Work with Docker Tools for Visual Studio

# Overview

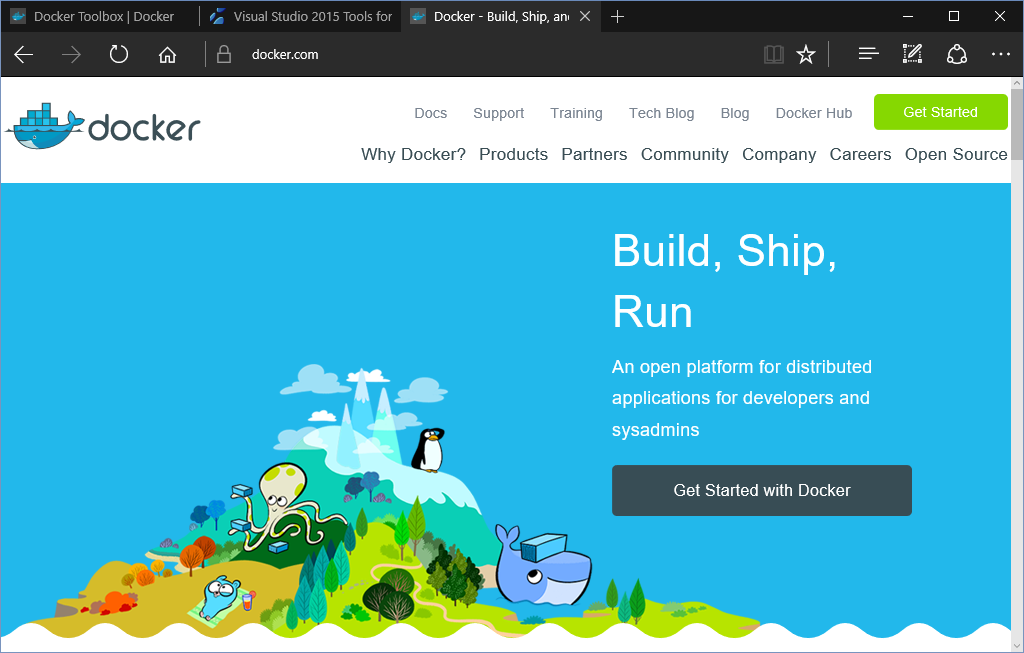
Docker is an open-source project that automates the deployment of applications inside software containers by providing an additional layer of abstraction. It also provides automation for operating-system-level virtualization.

Docker containers provide many benefits for the different app lifecycles:

* During development:
  + Instant provisioning of the required hardware resource as a container.
  + Instant provisioning of the required software stack configuration of the container.
  + Option to de-provision the container if it isn’t used. (This option needs to be coded.)
  + Integrated DevOps.  Development and operations are no longer disconnected.
* During testing:
  + No more huge, complex scripts are required to deploy apps and make them work.
  + Developers can relax – no more excuses like “It used to work for me.”
  + Everything is automated.  No time is wasted.  Continuous delivery and product releases.
* During deployment:
  + Every app gets a container for targeted error detection.
  + Instant rollback of faulty builds
  + Rolling upgrades with zero downtime
  + Enabled for microservice apps
  + Instant horizontal scalability

This tutorial teaches you how to work with Docker. It explains what types of containers are formed, and how to host a web app in Docker and deploy it in Azure. It also teaches you how modify an application that’s run locally in Docker, and how to manage load-balanced multi-containers by using Docker-Compose. In addition, it teaches you how to create Docker containers by using Docker Swarm.

After you’ve configured Docker, you’ll learn how to implement continuous integration and continuous deployment (CI/CD).



You will also learn:

* How to prepare your machine to host Docker applications by using [Visual Studio 2015 Tools for Docker](https://visualstudiogallery.msdn.microsoft.com/0f5b2caa-ea00-41c8-b8a2-058c7da0b3e4).
* About the Docker application life cycle and the types of containers that exist.
* How to host a web app in Docker and then deploy it in Azure, as well as how to modify an application that’s running locally in Docker.
* How to use Docker Compose to load balance multiple Docker containers.
* How to create Docker containers by using Docker Swarm.
* How to implement CI/CD with Docker.

# Set up the development environment

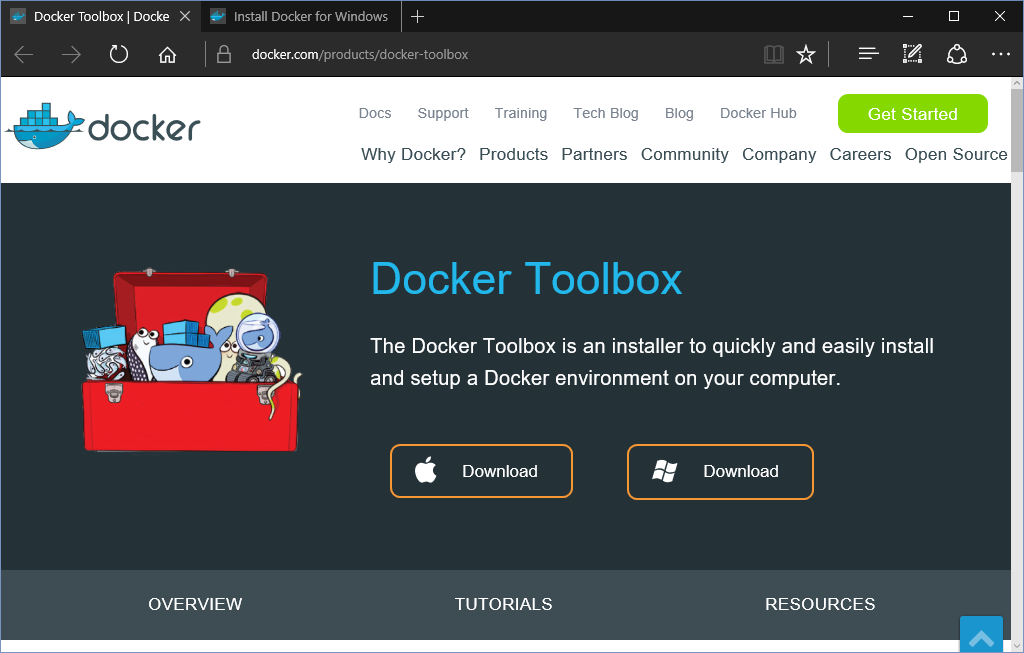
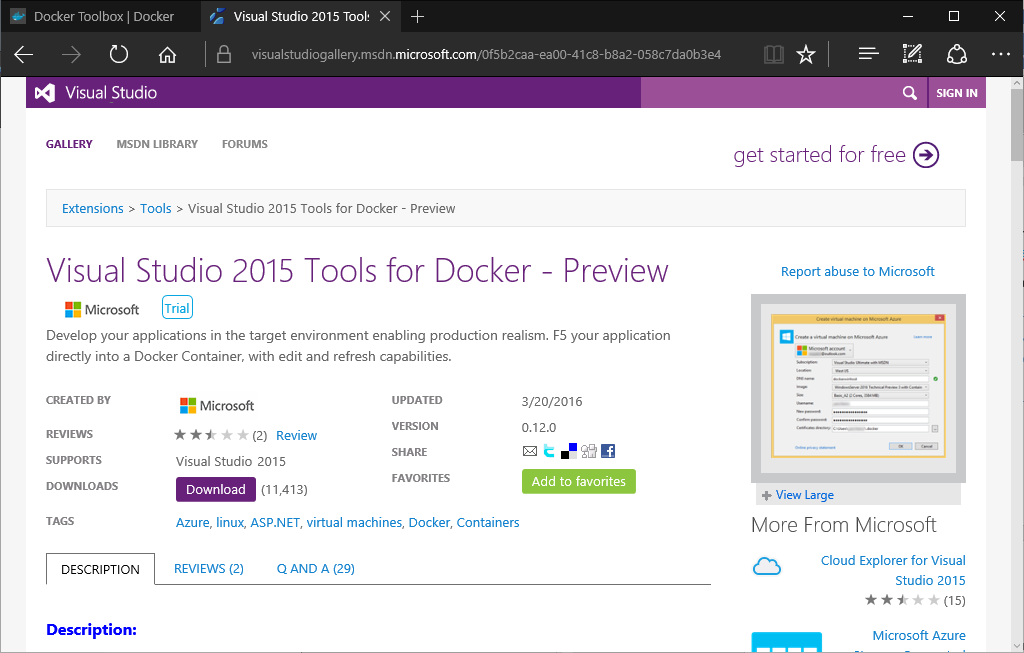
To start, set up your development environment by installing the latest version of the [Azure SDK](https://www.microsoft.com/web/handlers/webpi.ashx/getinstaller/VWDOrVs2015AzurePack.appids).

[Visual Studio 2015](https://go.microsoft.com/fwlink/?linkid=746481&clcid=0x409)▶

[Visual Studio 2013](https://go.microsoft.com/fwlink/?linkid=746482&clcid=0x409)▶

If you don't have Visual Studio installed, use the link for Visual Studio 2015. Visual Studio will be installed along with the SDK.

You also need to install [Docker Toolbox](https://www.docker.com/products/docker-toolbox) for your operating system, as well as the [Visual Studio 2015 Tools for Docker](https://visualstudiogallery.msdn.microsoft.com/0f5b2caa-ea00-41c8-b8a2-058c7da0b3e4) extension. This extension gives you the option to work with and deploy Docker containers from Visual Studio 2015.



# Host an app in a single Docker container

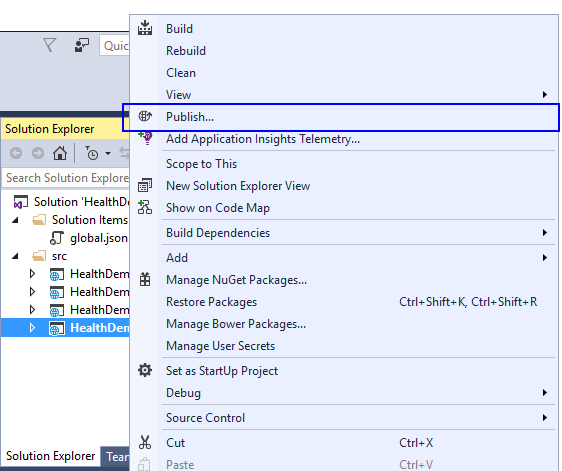
There are two different containers available:

* **Windows Server containers**: A Windows Server container host that’s running Windows Server 2016 (full or core), either on-premises or in Azure. Provides an isolated, portable, and resource-controlled operating environment in which to run applications and host processes. It also provides isolation between the container and host, through process and namespace isolation.
* **Hyper-V container**: A Windows container host that’s enabled with nested virtualization. It’s not available in Azure, which means that an on-premises container host is needed. It provides an additional layer of isolation for Windows Server containers. Each Hyper-V container is created within a highly optimized virtual machine. While a Windows Server container shares a kernel with the container host, a Hyper-V container is completely isolated. Hyper-V containers are created and managed identically to Windows Server containers. For more information about Hyper-V containers see [Managing Hyper-V containers](https://msdn.microsoft.com/en-us/virtualization/windowscontainers/management/hyperv_container).

In this tutorial, you’ll learn how to host a web app on a Windows Server container.

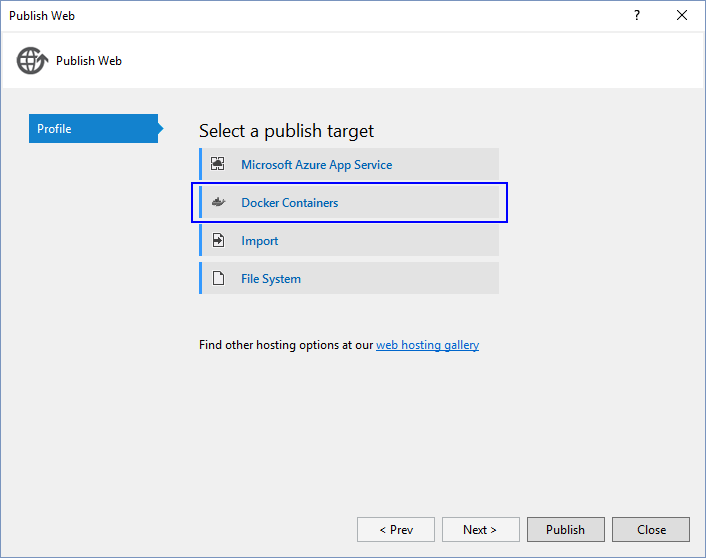
After you have a project that’s ready to publish in a Docker container in Azure, open it with Visual Studio 2015. In this tutorial, you’ll open 01\_Demos\_ASPNET5. This tutorial is available in the [Demos repository](https://github.com/Microsoft/HealthClinic.biz) that was presented at [Microsoft Connect(); //2015](https://channel9.msdn.com/Events/Visual-Studio/Connect-event-2015/).

1. After you’ve opened the demo, right-click on Web project (**MyHeath.Web**). Then, on the contextual menu, select **Publish**….



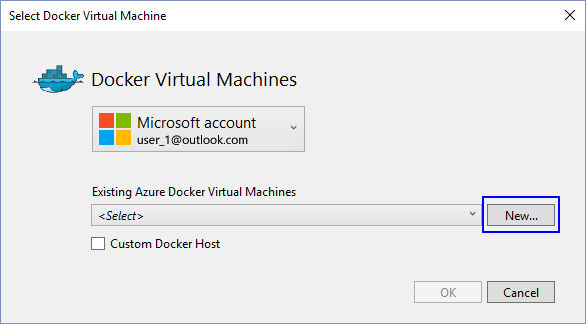
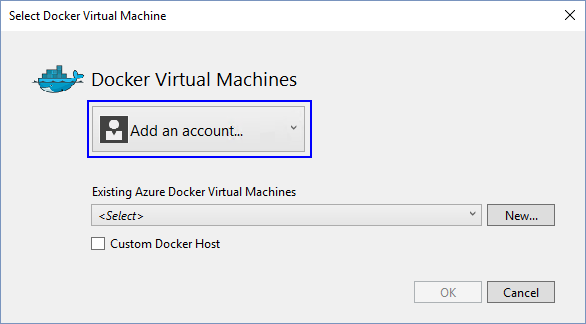
1. In the **Select a publish target** section of the **Publish Web** screen, select **Docker Containers**.

If the option isn’t available, make sure you’ve installed [Visual Studio 2015 Tools for Docker](https://visualstudiogallery.msdn.microsoft.com/0f5b2caa-ea00-41c8-b8a2-058c7da0b3e4).



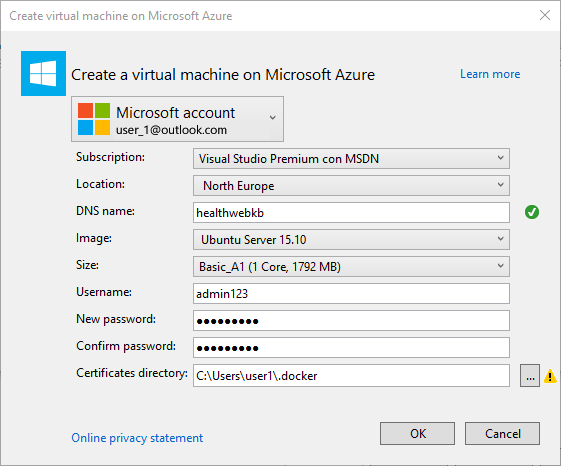
1. The Select Docker Virtual Machine dialog box enables you to specify which Docker host you want to use to publish your project. You can create a new Docker host, or use an existing one that’s hosted in Azure or another location.

To create a new Docker host in Azure, sign in to your Azure subscription. Then select **New…**.



Alternatively, you publish in a custom Docker host. For more information, see the document about [how to publish a custom Docker host](https://azure.microsoft.com/en-us/documentation/articles/vs-azure-tools-docker-hosting-web-apps-in-docker/#provide-a-custom-docker-host).

1. Enter the required information in the **Create a virtual machine on Microsoft Azure** dialog box.



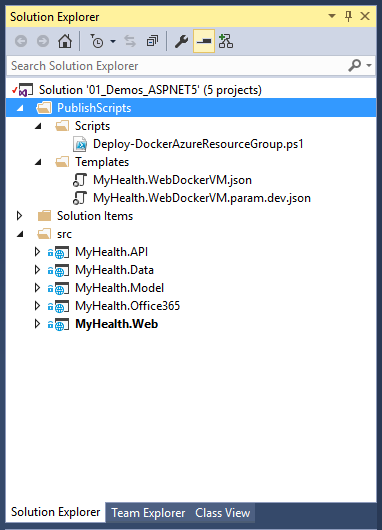
|  |  |
| --- | --- |
| **Propiedad** | **Configuración** |
| Location | Change this setting to the region closest to you. |
| DNS Name | Enter a unique name for the virtual machine. |
| Image | Choose an OS image to use in the Docker host. |
| Username | Enter a unique user name for the virtual machine. |
| Passwords | Enter a password for the user and then confirm it. |
| Certificates directory | Specify the directory where your Docker certificates are located. You can create a new directory or select an existing directory. We recommend that you use the default directory (c:\Users\[username]\.docker). The Auth options can’t be automatically retrieved if you reuse the same host on another project. |

1. Select **OK**.

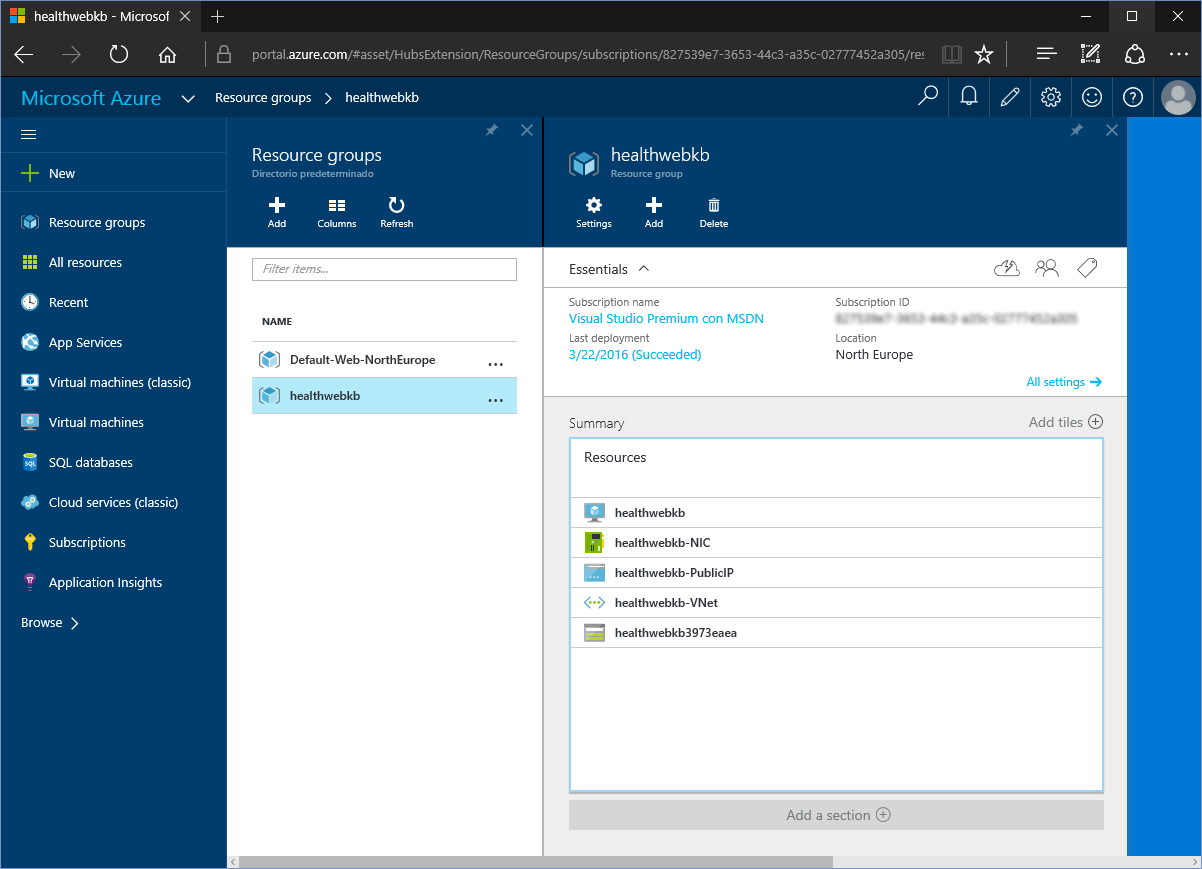
This will create a Linux virtual machine that’s configured with the Docker extension. You can also create a Windows Container host by using Windows Server 2016 Technical Preview 3 (TP3).

You will also see a message that the virtual machine was created in Azure.

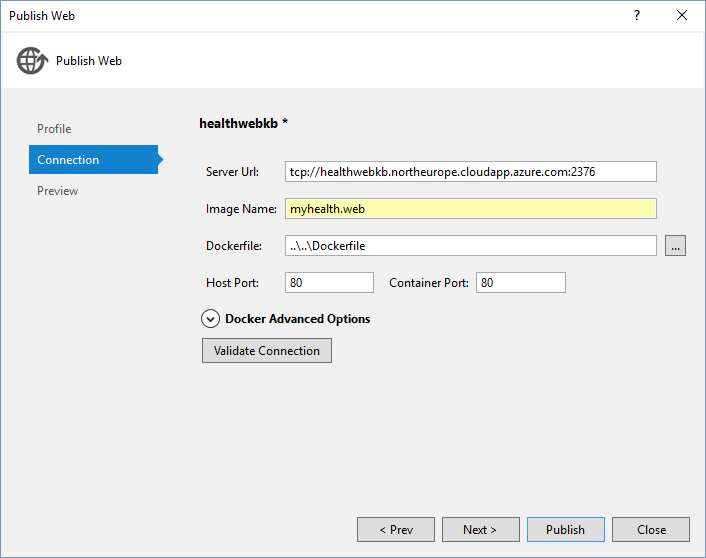
Visual Studio creates a file in the Azure Resource Manager template and in the parameter files. It also creates a PowerShell script to run the commands again in the future.



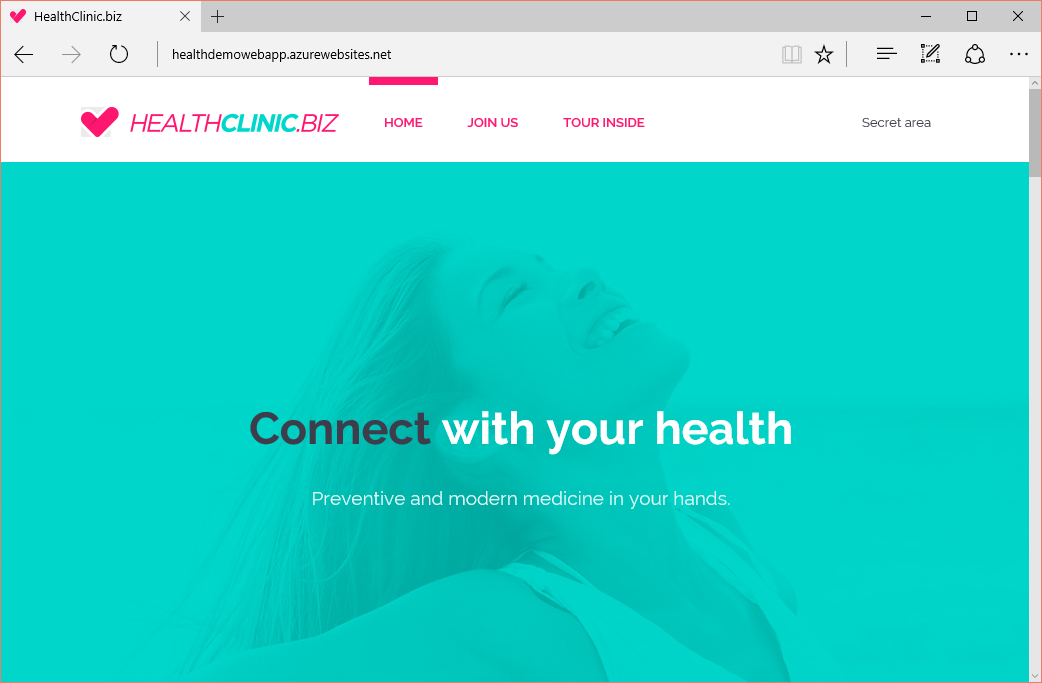
1. Visual Studio shows the progress in the **Output** window. Once the resources have been deployed, you can see them in the Azure portal.



1. Return to Visual Studio, and then select **Publish** on the web project.
2. Next, select the Docker virtual machine that you created in Azure, and then select **Continue**.



1. Select the **Publish** button, and the web application that you’re going to publish. The first time you publish an application in a Docker host, it might take a long time. Once the web app has been published, it will be shown.



# Load balance multiple Docker containers by using Docker Compose

Docker Compose is a tool that defines and runs complex applications on Linux virtual machines in Azure. With Compose, you can use a plain text file to define a complex application with Docker containers.

After a virtual machine as is configured as a Docker host, you can see how to do it by [using the extension for the virtual machine with Docker from the Azure command-line interface](https://azure.microsoft.com/es-es/documentation/articles/virtual-machines-docker-with-xplat-cli/).

After you configure the Docker Linux virtual machine, connect it to the client device with SSH. It’s possible that Compose has already installed, but if not, install Compose by running the two following commands:

$ curl -L https://github.com/docker/compose/releases/download/1.1.0/docker-compose-`uname -s`-`uname -m` > /usr/local/bin/docker-compose

$ chmod +x /usr/local/bin/docker-compose

To test the Compose installation, run the next command.

$ docker-compose --version

The result can be shown as docker-compose 1.4.1 file.

You must create a docker-compose.yml file. This is a configuration file that defines the Docker containers that run the virtual machine. The file specifies the image that runs on each container. For more information about the syntax of a .yml file, see the [reference document docker-compose.yml](https://docs.docker.com/compose/compose-file/).

Create a directory on your virtual machine, and use the text editor to create the docker-compose.yml file. To test a simple example, copy the following code into the file. This configuration uses images from [DockerHub Registry](https://hub.docker.com/explore/) to install WordPress (the open-source content management and blogging system) and a back-end MySQL linked database.

wordpress:

image: wordpress

links:

- db:mysql

ports:

- 8080:80

db:

image: mariadb

environment:

MYSQL\_ROOT\_PASSWORD: <your password>

In the virtual machine work directory, run this command:

$ docker-compose up -d

This runs the specified Docker containers in docker-compose.yml. You will see the following message:

Creating wordpress\_db\_1...

Creating wordpress\_wordpress\_1...

You can check which containers are active, write docker-compose ps. You will see something like this:

Name Command State Ports

-------------------------------------------------------------------------

wordpress\_db\_1 /docker- Up 3306/tcp

entrypoint.sh

mysqld

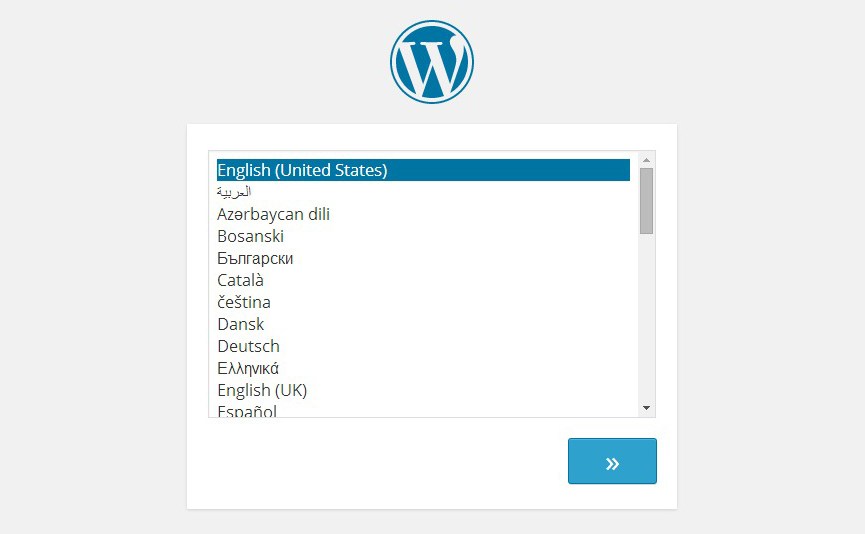
wordpress\_wordpr /entrypoint.sh Up 0.0.0.0:8080->80

ess\_1 apache2-for ... /tcp

You can now connect to WordPress directly on the VM by going to http://localhost:8080. If you want to connect to the VM over the Internet, first configure an HTTP endpoint on the VM that maps public port 80 to private port 8080. For example, in an Azure Service Management deployment, you can run the following Azure CLI command:

$ azure vm endpoint create <machine-name> 80 8080

You should now see the WordPress start screen, where you can complete the installation and get started with the application.

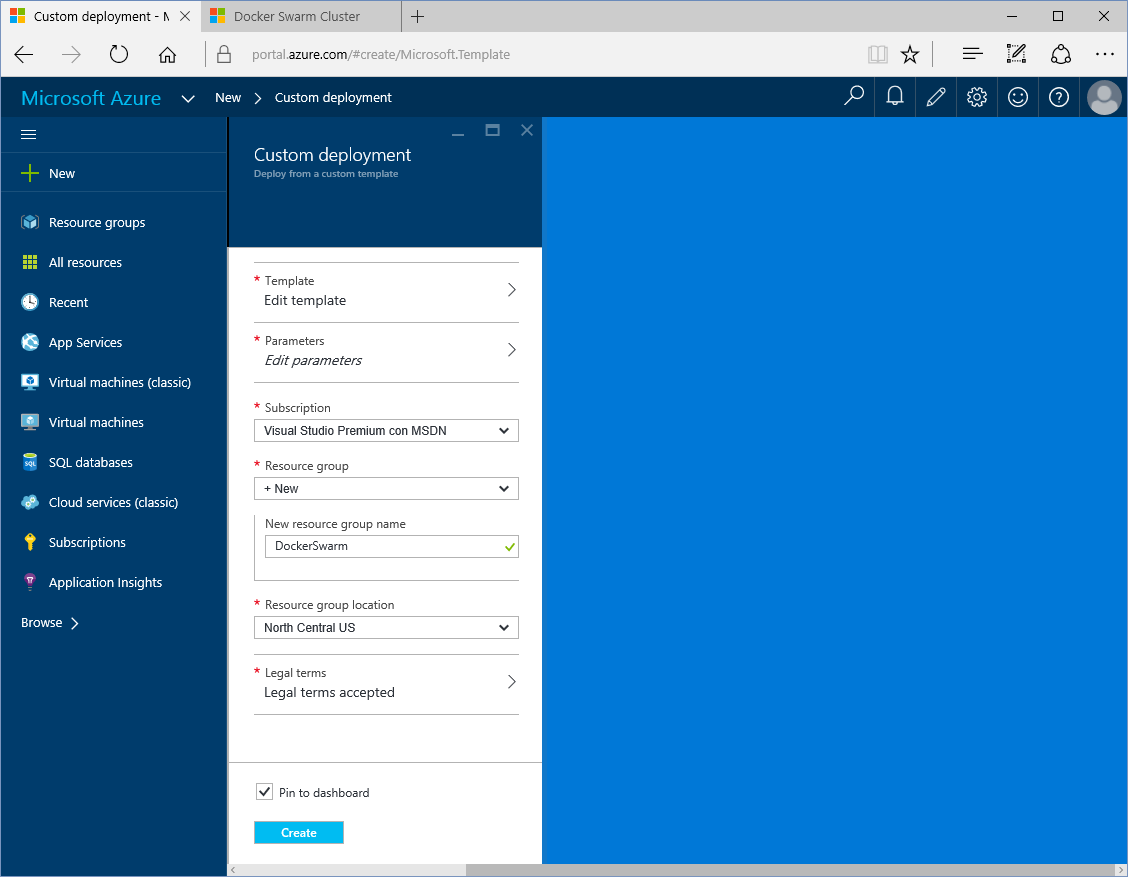
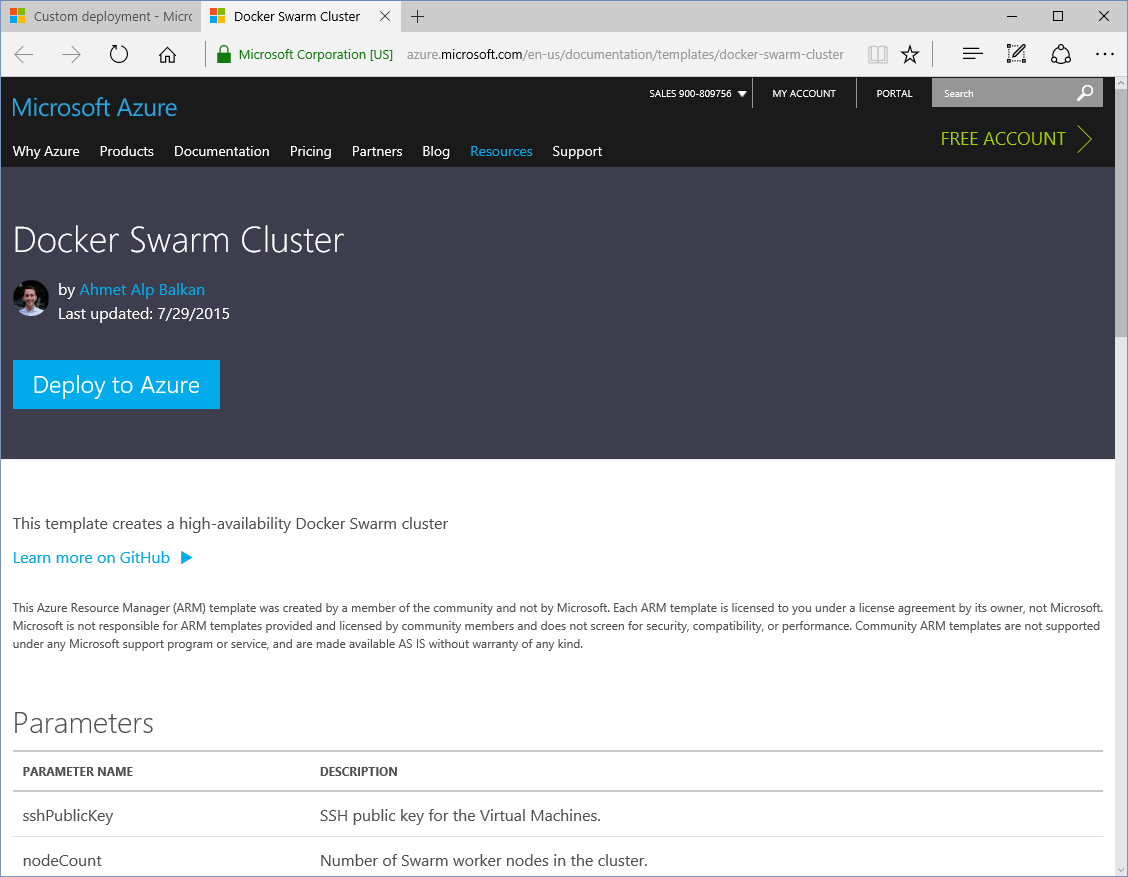


# Add container orchestration via Docker Swarm

[Docker Swarm](https://www.docker.com/docker-swarm) is a native clustering tool for [Docker](https://www.docker.com/docker-engine), which turns multiple Docker engines into a cluster by making it look like a single Docker engine. This lets enables you to create a pool of virtual machines that are container hosts. You can use these to scale out your applications with Docker as if you were using a single VM.

In this sense, Docker Swarm is a very practical and easy solution to the container orchestration problem. You can deploy containers to the cluster with the Docker command-line tool as usual.

There is [a template for Azure Resource Management](https://azure.microsoft.com/en-us/documentation/templates/docker-swarm-cluster/) that’s available for deploying Docker Swarm clusters in an easy way. You can find it in the Azure portal, or by using [Azure CLI](https://azure.microsoft.com/en-us/documentation/articles/xplat-install/) or Azure PowerShell to deploy it [from GitHub](https://github.com/Azure/azure-quickstart-templates/tree/master/docker-swarm-cluster).

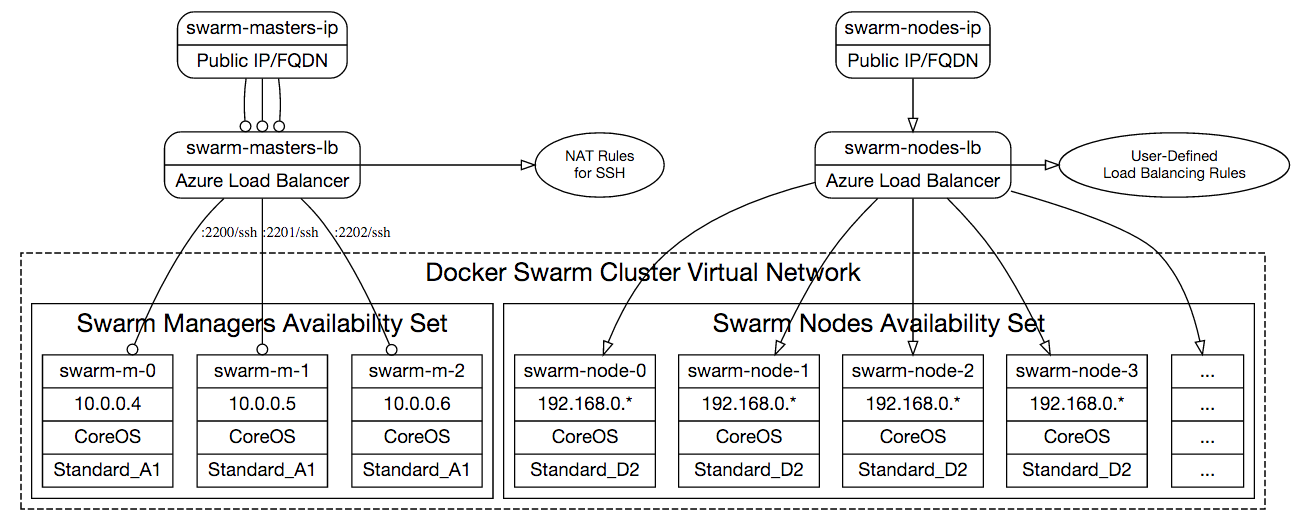


You can use the **ssh-keygen** command in Linux/Mac or Cygwin/MinGW to create a public and private pair key. You can configure the *sshPublicKey* parameter that’s in the template.

The template deploys a Docker Swarm cluster in Azure with three Swarm managers and a specific number of Swarm nodes in a resource group. The template uses CoreOS as the host operating system.

These manager VMs are the smallest size, “Standard\_A0”. They don’t run the user workloads and simply serve as cluster managers.

The template creates the following topology:



The cluster will be interconnected with a Docker multi-host networking setup so that you can easily create overlay networks with Docker network create commands.

The Swarm manager nodes run two containers: the [swarm agent](https://github.com/docker/swarm) and [Consul](https://www.consul.io/), which is used to discover the Swarm worker nodes.

You can configure *nodeCount* parameters to create as many Swarm worker instances as you like. Each Swarm worker VM is the “Standard\_A2” size.

Nodes in the Swarm cluster that accept Docker workloads do not have public IP addresses. They are accessible through Swarm manager VMs over SSH. To access a worker node, use SSH to connect to a master VM. Then use the private IP address of the worker node to connect to the worker node with SSH (Use the same SSH key you used to connect to master.) Alternatively, you can establish an SSH tunnel on your development machine and connect directly to the worker VM by using its private IP address.

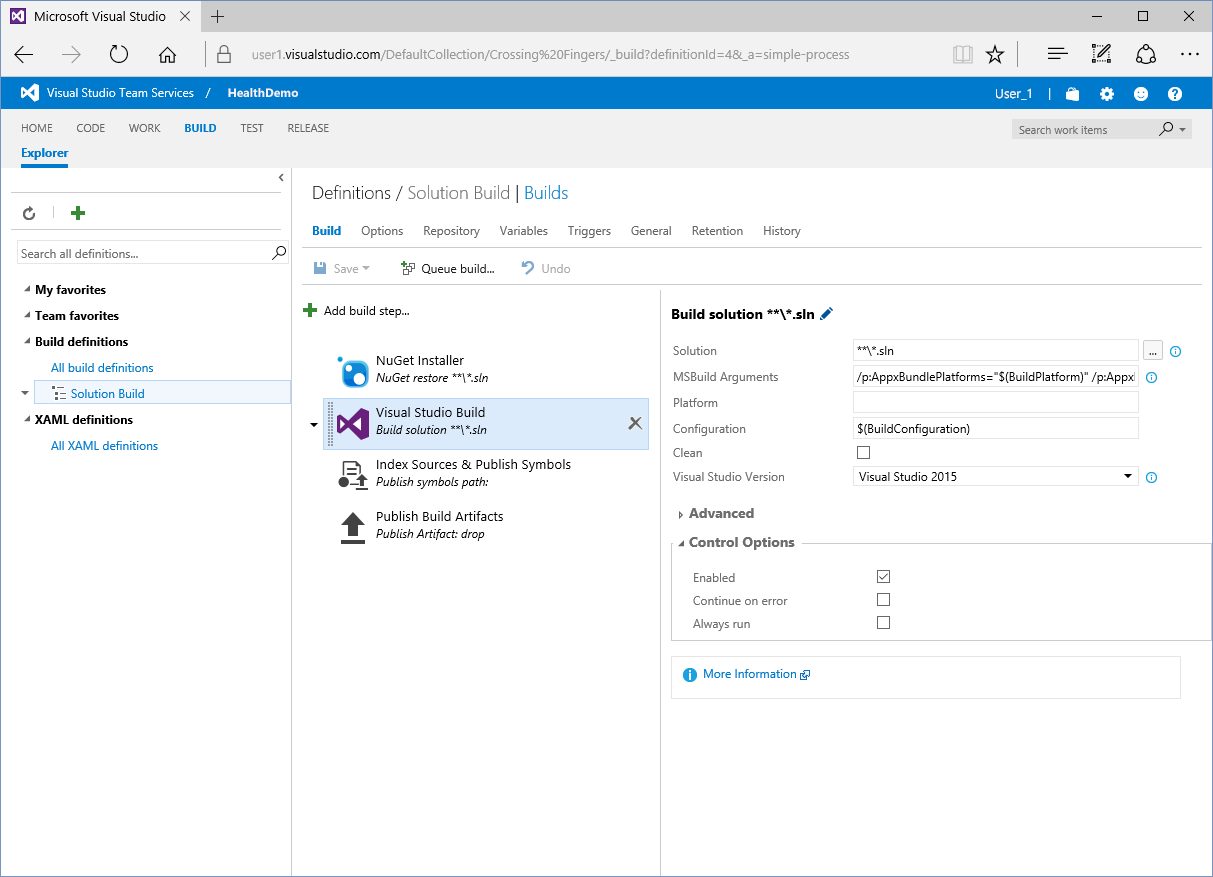
The swarm worker VMs node VMs are behind a load balancer (called swarm-lb-nodes). Any multi-instance services that are deployed across worker VMs can be served to the public Internet by creating probes and load-balancing rules on this load balancer resource. The load balancer's public DNS address is emitted as an output of the template deployment.

For more information, see the related instructions here: <https://github.com/Azure/azure-quickstart-templates/blob/master/docker-swarm-cluster/README.md>

# Implement continuous integration and continuous deployment

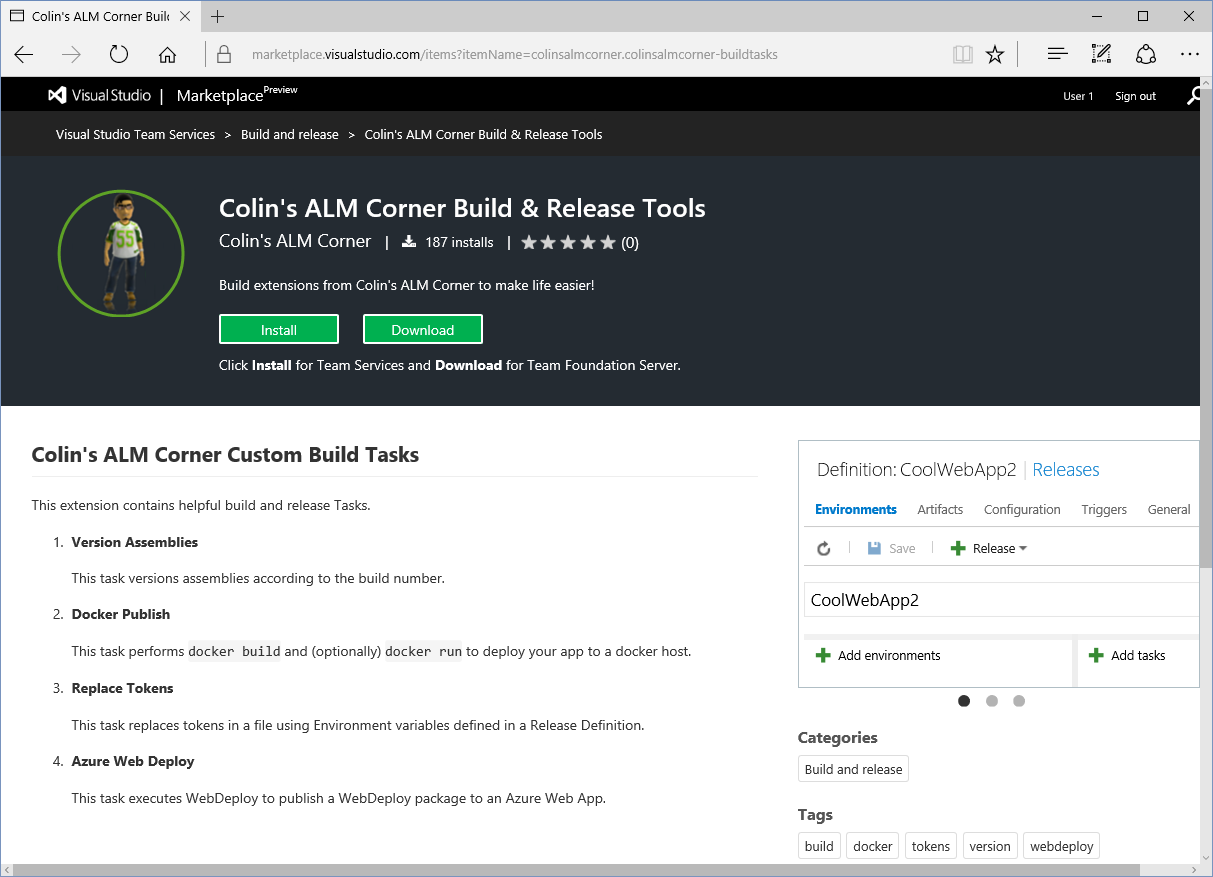
After the available Docker publishes a profile in your project, the project is available in Visual Studio Team Services. IT has a deployment machine and a Docker integration in Azure (all of which we discussed earlier). You can set the CI/CD of your project in Visual Studio Team Services easily.

To do this, go to the **Build** tab of your project. Then create a new build (if one has not already been created via the settings in your project). You can find instructions in the article [Continuous Delivery to Azure using Visual Studio Team Services](https://azure.microsoft.com/en-us/documentation/articles/cloud-services-continuous-delivery-use-vso/).

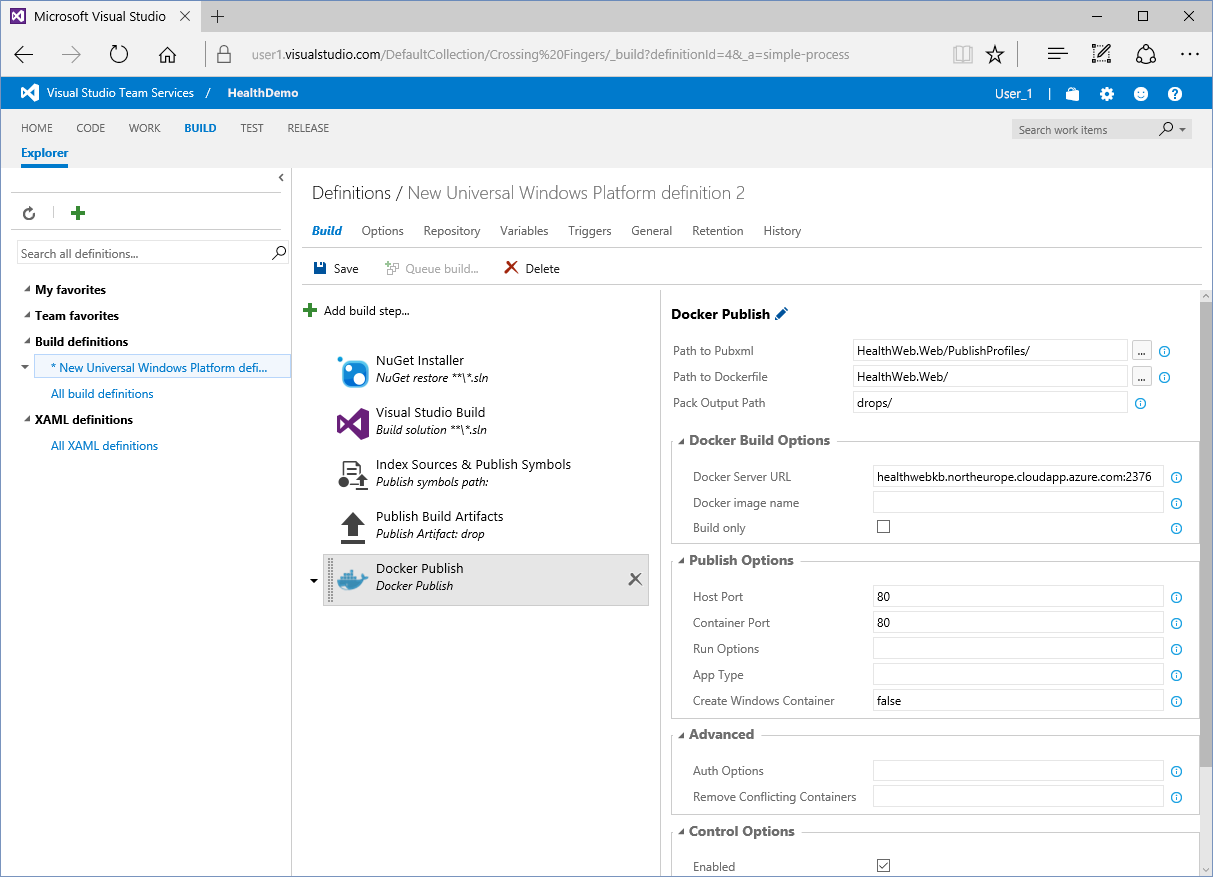


After you’ve configured your project, you can perform a manual deployment. However, it’s easier to add an extension to Visual Studio Team Services. The extensions is called Colin’s ALM Corner Build & Release Tools, and you can find it at <https://marketplace.visualstudio.com/items?itemName=colinsalmcorner.colinsalmcorner-buildtasks>

This extension, among other things, enables you to create a Docker publish task that’s running multiple scripts on the Docker machine host that complains with the Docker project containers deployment:



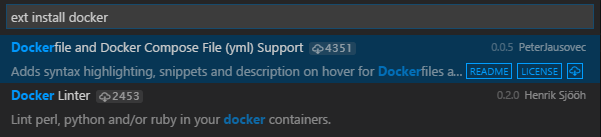
To do this, add the Docker publish task to the build steps; the step settings, as all the created steps required in previous steps are trivial, unless in advances cases, where it passes to indicate the location of various directories and files to use as well as the destination directory.



A build will be generated. It will create a container for your project on a Docker machine host.

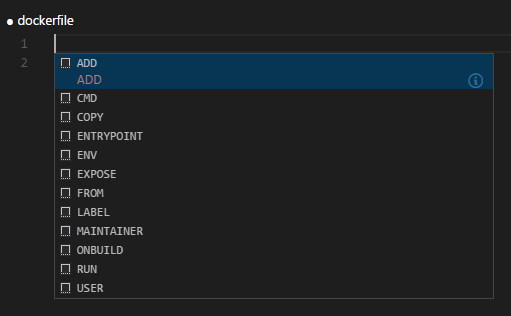
# Use Docker with Visual Studio Code

Visual Studio Code supports the creation of Docker-related files that use extensions. As with other extensions, select **F1,** and type “**ext install**” to run **Extensions: Install Extension**. This displays the list of extensions that are available to use. If you continue writing docker, you can filter results to find the [Dockerfile and Docker Compose File (yml) Support](https://marketplace.visualstudio.com/items?itemName=PeterJausovec.vscode-docker) extensions. Select the extension to install it. (It’s possible that you might have to restart Visual Studio Code).

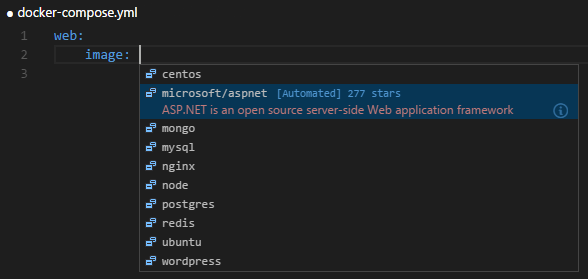


This extension provides Syntax Highlight, snippets, and other IntelliSense features to the Dockerfiles and Docker Compose files.

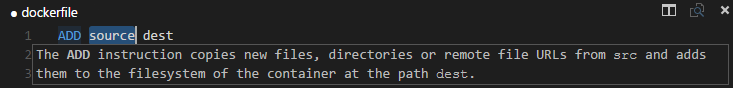
Pressing **Ctrl+Espacio** will display a list of snippets with the available commands for Dockerfiles files and composition directives for docker-compose.yml files.



Selecting **Ctrl+Espacio** in the image directive will display a list of publicly available images in Docker Hub.



Hover the mouse over the document fields, and they will display information about commands, images, and directives.



Manually creating Docker containers and Docker Compose containers will become much easier.

# Other suggested topics to explore

* Learn about how to deploy an ASP.NET container to a remote Docker host. Create and publish a new Docker container. Provide a custom Docker host. Test a Docker host:

<https://azure.microsoft.com/en-us/documentation/articles/vs-azure-tools-docker-hosting-web-apps-in-docker/>

* Get started with Docker Compose to define and run multi-container applications on Azure virtual machines:

<https://azure.microsoft.com/en-us/documentation/articles/virtual-machines-docker-compose-quickstart/>

* Read the Docker Compose official documentation:

<https://docs.docker.com/compose/overview/>

* Read about Docker Swarm cluster in GitHub:

<https://github.com/Azure/azure-quickstart-templates/blob/master/docker-swarm-cluster/README.md>

* Read about Docker Swarm cluster in Azure:

<https://azure.microsoft.com/es-es/blog/docker-swarm-clusters-on-azure/>

* Learn about working with Docker in Visual Studio Code:

<https://code.visualstudio.com/Docs/languages/dockerfile>

* Learn more about the tools and find the recently released installers.

[Azure developer tools](http://azure.com/tools)