

Azure DevOps and Jenkins in perfect harmony



Brian Benz

Nov 2, 2018 · 11 min read

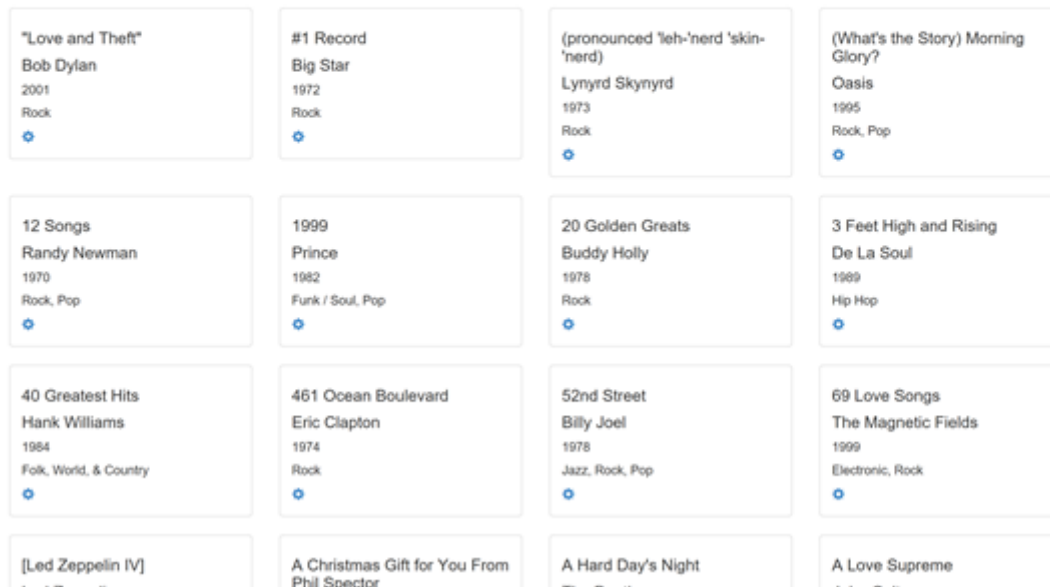
I recently had the opportunity to present several talks at DevOps World | Jenkins World 2018 in San Francisco. It was a great experience as always, and I want to say thanks to those of you who visited our booth and checked out the talks from my myself and my fellow Microsoft Cloud Developer Advocate Jessica Deen. I've had several follow-up questions on my demos that have inspired me to document and share them here today.

The Demo Application: Spring Music

When I build a DevOps demo I like to use a base app that will reflect real-world experiences that real DevOps professionals encounter in their daily work.. I really like using this version of Spring Music because it's a Java-based Spring Boot application that can run by itself or connect to a variety of Azure data services on the back-end including NoSQL and SQL storage options. It also uses a built-in Gradle wrapper for builds, which makes it portable to run on all of Azure compute services, including Windows and Linux VMs, Azure Kubernetes Service, and Azure App Service on Linux. This means that it's a single application that can show multiple real-world use cases for running and deploying an application in the cloud. It's also a great application to demonstrate multiple real-world use cases for working with popular CI/CD/DevOps tools and services.

Here's a screen shot of the application itself. You can view Rolling Stone's top 500 Albums of all time, by Title, artist, year and Genre. There's also a list view, and you can edit and add albums.





Combining Azure DevOps and Jenkins to deploy the app.

So now we know enough about the app — let's move on to how we get it in the cloud using Azure DevOps and Jenkins. For a complete picture of how Jenkins and Azure DevOps complement each other, have a look at my post on the Microsoft Open Source Blog. In this post we'll cover a couple of key scenarios in depth.

I work a lot with developers who need to manage and deliver software projects that contain multiple languages and platforms. A large number of them use Azure DevOps, which is the next evolutionary step up from Visual Studio Team Services, which had its Origins in Team Foundation Server. They like features like free private Git repos, test plans, Kanban boards and custom team dashboards.

A large number of these developers also have CI/CD pipelines running in Jenkins. These developers have leveraged the incredible selection of Jenkins Plugins available to make building and testing of their code as effortless as possible.

The good news is that you don't have to choose one or the other! Savvy development shops have leveraged the best features of both platforms, and Microsoft has made that easy by developing several plugins that combine the best of both. For example, you can provision and manage Azure Virtual Machines as Jenkins Agents with the Jenkins Agent Plugin for Azure. You can also deploy build artifacts to several destinations, including cloud storage using the Azure Storage plugin for Jenkins.

Prerequisites

An Azure DevOps Account

If you don't have one already, can get a free Azure DevOps account [here](#).

Jenkins

If you already have Jenkins up and running, you can add our plugins from the marketplace. If you don't, we also have you covered with a sample architecture and template that can be deployed securely and easily to Azure on Ubuntu Linux, and includes the most popular Microsoft Azure plugins for Jenkins pre-installed. Either way, I'm going to show you steps for working with Azure DevOps and Jenkins together to deliver an application to Azure.

Jenkins Azure Credentials Plugin

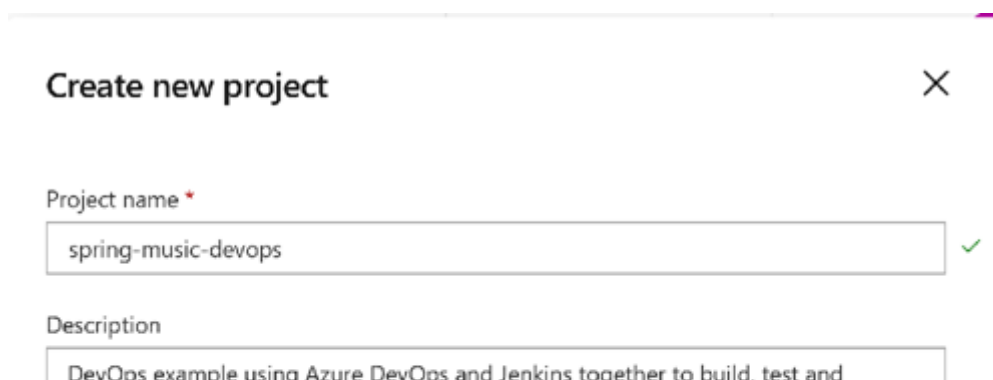
If you use the template above to create Jenkins, the plugins you need are preinstalled. If not, you'll need the Azure Credentials plugin as a minimum, which enables storage of Azure credentials in Jenkins. Check the end of this post for more Jenkins plugins to explore.

Create a new Azure DevOps project

Let's set up a base by creating a new project in Azure DevOps and importing the Spring Music github repo into a new Azure DevOps repository. Go to <https://dev.azure.com>, log in or create your free account, then click on "Create Project" on the top right and give the project a descriptive name (I called mine spring-music-devops)

By default, the Azure DevOps repository is private, which is great if your app is enabled to share confidential connections and keys that you need when building and testing your application before it is deployed. Of course, you can also make the repository public as well, but we'll keep it private for now.

Fill in a name, description and click create:



Create new project ✕

Project name *


spring-music-devops ✓

Description

DevOps example using Azure DevOps and Jenkins together to build, test and


deploy a sample application

Visibility



Public ⓘ

Anyone on the internet can view the project. Certain features like TFVC are not supported.



Private

Only people you give access to will be able to view this project.

Public projects are disabled for your organization. You can turn on public visibility with [organization policies](#).

▼ Advanced

Create

Cancel

Azure DevOps creates a project you can use to manage Kanban boards, pipelines, test, and artifacts, but let's focus on setting up the repo for now. Click on **Repos**:



Welcome to the project!

What service would you like to start with?

Boards

Repos

Pipelines

Test Plans

Artifacts

[or manage your services](#)

This will create a new repo for our project based on the Spring Music GitHub Repo.

Import a GitHub repo into your Azure DevOps Project

Click on the **import** button under **import a repository**:



Then enter the clone URL: `https://github.com/bbenz/spring-music-azure`

Import a Git repository



Source type

Clone URL *

☐ Requires authorization

Import

Close

Once the import is done, you can see the imported code under `repos > files` in your project.

Create an Azure DevOps Build Pipeline for Jenkins

Next up, we're going to pass code to Jenkins to process a build and test. Go to Pipelines>Builds in your Azure DevOps Project and click on **New Pipeline**:









No build pipelines were found

Automate your build in a few easy steps with a new pipeline.



New pipeline

Leave all the defaults as-is and click **Continue**:



Select a source

 Azure Repos Git	 GitHub	 GitHub Enterprise	 Subversion	 Bitbucket Cloud
 External Git				

Team project

 spring-music-devops 

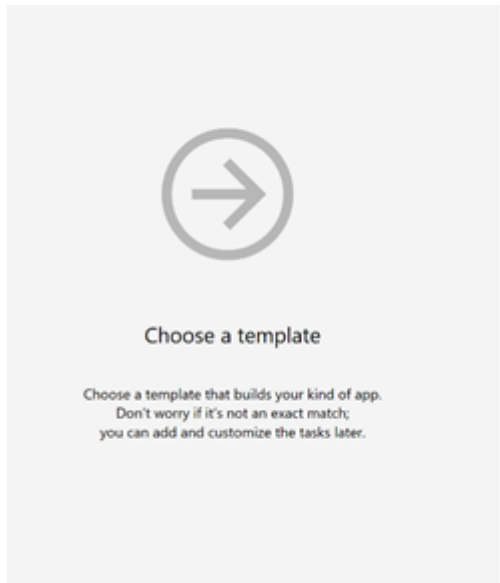
Repository

 spring-music-devops 

Default branch for manual and scheduled builds

[Continue](#)

Scroll down and choose the **Jenkins** template, click **Apply**:









Choose a template

Choose a template that builds your kind of app.
Don't worry if it's not an exact match;
you can add and customize the tasks later.


Select a template

Or start with an [Empty job](#)

-  **Go**
Build a Go application.
-  **Gradle**
Build and test a Java project with Gradle.
-  **Jenkins**
Queue a Jenkins job and download its artifacts. [Apply](#)
-  **Load test using Azure IaaS virtual machines**
Create a rig on Azure IaaS virtual machines to run load tests using VSTS cloud-based load testing service.
-  **Node.js With Grunt**
Build a Node.js application using the Grunt task runner.
-  **Node.js With gulp**
Build a Node.js application using the gulp task runner.

Create a Personal Access Token for connecting Jenkins and Azure DevOps

Next up, we need to create a Personal Access Token that will be used by the Jenkins build process to access the private repo we just set up. Click on your ID on the top right in Azure DevOps and select security:

 **Brian Benz**
bbenz@microsoft.com

[My profile](#)[Security](#)[Usage](#)[Notification settings](#)[Preview features](#)

[Sign out](#)

Click on new Token:

Personal Access Tokens

These can be used instead of a password for applications like Git

[+ New Token](#)

Next, enter a name, keep the default organization, and set the expiration and scopes. Because this is a demo, full access is fine, but normally you would set the scopes (read, write, manage, etc) for each Azure DevOps feature you want to use this token with:

Create a new personal access token

Name *

Organization

Expiration

Scopes

Authorize the scope of access associated with this token

Scopes

☒ Full access

☐ Custom defined

[Create](#)[Cancel](#)

Click create and you will get a token that you can use for Jenkins. Copy it to a local file for now, but remember to treat tokens as you would any password in a secure

credentials or password manager.

Next, you'll need the full link to the repository for the Jenkins job we're building in the next step. In your Azure DevOps repo, click on clone on the top right, and copy the HTTPS link in the same place you have your Personal Access Token.

Clone repository

Clone Git repository using command line or IDE

Command line

HTTPS SSH

com/spring-music-devops/_git/spring-music-devops

Generate Git credentials **Copy clone URL to clipboard**

IDE

Clone in VS Code | v

ⓘ Having problems authenticating in Git? Be sure to get the latest version of [Git for Windows](#) or our plugins for [IntelliJ](#), [Eclipse](#), [Android Studio](#) or [Windows command line](#).

Set up a Jenkins Build Project

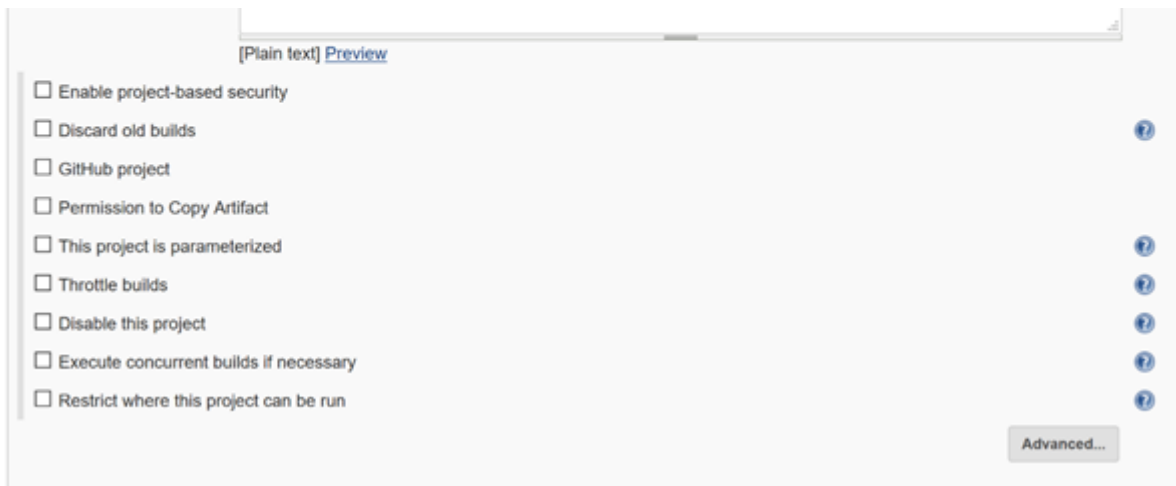
Now that we have the repo and the credentials to access it from Jenkins, we're ready to set up a Jenkins build.

Log in to Jenkins and set up a new freestyle project. I called mine spring-music-azure-build. Select Freestyle project and click continue. Add a description and deselect "restrict where this project can run".

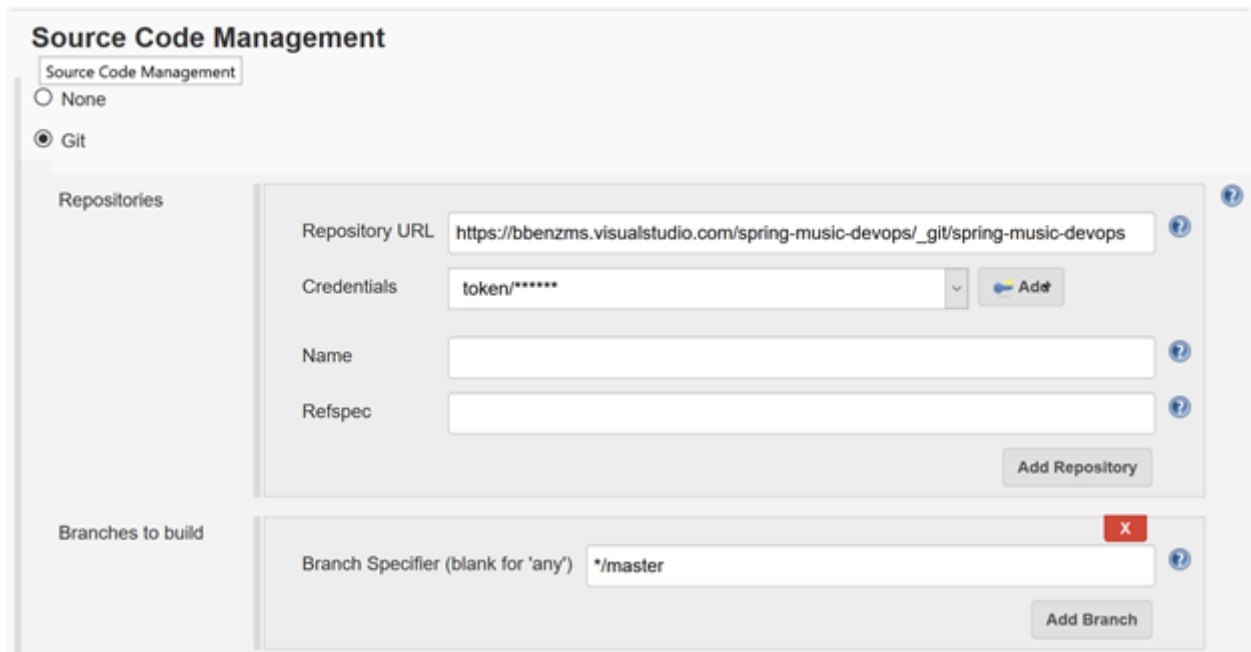
General Source Code Management Build Triggers Build Environment Build Post-build Actions

>>

Description Building spring-music-azure from a Azure DevOps private repo



Next, under source code management, paste the HTTPS clone URL you saved a couple of steps ago. You'll likely get an error saying you can't access the private repo, which is OK and means Azure DevOps security is working.



Next, let's add the credentials so that you can access your private Azure DevOps Repo. Click on the **Add** button to the right of **Credentials**. Choose **Username with Password** from the **Kind** drop-down. Use **token** for the Username and paste the Personal Access token you generated and saved into the Password field. Leave the rest blank and click the **Add** button. When you've saved the credentials you should see any access-related error messages disappear.

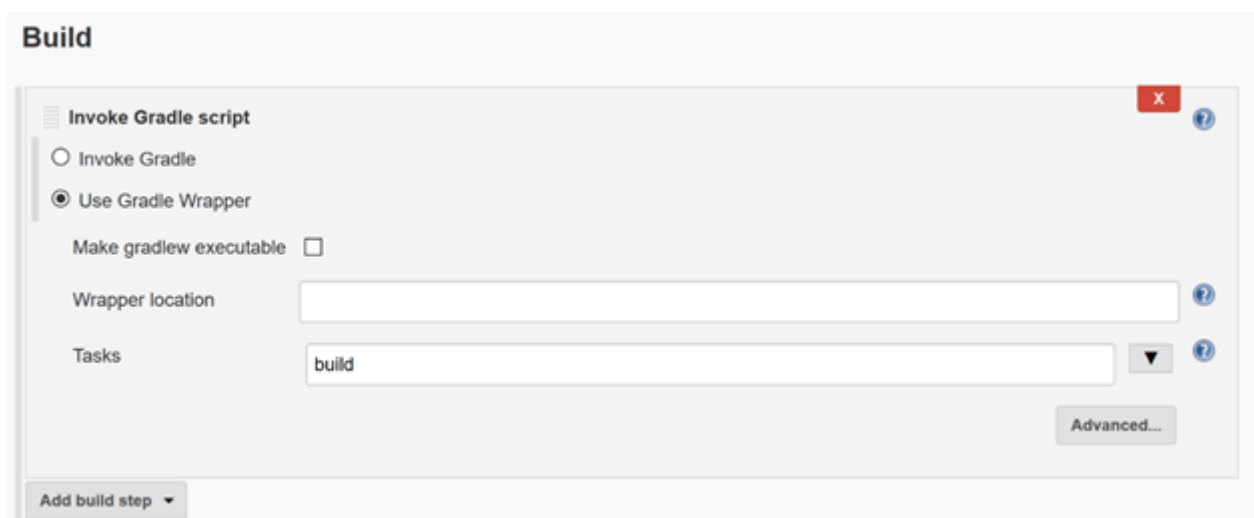




At this point we have access to the Azure DevOps repo, and now we have to tell Jenkins what to do for this build. Under Build, click the Add Build Step button then select invoke Gradle script:

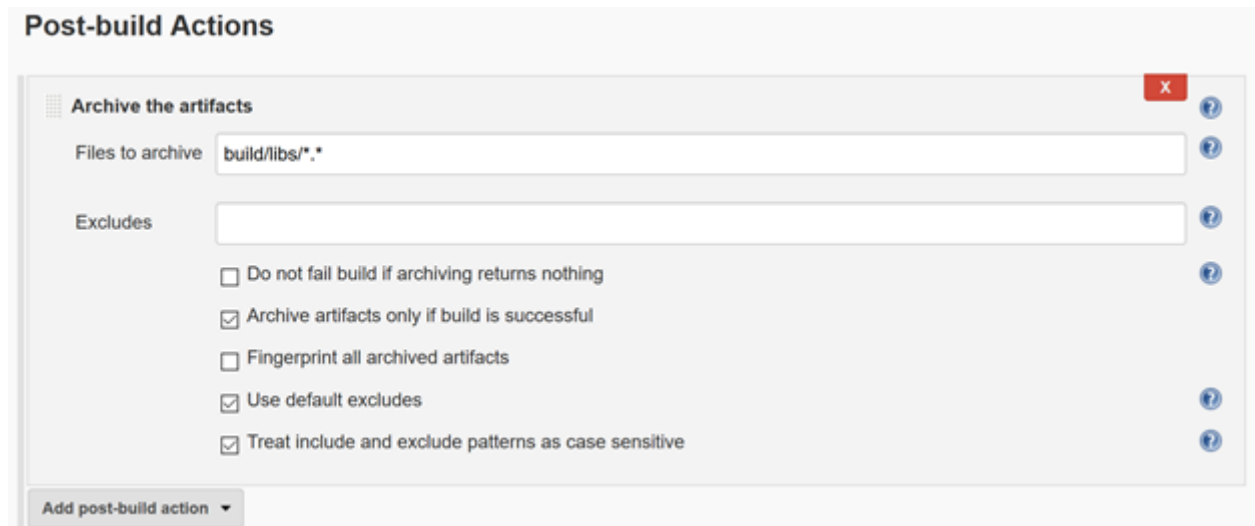
- Copy artifacts from another project
- Deploy to Azure Container Service (AKS)
- Deploy to Kubernetes
- Download from Azure storage
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script**
- Invoke top-level Maven targets
- Run with timeout
- Set build pending status in TFS/Team Services
- Set build status to "pending" on GitHub commit

In the box that appears, select **Use Gradle Wrapper** and under tasks, specify **build**:



Next, set up a post-build action to save the build artifacts by selecting “Archive the Artifacts” from the list of choices. Add **build/libs/spring-music.jar** to **Files to**

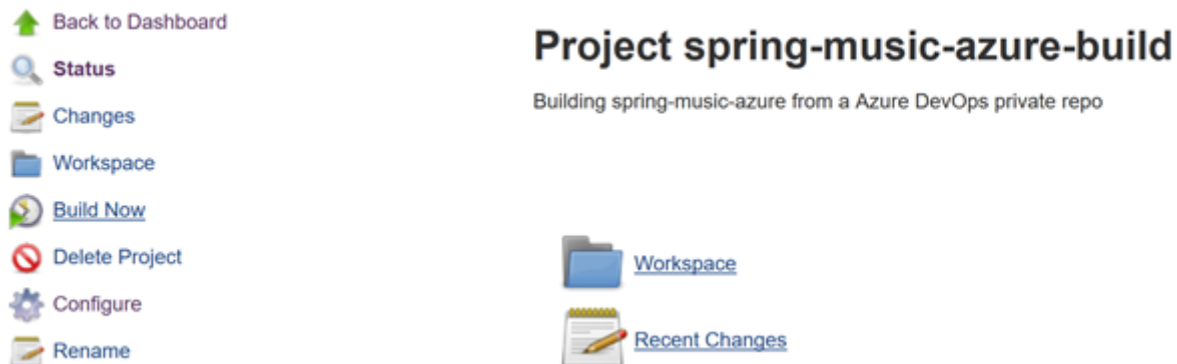
Archive. Check the **Archive artifacts only if build is successful** box. Then click save:



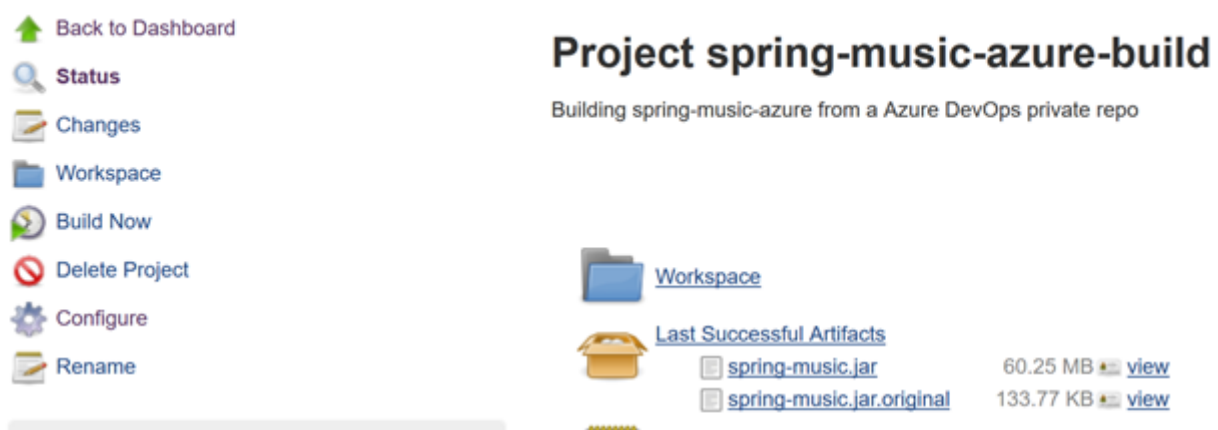
The screenshot shows the 'Post-build Actions' configuration in Jenkins. The 'Archive the artifacts' action is selected. The 'Files to archive' field contains 'build/libs/*.*'. The 'Excludes' field is empty. The following options are checked: 'Archive artifacts only if build is successful', 'Use default excludes', and 'Treat include and exclude patterns as case sensitive'. Other options like 'Do not fail build if archiving returns nothing', 'Fingerprint all archived artifacts', and 'Use default excludes' are unchecked. A red 'X' icon is visible in the top right corner of the configuration box.

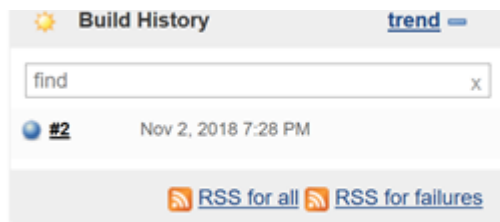
Test the Jenkins Build

It's now time to try out your build task. When the new project is saved, run a test by clicking on **build now** from the options on the left:



Click on the job that pops up in build history below, then click on console output to see the full build process in action. When done, you should see the comforting blue sphere in the Build History and build artifacts generated on the right:



[Recent Changes](#)

Permalinks

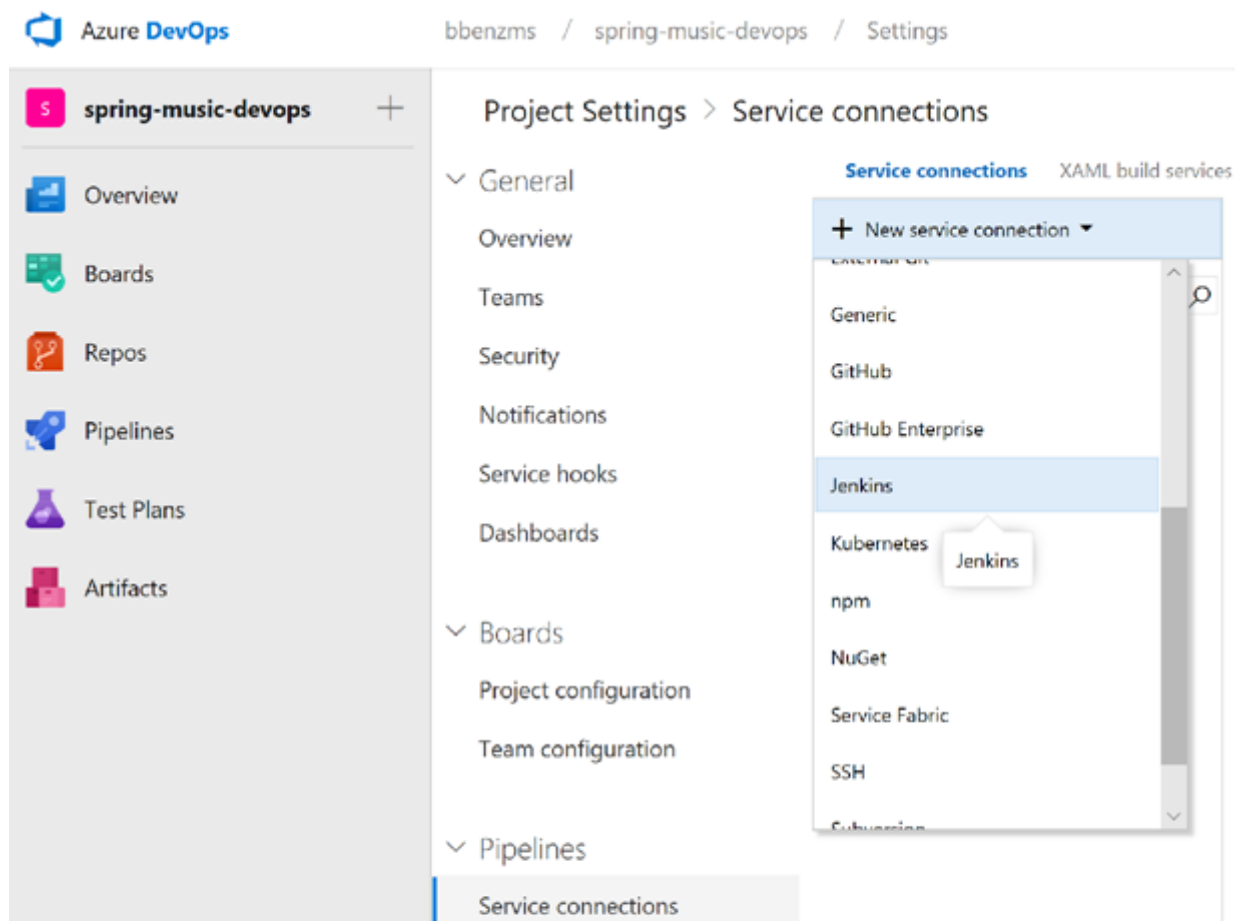
- [Last build \(#2\), 9 min 31 sec ago](#)
- [Last stable build \(#2\), 9 min 31 sec ago](#)
- [Last successful build \(#2\), 9 min 31 sec ago](#)
- [Last completed build \(#2\), 9 min 31 sec ago](#)

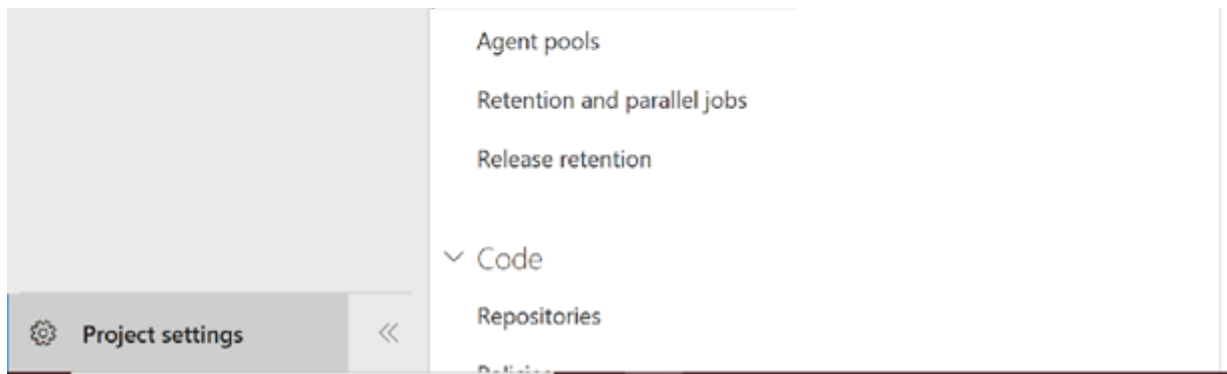
Set up an Azure DevOps pipeline release

Now that we have Jenkins accessing our private repo and building artifacts from the latest code in the repo, we're ready to trigger an Azure DevOps pipeline release when a Jenkins build completes successfully. To do that we need to go back to Azure DevOps and set up access to Jenkins, then come back to Jenkins and select a second Jenkins post-build action to trigger the Azure DevOps release.

First, let's go back to Azure DevOps and set up access to Jenkins via a Jenkins service endpoint.

Click on **Project Settings** on the bottom right of the project screen. Navigate to Pipelines > Service Connections > New Service connections and choose Jenkins from the drop-down list:





Add the information for your Jenkins service. You can call the service whatever you want it to be, and add the other info for your Jenkins instance. To be more secure, you may want to make a Jenkins user just for this connection. Click verify connection, then OK:

×

Add Jenkins service connection

Connection name

spring-music-devops-jenkins

Server URL

http://yourjenkins.westus2.cloudapp.azure.com/

Accept untrusted SSL certificates

☒

ⓘ

Username

adminID

×

ⓘ

Password

●●●●●●●●●●●●●●●●

ⓘ

[Learn More ⓘ](#)

Connection: ✔ Verified

[Verify connection](#)

OK

Close

Next up, we create a release pipeline in Azure DevOps pipelines. In Azure DevOps, navigate to Pipelines > Releases and click new pipeline. Choose a template (I chose empty job for now) and name your pipeline stage.

Stage

 Delete  Move  

Jenkins Release Stage 1

Properties ^

Name and owners of the stage

Stage name

Jenkins Release Stage 1 

Stage owner

 BB Brian Benz 

Next, choose **add an artifact** and **Jenkins**. Select the Service Connection, Job and Default version from the drop-down lists, then click Add. Don't forget to press Save on the top right:

Add an artifact

Source type



Azure Artifacts





Azure Contai...




Docker Hub



Jenkins

[4 more artifact types](#) Service connection * | [Manage](#) spring-music-devops-jenkins  Jenkins Job * spring-music-azure-build ☐ Download artifacts from Azure storage Default version * Latest Source alias * 

_spring-music-azure-build

 The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **spring-music-azure-build** published the following artifacts: **spring-music.jar**, **spring-music.jar.original**.

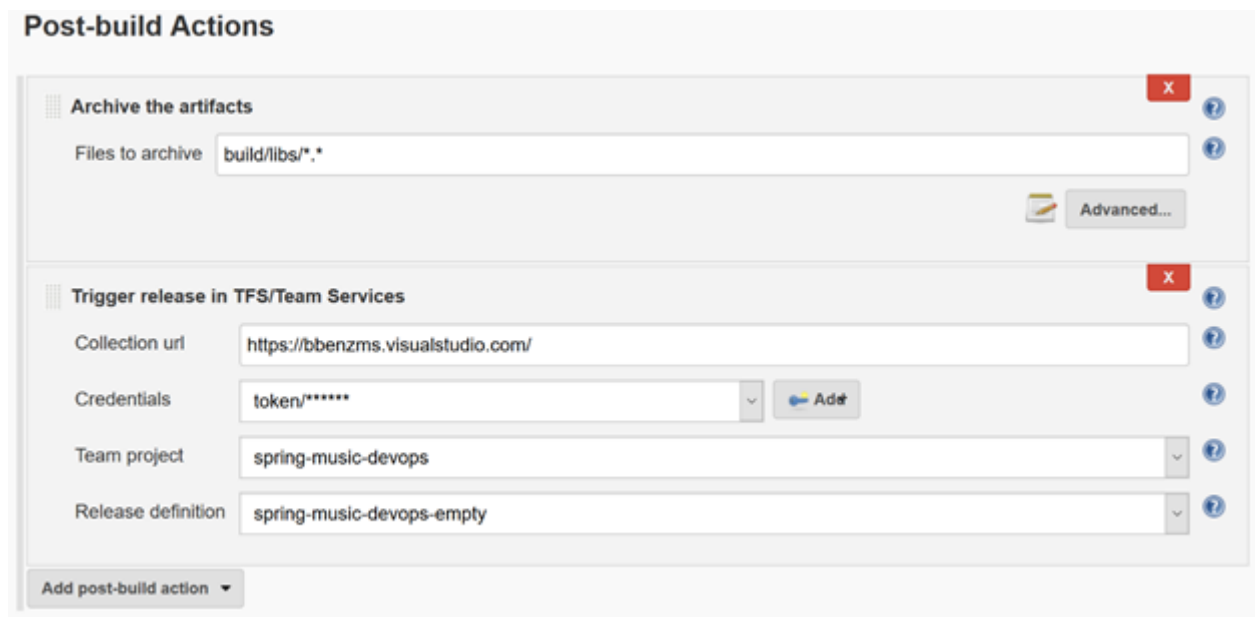
Add

Trigger the release with a Post Build Action in Jenkins

Next we need to trigger this new Azure DevOps pipeline release when Jenkins successfully completes a build. To do that, we go back to Jenkins and add another post-build action to our Jenkins build job.

In Jenkins, click on the build project then click configure, and scroll down to post-build actions. Click on Add post-build actions, then select Trigger release in TFS/Team services.

The collection URL is your Azure DevOps URL. Select the personal access token you already created in previous steps for credentials, the Team Project, and the Azure DevOps pipeline release from the drop-down lists, then save the new post-build action.



The screenshot shows the 'Post-build Actions' configuration page in Jenkins. It contains two main sections: 'Archive the artifacts' and 'Trigger release in TFS/Team Services'. The 'Archive the artifacts' section has a text field for 'Files to archive' with the value 'build/libs/*.*' and an 'Advanced...' button. The 'Trigger release in TFS/Team Services' section has four fields: 'Collection url' (https://bbenzms.visualstudio.com/), 'Credentials' (token/***** with an 'Add' button), 'Team project' (spring-music-devops), and 'Release definition' (spring-music-devops-empty). At the bottom, there is an 'Add post-build action' button.

You're now ready to test the build and release! In Jenkins, click Build now from the project options on the right. When everything completes successfully, click on the build details to see an Azure DevOps release summary added as a final step in the build:

Build #3 (Nov 2, 2018 8:51:41 PM)



Build Artifacts

-  [spring-music.jar](#) 60.25 MB [view](#)
-  [spring-music.jar.original](#) 133.77 KB [view](#)



No changes.

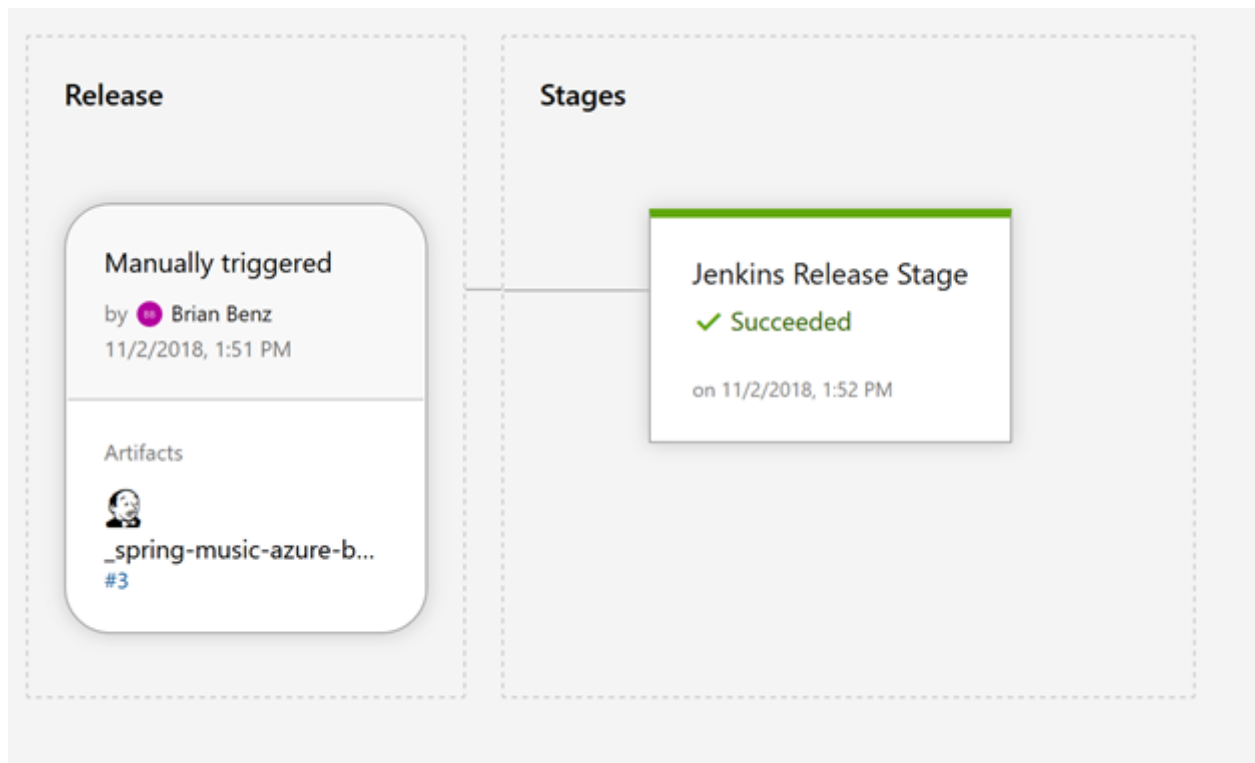
Started by user [Brian Benz](#)

Revision: 890bfa2a69fefe5e73547ee78b5fbd06bf796ec7

• refs/remotes/origin/master

Release summary: https://bbenzms.visualstudio.com/ecb9e1ed-b823-4105-a348-f670aa16e3fa/_release?releaseId=1&a=release-summary

Clicking on the Release summary link will take you to a summary of the actions performed by Azure DevOps to complete the release:



That's it! To summarize, Azure DevOps is used to securely host the project code, then Jenkins is used to build and test the code using gradle. The Jenkins build produces artifacts in Jenkins that are used to trigger a pipeline release in Azure DevOps services.

Next Steps

That's Azure DevOps Services and Jenkins living in harmony! For next steps, you can trigger the build process when code changes in the repo, and the release pipeline to deploy to an Azure service, or anywhere else for that matter.

Here are a few Jenkins plugins you may want to explore. Note that all our Jenkins plugins are open source and available on GitHub. To see all the latest list, visit the Jenkins Plugin site and search for Azure.

- Azure AD plugin allows the Jenkins server to support SSO for users based on Azure AD.
- Azure VM Agents plugin uses an Azure Resource Manager template to create Jenkins agents in Azure virtual machines.
- Azure Storage plugin uploads build artifacts to, or downloads build dependencies from, Azure Blob storage.
- Azure Container Agents helps you to run a container as an agent in Jenkins.
- Kubernetes Continuous Deploy deploys resource configurations to a Kubernetes cluster.
- Azure Container Service deploys configurations to Azure Container Service with Kubernetes, DC/OS with Marathon, or Docker Swarm.
- Azure Functions deploys your project to Azure Function.
- Azure App Service deploys to Azure App Service.

If you have questions or issues, please let me know in the comments, we always value feedback!

Thanks,

-BB

[Jenkins](#) [Azure](#) [DevOps](#) [Java](#) [Spring Boot](#)

[About](#) [Help](#) [Legal](#)