**Deploy a Spring Boot application to Linux on Azure App Service**

* 10/14/2020
* 6 minutes to read
  + [](https://github.com/bmitchell287)
  + [](https://github.com/KarlErickson)
  + [](https://github.com/moarychan)
  + [](https://github.com/DCtheGeek)
  + [](https://github.com/ZhengJun003)
  + +2

This tutorial walks through using [Docker](https://www.docker.com/) to containerize your [Spring Boot](http://projects.spring.io/spring-boot/) application and deploy your own docker image to a Linux host in the [Azure App Service](https://docs.microsoft.com/en-us/azure/app-service/containers/app-service-linux-intro).

**Prerequisites**

In order to complete the steps in this tutorial, you need to have the following prerequisites:

* An Azure subscription; if you don't already have an Azure subscription, you can activate your [MSDN subscriber benefits](https://azure.microsoft.com/pricing/member-offers/msdn-benefits-details/) or sign up for a [free Azure account](https://azure.microsoft.com/pricing/free-trial/).
* The [Azure Command-Line Interface (CLI)](https://docs.microsoft.com/en-us/cli/azure/overview).
* A supported Java Development Kit (JDK). For more information about the JDKs available for use when developing on Azure, see <https://aka.ms/azure-jdks>.
* [Apache Maven] build tool (Version 3).
* A [Git](https://github.com/) client.
* A [Docker](https://www.docker.com/) client.

**Note**

Due to the virtualization requirements of this tutorial, you cannot follow the steps in this article on a virtual machine; you must use a physical computer with virtualization features enabled.

**Create the Spring Boot on Docker Getting Started web app**

The following steps walk through the steps that are required to create a simple Spring Boot web application and test it locally.

1. Open a command-prompt and create a local directory to hold your application, and change to that directory; for example:

BashCopy

mkdir SpringBoot

cd SpringBoot

1. Clone the [Spring Boot on Docker Getting Started](https://github.com/spring-guides/gs-spring-boot-docker) sample project into the directory you created; for example:

BashCopy

git clone https://github.com/spring-guides/gs-spring-boot-docker.git

1. Change directory to the completed project; for example:

BashCopy

cd gs-spring-boot-docker/complete

1. Build the JAR file using Maven; for example:

BashCopy

mvn package

1. Once the web app has been created, change directory to the target directory where the JAR file is located and start the web app; for example:

BashCopy

cd target

java -jar gs-spring-boot-docker-0.1.0.jar --server.port=80

1. Test the web app by browsing to it locally using a web browser. For example, if you have curl available and you configured the Tomcat server to run on port 80:

BashCopy

curl http://localhost

1. You should see the following message displayed: **Hello Docker World**

**Create an Azure Container Registry to use as a Private Docker Registry**

The following steps walk through using the Azure portal to create an Azure Container Registry.

**Note**

If you want to use the Azure CLI instead of the Azure portal, follow the steps in [**Create a private Docker container registry using the Azure CLI 2.0**](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-azure-cli).

1. Browse to the [Azure portal](https://portal.azure.com/) and sign in.

Once you have signed in to your account on the Azure portal, follow the steps in the [Create a private Docker container registry using the Azure portal](https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-portal) article, which are paraphrased in the following steps for the sake of expediency.

1. Click the menu icon for **New**, select **Containers**, and then select **Azure Container Registry**.
2. When the **Create container registry** page is displayed, enter **Registry name**, **Subscription**, **Resource group**, and **Location**. Then select **Create**.

**Configure Maven to build image to your Azure Container Registry**

1. Navigate to the completed project directory for your Spring Boot application, (for example: "*C:\SpringBoot\gs-spring-boot-docker\complete*" or "*/users/robert/SpringBoot/gs-spring-boot-docker/complete*"), and open the *pom.xml* file with a text editor.
2. Update the <properties> collection in the *pom.xml* file with the latest version of [jib-maven-plugin](https://github.com/GoogleContainerTools/jib/tree/master/jib-maven-plugin), login server value, and access settings for your Azure Container Registry from the previous section of this tutorial. For example:

XMLCopy

<properties>

<jib-maven-plugin.version>2.5.2</jib-maven-plugin.version>

<docker.image.prefix>wingtiptoysregistry.azurecr.io</docker.image.prefix>

<java.version>1.8</java.version>

</properties>

1. Add [jib-maven-plugin](https://github.com/GoogleContainerTools/jib/tree/master/jib-maven-plugin) to the <plugins> collection in the *pom.xml* file. This example uses version 2.2.0.

Specify the base image at <from>/<image>, here mcr.microsoft.com/java/jre:8-zulu-alpine. Specify the name of the final image to be built from the base in <to>/<image>.

Authentication {docker.image.prefix} is the **Login server** on the registry page shown previously. The {project.artifactId} is the name and version number of the JAR file from the first Maven build of the project.

XMLCopy

<plugin>

<artifactId>jib-maven-plugin</artifactId>

<groupId>com.google.cloud.tools</groupId>

<version>${jib-maven-plugin.version}</version>

<configuration>

<from>

<image>mcr.microsoft.com/java/jre:8-zulu-alpine</image>

</from>

<to>

<image>${docker.image.prefix}/${project.artifactId}</image>

</to>

</configuration>

</plugin>

1. Navigate to the completed project directory for your Spring Boot application and run the following command to rebuild the application and push the container to your Azure Container Registry:

BashCopy

az acr login -n wingtiptoysregistry && mvn compile jib:build

**Note**

1. The command az acr login ... will try to login to Azure Container Registry, otherwise you will need to provide <username> and <password> for jib-maven-plugin, see [**Authentication Methods**](https://github.com/GoogleContainerTools/jib/tree/master/jib-maven-plugin#authentication-methods) in jib.
2. When you are using Jib to push your image to the Azure Container Registry, the image will not use the *Dockerfile*, see [**this**](https://cloudplatform.googleblog.com/2018/07/introducing-jib-build-java-docker-images-better.html) document for details.

**Create a web app on Linux on Azure App Service using your container image**

1. Browse to the [Azure portal](https://portal.azure.com/) and sign in.
2. Click the menu icon for **Create a resource**, select **Compute**, and then select **Web App for Containers**.
3. When the **Web App on Linux** page is displayed, enter the following information:
   * Choose your **Subscription** from the drop-down list.
   * Choose an existing **Resource Group**, or specify a name to create a new resource group.
   * Enter a unique name for the **App name**; for example: "*wingtiptoyslinux*"
   * Specify Docker Container to **Publish**.
   * Choose *Linux* as the **Operating System**.
   * Select **Region**.
   * Accept **Linux Plan** and choose an existing **App Service Plan**, or select **Create new** to create a new app service plan.
   * Click **Next: Docker**.

On the **Web App** page select **Docker**, and enter the following information:

* + Select **Single Container**.
  + **Registry**: Choose your container, for example: "*wingtiptoysregistry*"
  + **Image**: Select the image created previously, for example: "*gs-spring-boot-docker*"
  + **Tag**: Choose the tag for the image; for example: "*latest*"
  + **Startup Command**: Keep it blank since the image already has the start up command

After you have entered all of the above information, select **Review + create**.

* + Click **Review + create**.

Review the information and select **Create**.

When the deployment is complete, select **Go to resource**. The deployment page displays the URL to access the application.

**Note**

Azure will automatically map Internet requests to embedded Tomcat server that is running on the port - 80. However, if you configured your embedded Tomcat server to run on port - 8080 or custom port, you need to add an environment variable to your web app that defines the port for your embedded Tomcat server. To do so, use the following steps:

1. Browse to the [**Azure portal**](https://portal.azure.com/) and sign in.
2. Select the icon for **Web Apps**, and select your app from the **App Services** page.
3. Select **Configuration** in the left navigation pane.
4. In the **Application settings** section, add a new setting named **WEBSITES\_PORT** and enter your custom port number for the value.
5. Select **OK**. Then select **Save**.

**Clean up resources**

When no longer needed, use the [Azure portal](https://portal.azure.com/) to delete the resources created in this article to avoid unexpected charges.