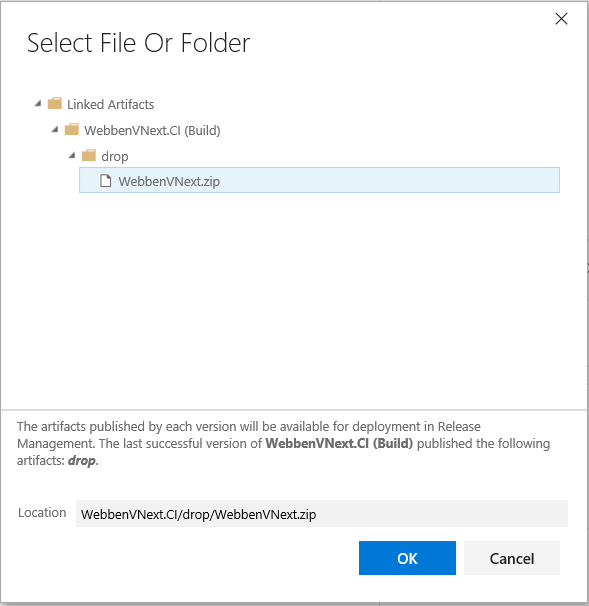
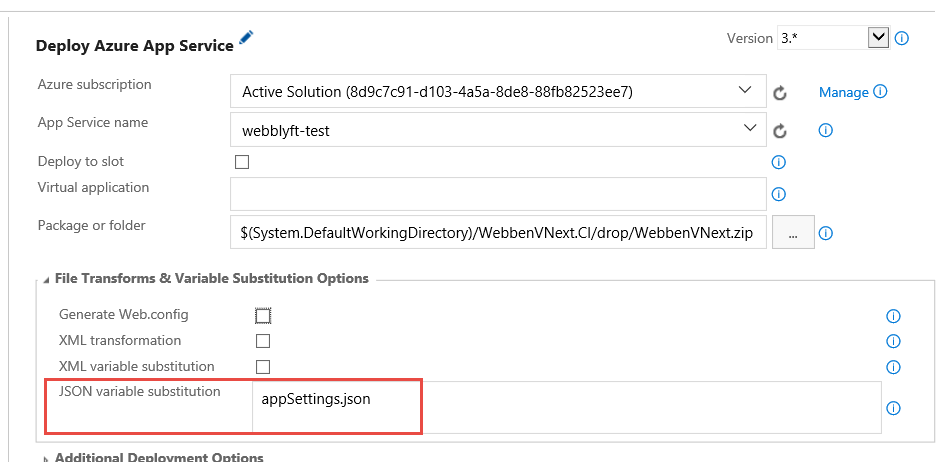
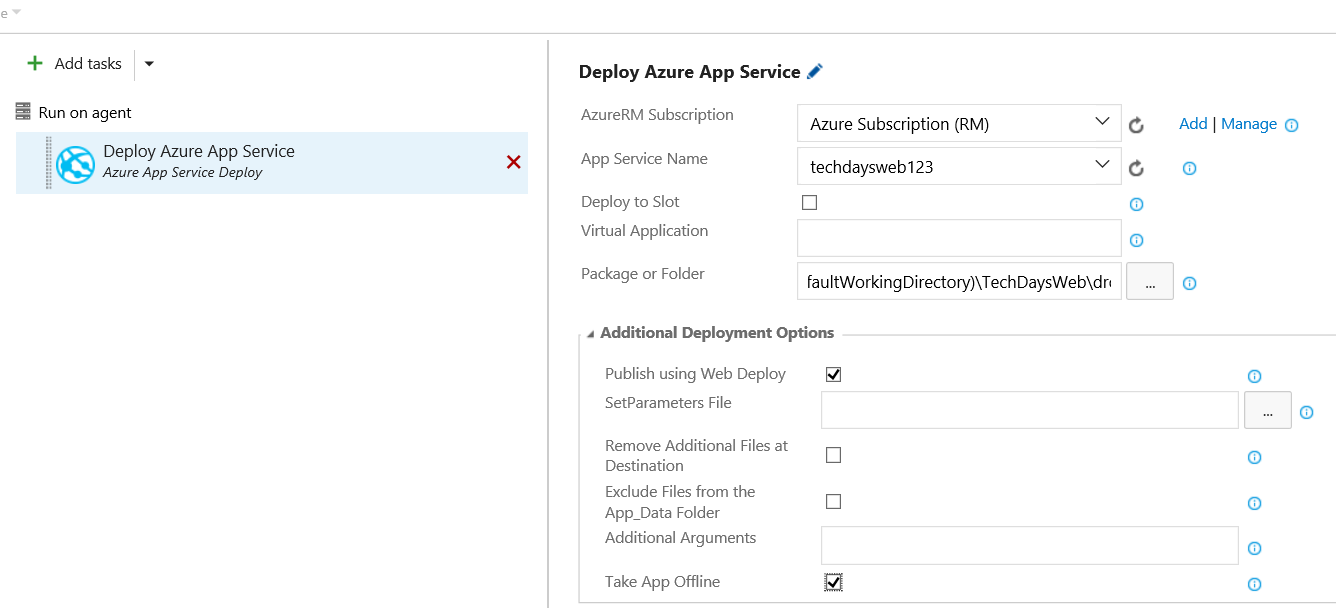
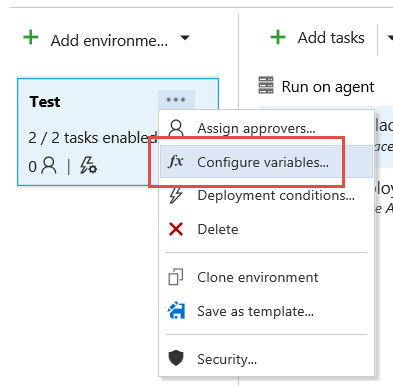
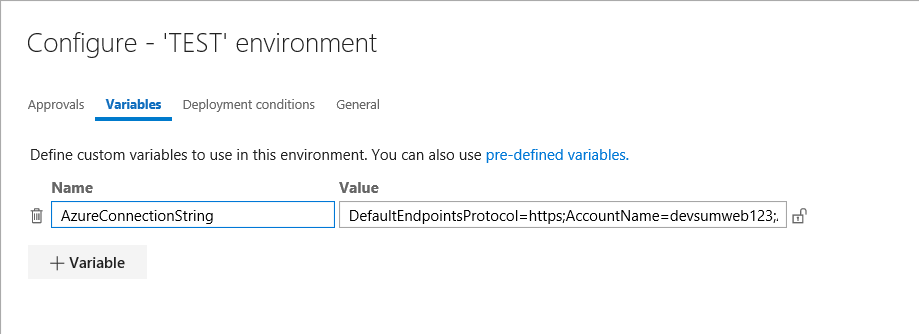
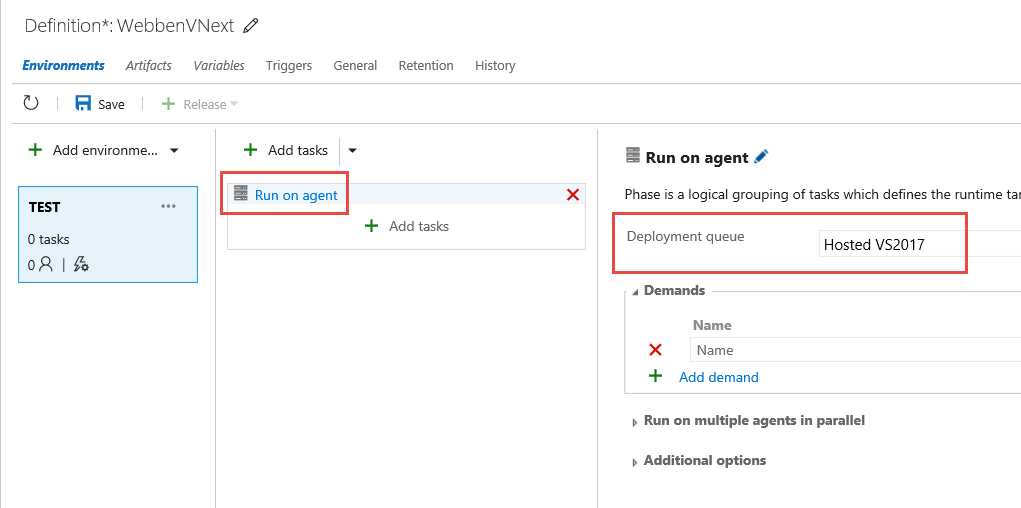
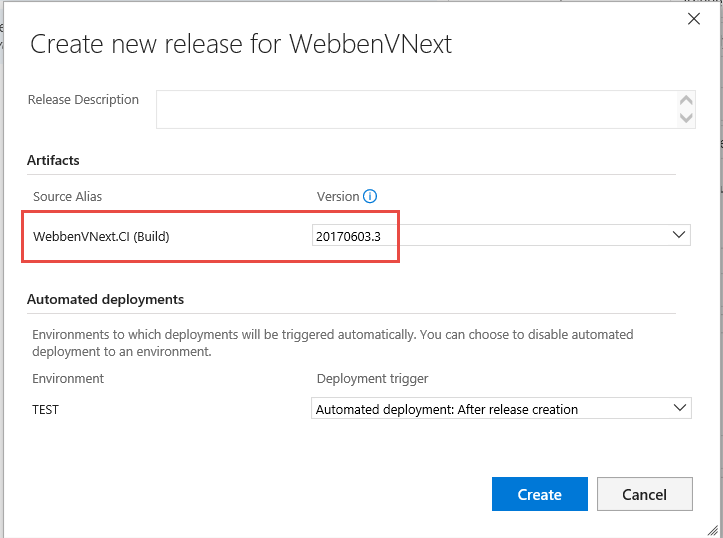
### Create an Azure Service Endpoint

When a build or a release needs to communicate with an external service (such as Azure), the information needed for this is defined in a *Service Endpoint*. This can often be created by an administrator that has the necessary permissions and credentials, then the service endpoints can be used when creating the build and release definitions.

1. Click on the Settings gear icon, and select **Services**
2. Select **New Service Endpoint** and select **Azure Resource Manager** from the list  
     
   
3. Enter a name for the endpoint (for example **Azure Service Endpoint**) and then select your Azure subscription from the dropdown list  
     
   
4. Press OK to create the service endpoint

### Create a Release Definition

Now it is time to create the release definition

1. Go to the Release hub and create a new release definition
2. Select the **Azure App Service Deployment** template  
     
   
3. Select the Team Project and build definition that you just created.  
   Check the Continuous Deployment option  
     
   
4. Select **Create** to create the release definition
5. Rename the default environment to **Test**
6. Select the **Deploy Azure App Service** task and configure it:  
   1. Select the Azure Service Endpoint that you created before
   2. Select the Azure Web App that you just created in the drop down list
   3. In the **Package or Folder**, browse to the generated zip file in the build output:  
        
      
   4. In the File transforms.. section, enter appSettings.json in the JSON variable substitution field:  
        
      
   5. Check the **Take App Offline** option  
        
      
7. Click on the context menu for the **Test** environment and select **Configure Variables**  
     
   
8. Remove all existing variables from the environment
9. Add a new variable called **AzureConnectionString** and give it the value  
     
   *DefaultEndpointsProtocol=https;AccountName=****STORAGEACCOUNT****;AccountKey=****ACCOUNTKEY*** Where *STORAGEACCOUNT* and *ACCOUNTKEY* is replaced with the values that you noted before when creating the storage account  
     
   
10. Select the **Run on agent** link in the TEST environment, and select Hosted VS2017 as the deployment queue  
      
    
11. Rename therelease definition to WebbenVNext
12. **S**ave the release definition
13. Start a new release and select the latest build  
      
    
14. The release should finish successfully. After that, browse to   
    https://<WEBAPPNAME>.azurewebsites.net to verify that the site is working properly

# Next Step

In this next step you will learn how to enhances the web application client using TypeScript. This will include using tools such as NPM and Gulp to install and run steps as part of a front-end build pipeline.

## Challenges

In case you have time left, here are some challenges that you can try to implement.

* Define a production environment for your application. E.g. create a separate web app for your production site and extend your release definition to deploy to production after the test app has been deployed.  
    
  Note that you should have a different storage account as well for production, so you need to apply different connection strings for this environment.
* Use **Azure Web App Deployment Slots** to implement Blue/Green deployment. Note that this requires using a Standard Pricing Plan which will incur some costs