DevOps Kata

**Release management**

Last updated: 11/27/2016



1. **TABLE OF CONTENT**

[Overview 3](#_Toc468083202)

[Prerequisites 3](#_Toc468083203)

[Exercises 3](#_Toc468083204)

[Exercise 1: Pre-reqs 4](#_Toc468083205)

[Task 1: VSTS 4](#_Toc468083206)

[Task 2: Create 2 Azure Web Apps 5](#_Toc468083207)

[Exercise 2: code and commit 7](#_Toc468083208)

[Exercise 3: Build 10](#_Toc468083209)

[Exercise 4: Release 13](#_Toc468083210)

## Overview

Whether your app is on-premises or in the cloud, you can automate build-deploy-test workflows and choose the technologies and frameworks, then test your changes continuously in a fast, scalable, and efficient manner. Continuous testing with Visual Studio Team Services or Team Foundation Server ensures your app still works after every check-in and build, enabling you to find problems earlier by running tests automatically with each build. Choose the test technologies and frameworks you prefer to use. When your build is done, review your test results to start resolving the problems you find. Rich and actionable build-on-build reports let you instantly see if your builds are getting healthier. But it's not just about speed - detailed and customizable test results measure the quality of your app.

To determine what proportion of your project’s code is actually being tested by coded tests such as unit tests, you can use the code coverage feature of Visual Studio. To guard effectively against bugs, your tests should exercise or ‘cover’ a large proportion of your code. Code coverage analysis can be applied to both managed (CLI) and unmanaged (native) code. Code coverage is an option when you run test methods using Test Explorer. The results table shows the percentage of the code that was run in each assembly, class, and method. In addition, the source editor shows you which code has been tested.

### Prerequisites

1. In order to complete the lab
2. 1. Log on to your Visual Studio / MSDN subscription and create or use your own VSTS instance. Alternatively, you can use a team sandbox VSTS instance if you have one.
3. 2. Have an azure subscription available that you can deploy resource to.
4. 3. Visual Studio Enterprise

### Exercises

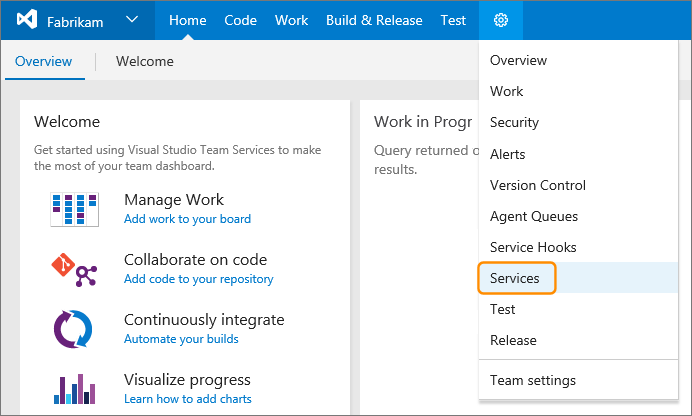
* 1. This hands-on lab includes the following exercises:
  2. Create a new web application
  3. Add unit tests and code coverage
  4. Add to VSTS build service and monitor results
  5. Estimated time to complete this exersize: **15 minutes**.

Exercise 1: Pre-reqs

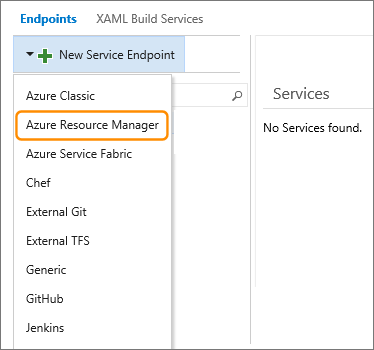
#### Task 1: VSTS

You need an Azure subscription to perform these steps.

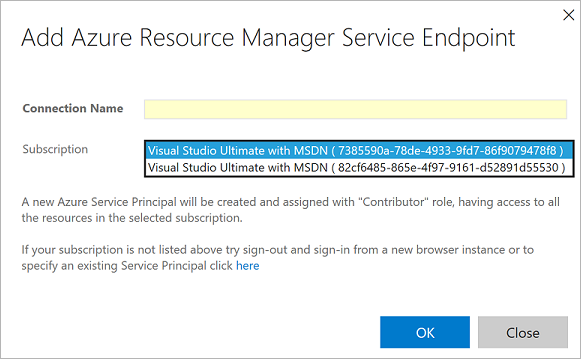
1. Open your TFS team project in your web browser. Choose the settings icon **Settings** icon in the menu bar and select **Services**.



1. In the **Services** tab, choose **New Service Endpoint** and select **Azure Resource Manager**.

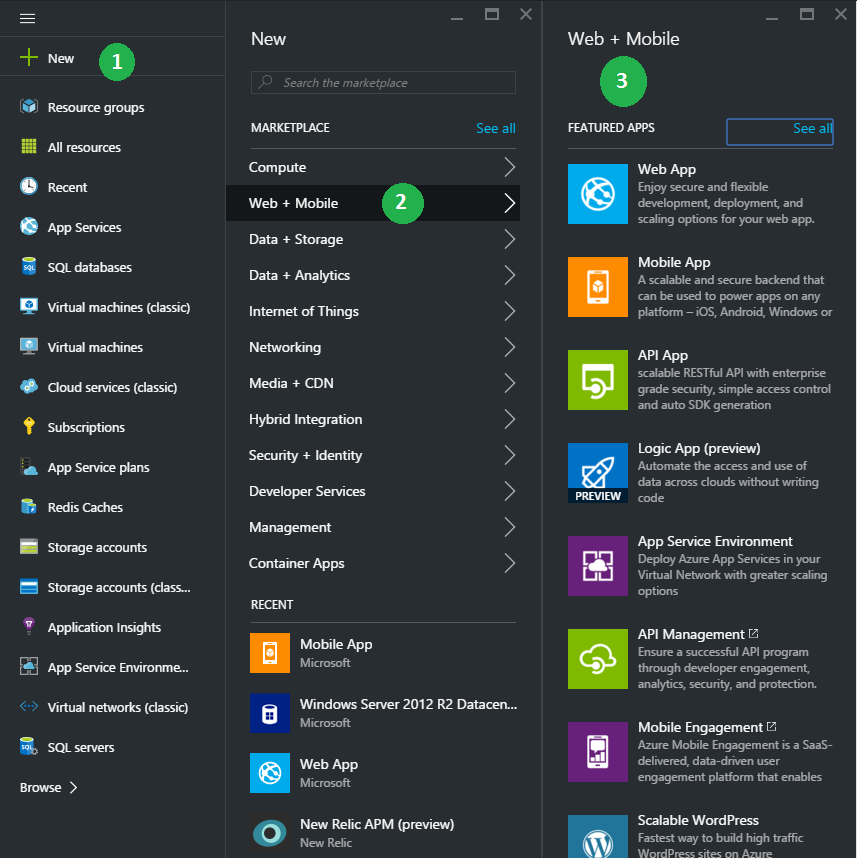


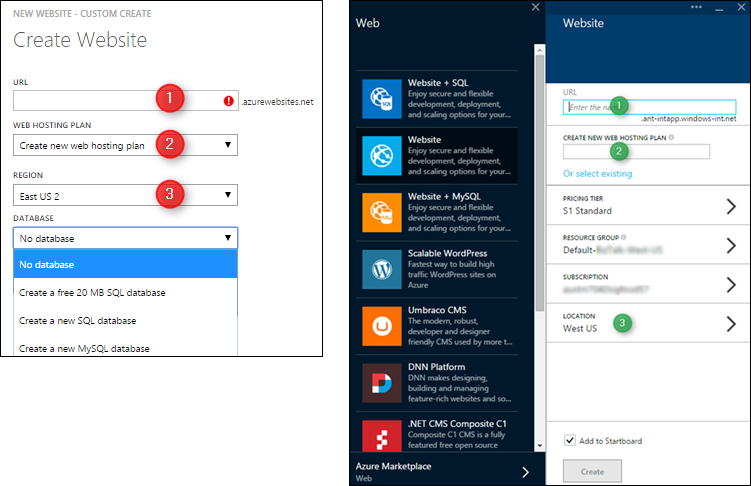
1. Enter a user-friendly name for the connection such as **Fabrikam** and select your Azure subscription.



1. Choose **OK** to save the Azure service endpoint.

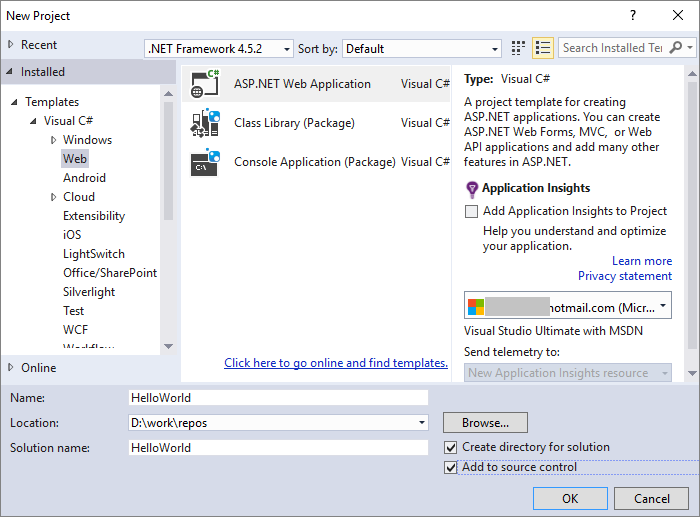
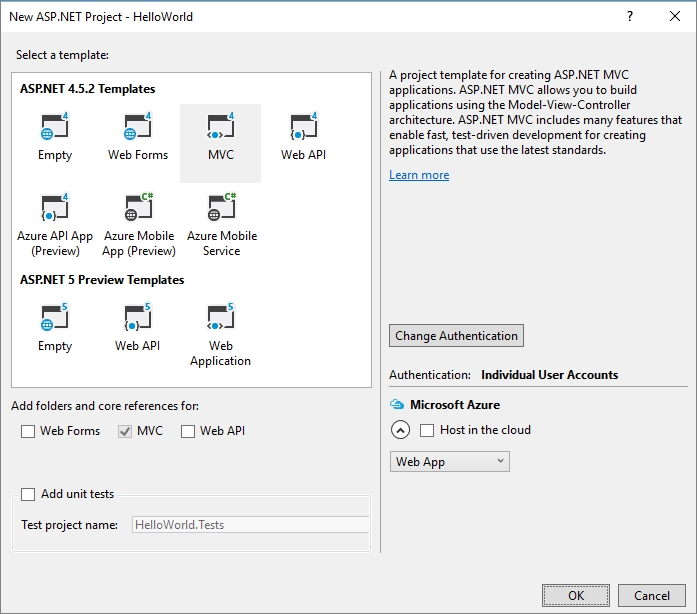
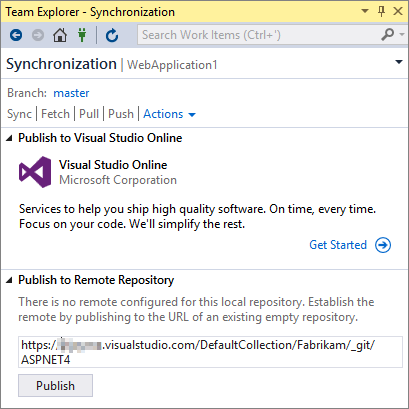
#### Task 2: Create 2 Azure Web Apps

1. 1. Creating new web apps is still as easy as 1-2-3. Navigate to the portal and click the following:  
   

1. 2. Specify URL, App Service plan, and location in the portal just as you do in the classic portal.
2. 

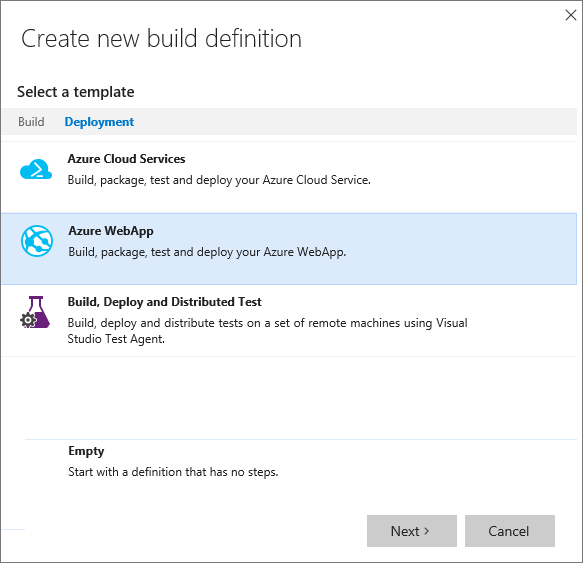
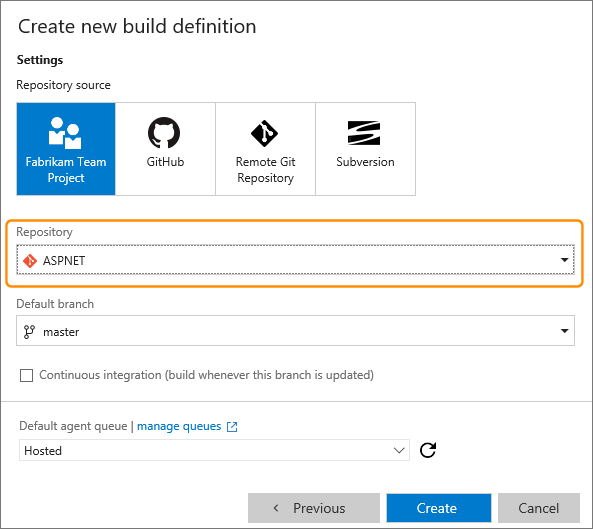
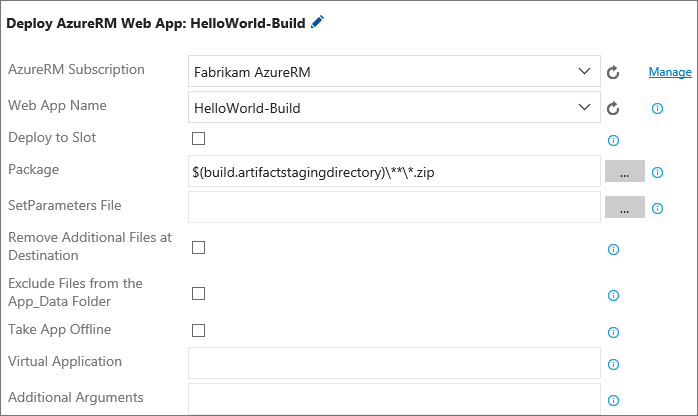
Exercise 2: code and commit

Carry out the following steps to create a simple ASP.NET MVC app to deploy.

1. Create a new **ASP.NET Web Application** project in Visual Studio.
2. Choose the **MVC** project type and leave the other settings at their defaults.
3. Commit the project into a Team Foundation Server or Visual Studio Team Services TFVC or GIT repository.

Exercise 3: Build

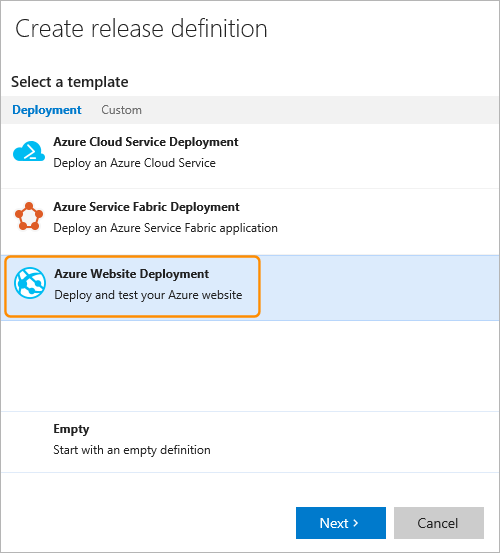
Carry out the following steps to build and deploy the ASP.NET MVC app by using a build definition.

1. Open the **Builds** tab of the **Build & Release** hub and choose **+ New** to create a new build definition.
2. In the **Deployment** tab of the **Create new build definition** dialog, select **Azure WebApp** and choose **Next**.
3. Select the repository where you committed your app and choose **Create**.
4. Add an **Azure RM Web App Deployment** task step to the new build definition and configure it as follows:
   * Select your Azure subscription
   * Type a name for the website
5. Save the build definition.
6. Queue a new build and verify that the artifact containing the zip file is published and that the website is correctly deployed.

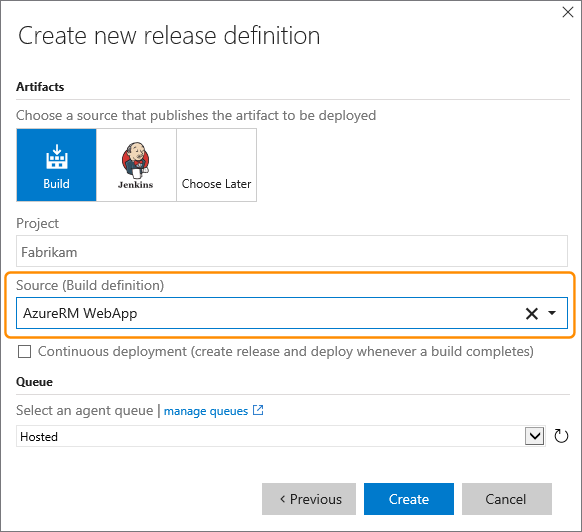
Exercise 4: Release

Carry out the following steps to deploy the ASP.NET MVC app by using a release definition.

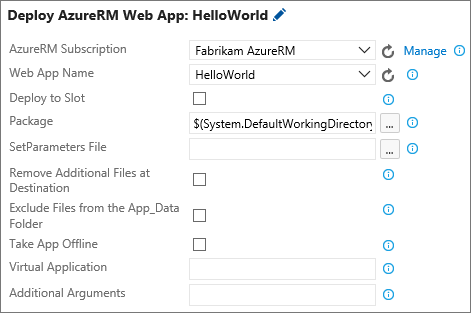
1. Open the **Releases** tab of the **Build & Release** hub and choose the plusicon to create a new release definition.
2. In the **Create release definition** dialog, select the **Azure Website Deployment** template and choose **Next**.



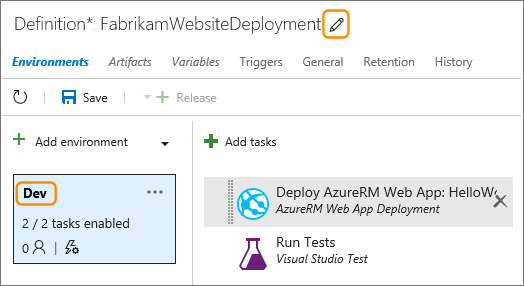
1. In the next page, select the build definition you created earlier and choose **Create**. This creates a new release definition with one default environment that contains the appropriate tasks.



1. Configure the **Azure Web App Deployment** task as follows:
   * Select the name of the Azure service endpoint you defined earlier
   * Type a name for the website instance



1. Optionally edit the name of the new release definition and change the name of the environment from **Default Environment** to **Dev**.



1. Save the new release definition.
2. Create a new release and select the latest build to deploy it to the single environment in the definition.
3. Optionally, create additional environments in the same release definition, specify different Azure websites in each environment, and try deploying to multiple environments.