BIZTALK 360

BizTalk Server

CI/CD from zero to hero

Authors

Sandro Pereira
Azure MVP & MCTS BizTalk Server 2010

Pedro Almeida
Senior Integration Consultant at DevScope

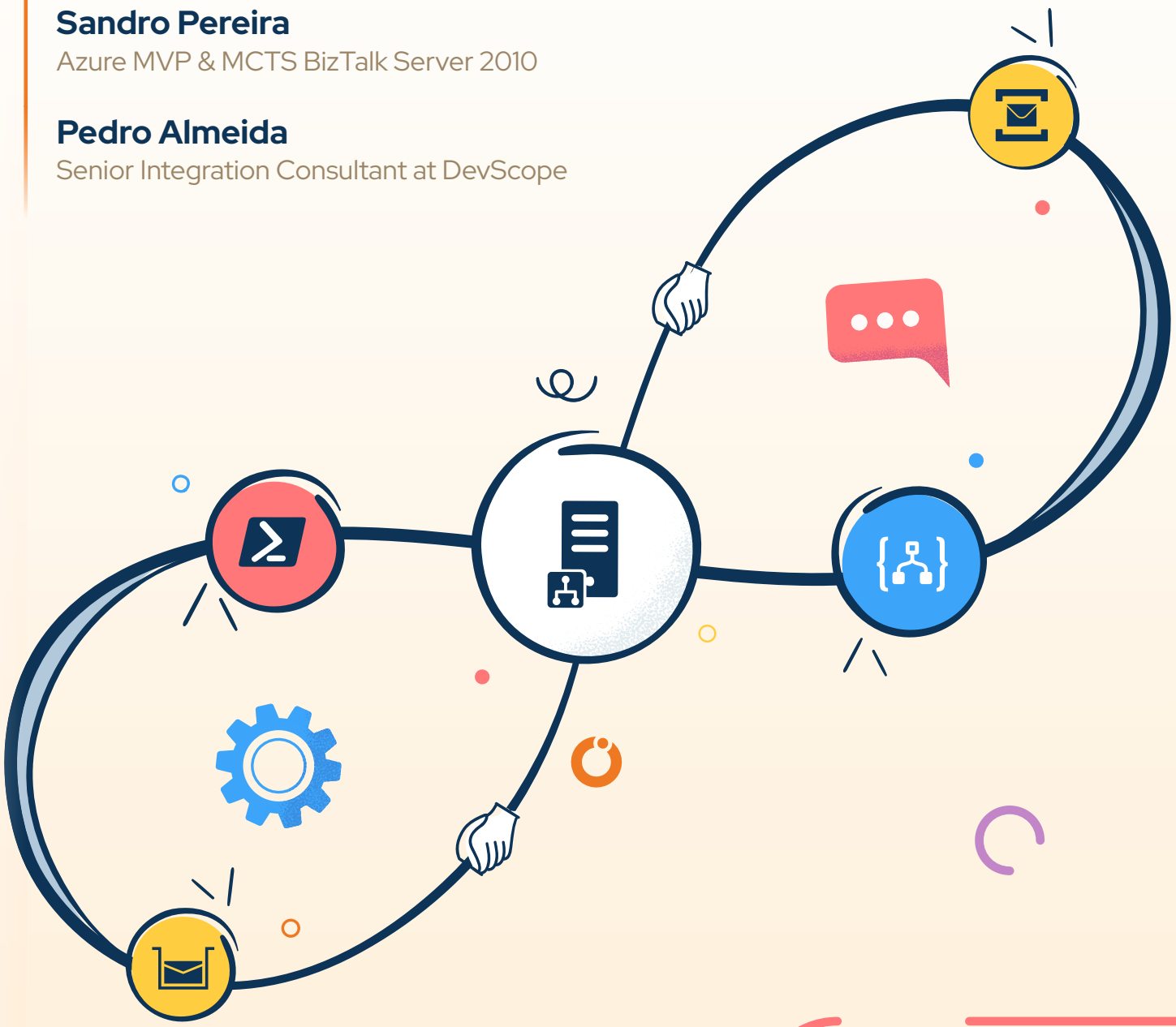


Table of Contents

About the Authors

Written by

Sandro Pereira

[Azure MVP & MCTS BizTalk Server 2010]

Sandro Pereira lives in Portugal and is currently working as an Integrator consultant at DevScope (http://www.devscope.net). In the past years, he has been working on implementing Integration scenarios both on-premises and cloud for various clients, each with different scenarios from a technical point of view, size, and criticality, using Microsoft Azure (API Management, Logic Apps, Service Bus, Event Hubs, PowerApps, Power Automate, ...), Microsoft BizTalk Server and different technologies like AS2, EDI, RosettaNet, SAP, TIBCO and so on

Sandro is very active in the BizTalk community as blogger (https://blog.sandropereira.com), member and moderator on the MSDN BizTalk Server Forums, TechNet Wiki author, Code Gallery and GitHub contributor, member of several online communities, guest author at BizTalk360 and Serverless360, public speaker and technical reviewer of several BizTalk and Azure books and whitepapers, all focused on Integration. He is also the author of the book BizTalk Mapping Patterns & Best Practices.

He has been awarded the Microsoft Most Valuable Professional (MVP) since January 2011, for his contributions to the world-wide BizTalk Server community (https://mvp.microsoft.com/en-us/PublicProfile/4030655). He currently holds MCTS: BizTalk Server 2006 and BizTalk Server 2010 certifications.

You can contact Sandro at **sandro-pereira@live.com.pt**

Twitter: **@sandro_asp**

Written by

Pedro Almeida

Senior Integration Consultant at DevScope

Pedro Almeida is a Senior Integration Developer at Devscope, working with Logic Apps, BizTalk and other related products. Although he started his career as a Dynamics CRM Consultant, Integrations quickly caught his eyes and has made it his primary area of interest and work. Since then, Pedro has worked with customers from very different areas, from Retail to Banking to Governmental Services and others.

You can contact Pedro at mailto: **pedro.miguel_almeida@outlook.com**

Twitter: **@ItsNotRcktScnce**

Introduction

Continuous integration (CI) and continuous deployment (CD) is a development practice that has become an essential aspect of BizTalk development. It allows organizations to automate the deployment of BizTalk solutions through multiple environments without error-prone manual steps, detect issues earlier and ultimately deliver value to end-users faster.

What is Continuous Integration (CI) and Continuous Deployment (CD)

Continuous integration (CI) is a software development strategy that increases development speed while ensuring the code quality that the teams deploy. Developers continuously commit code in small increments (at least daily or even several times a day), which is then automatically built and tested before it is merged with the shared repository.

Continuous Integration automates the building and testing of your software. Continuous deployment is an extension of this automation and allows your software to be deployed after every code commit that passes your test suite.

What are CI/CD Pipelines?

A CI/CD pipeline introduces monitoring and automation to improve the application development process, particularly at the integration and testing phases and during delivery and deployment. Although it is possible to execute each of the CI/CD pipeline steps manually, the true value of CI/CD pipelines is achieved through automation.

Continuous integration/continuous delivery (CI/CD) pipelines are a practice focused on improving software delivery using either a DevOps or site reliability engineering (SRE) approach.

A CI/CD pipeline may sound like overhead, but it isn't. It is essentially a runnable specification of the steps that any developer needs to perform to deliver a new software product version. In the absence of an automated pipeline, engineers would still need to perform these steps manually and, therefore, be far less productive.

Most software releases go through a couple of typical stages:

- **Source** stage: A source code repository triggers a pipeline run in most cases. A change in code triggers a notification to the CI/CD tool, which runs the corresponding pipeline.
- **Build** stage: The stage where the application is compiled, combining the source code and its dependencies to build a runnable instance of our product

that we can potentially ship to our end users. Programs written in languages such as C#, or C/C++ need to be compiled, whereas Python or JavaScript pro grams work without this step.

• **Test** stage: The stage where code is tested, running automated tests to vali date our code correctness and the behavior of our product. Automation here can save both time and effort. The test stage acts as a safety net that prevents easily reproducible bugs from reaching the end-users. The responsibility of writing tests falls on the developers.

• **Deploy** stages: Once we have built a runnable instance of our code that has passed all predefined tests, we're ready to deploy it. There are usually multiple deploy environments, for example, a "DEV", "QA" or "STAGING" environment, which is used internally by the product team, and a "PRODUCTION" environment for end-users.

What is Azure DevOps?

Azure DevOps provides developer services for allowing teams to plan work, collaborate on code development, and build and deploy applications. Azure DevOps supports a collaborative culture and set of processes that bring together developers, project managers and contributors to develop software. It allows organizations to create and improve products at a faster pace than they can with traditional software development approaches.

Azure DevOps provides integrated features that you can access through your web browser or IDE client. You can use one or more of the following standalone services based on your business needs:

• **Azure Repos** provides Git repositories or Team Foundation Version Control (TFVC) for source control of your code. For more information about Azure Repos, see What is Azure Repos?.

• **Azure Pipelines** provides build and release services to support continuous integration and delivery of your applications. For more information about Azure Pipelines, see What is Azure Pipelines?.

• **Azure Boards** delivers a suite of Agile tools to support planning and tracking work, code defects, and issues using Kanban and Scrum methods. For more information about Azure Boards, see What is Azure Boards?.

• **Azure Test Plans** provides several tools to test your apps, including manual / exploratory testing and continuous testing. For more information about Azure

Test Plans, see Overview of Azure Test Plans.

• Azure Artifacts allows teams to share Maven, npm, NuGet, and more packages from public and private sources and integrate package sharing into your pipelines. For more information about Azure Artifacts, see Overview of Azure Artifacts.

BizTalk Server: Automation Deployment with Azure DevOps

Historically, deploying BizTalk Server solutions across environments is or can be a complicated process depending on the complexity ofyour solution. There are many ways to deploy BizTalk artifacts for example:

• Importing them as part of an application by using the Deployment Wizard (from an .msi file), importing them using BTSTask.exe – this is the default way to deploy across environments.

o You can replace and use allow BTSTask, PowerShell scripts.

• Or deploy them from Visual Studio – this is the default way to deploy to your development environment.

Through out the years, the BizTalk Server Community created an open-source deployment framework called Deployment Framework for BizTalk (BTDF) – https://github.com/BTDF/DeploymentFramework. The Deployment Framework for BizTalk is an easy-to-use toolkit for deploying and configuring your BizTalk solutions. In reality, BTDF is an MSBuild project with custom MSBuild tasks, and it can be customized according to customer BizTalk project needs It is also extensible: this framework brings new capabilities and advantages to deploying BizTalk Server solutions, but it also has limitations or disadvantages.

Microsoft has introduced automated deployment of BizTalk Applications in BizTalk Server 2016 Feature Packs using Azure DevOps (previously called Visual Studio Team Services – VSTS). In BizTalk Server 2016 Feature Pack 1, automatic deployment and application lifecycle management (ALM) experience was introduced. The automatic deployment process has been improved with the release of BizTalk Server 2016 Feature Pack 2. These features were only available on the Enterprise edition of BizTalk Server 2016.

BizTalk Server 2020 brings all these functionalities out-of-the-box across all editions: Enterprise, Standard, Development, or Branch.

To accomplish this, we need basically 3 main steps:

- BizTalk Server: Add a BizTalk Server Application project in your Visual Studio solution.
- DevOps: Create a build agent.
- DevOps: Create a Build and release Azure Pipeline.

This article addresses and explains how you can implement CI/CD oriented to BizTalk Server 2020 using Azure DevOps Pipelines.

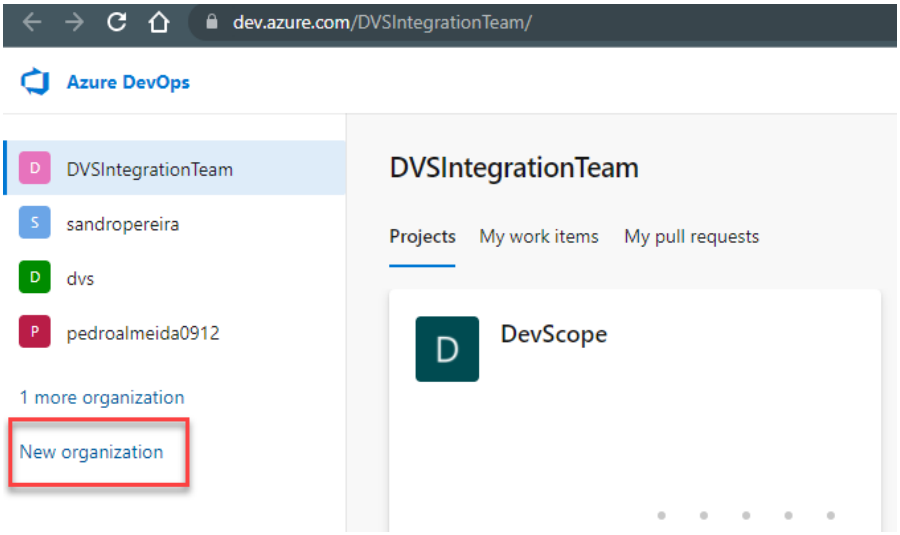
We start by configuring a small BizTalk Server 2020 project using Visual Studio 2019.

Create an organization or project collection

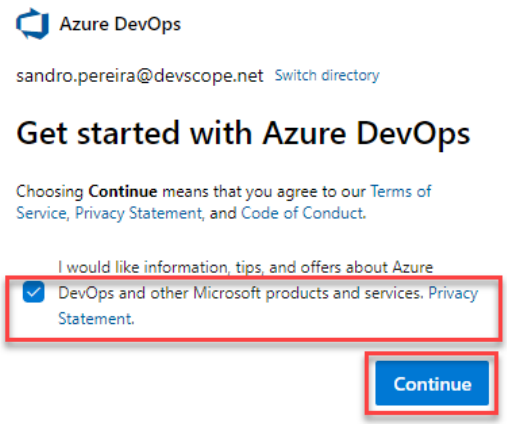
This chapter describes how to use Azure DevOps to create an Organization. An organization is used to connect groups of related projects, helping to scale up an enterprise. You can use a personal Microsoft account, GitHub account or a Microsoft work or school account. Use your work or school account to automatically connect your organization to your Azure Active Directory (Azure AD).

To create an Organization you need to:

- Sign in to Azure DevOps (https://dev.azure.com/).
- Select **New organization.**



- On the **Get started with Azure DevOps** window, check the privacy Statement and select **Continue**.



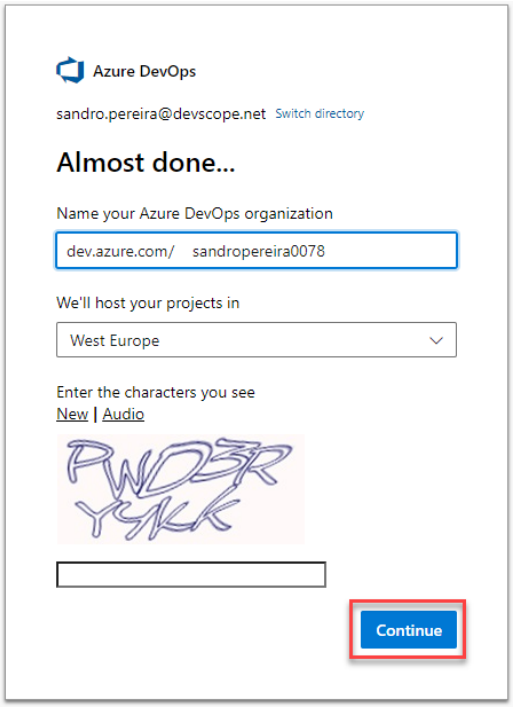
• On the **Almost done**... screen provide the following settings:

o Name: provide a name to your Azure DevOps Organization

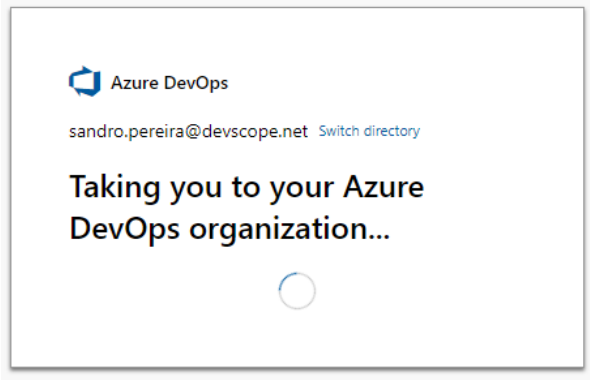
o Region: provide a region to where the projects of that organization will be hosted

o and then select **Continue.**

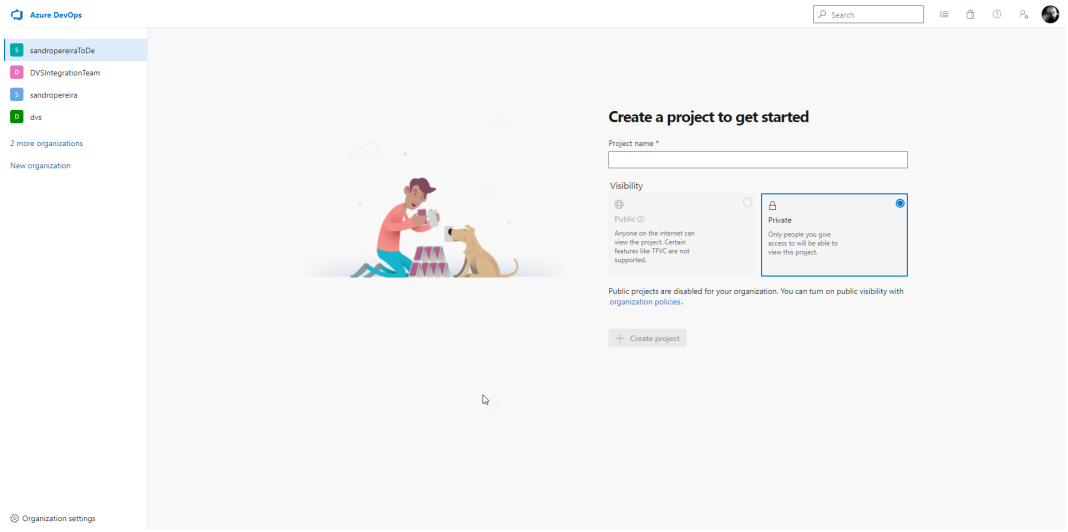
Note: that you need to response to a CAPTCHA challenge before continue.



• The organization process creation will take just a few seconds



• After the creation process finished you will be redirected to the Azure DevOps organization page where you can start creating your projects.



Congratulations, you're now an organization owner! You can always sign in to your organization at any time by using this URL template:

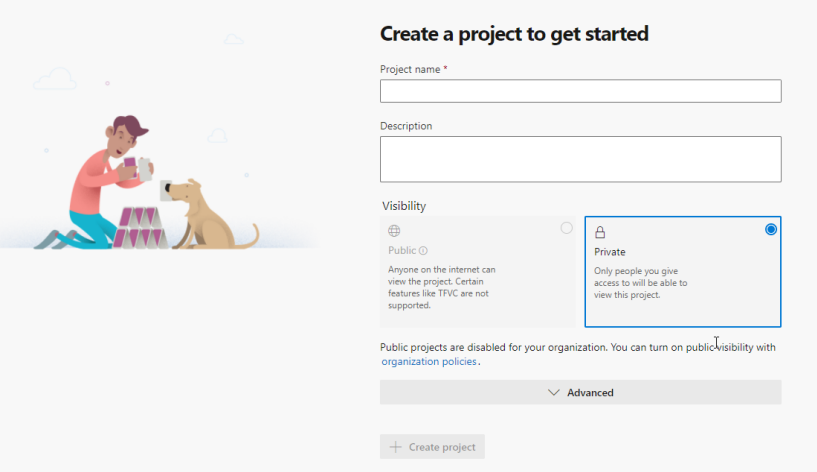
• https://dev.azure.com/{your-organization-name}.

Create a project in Azure DevOps

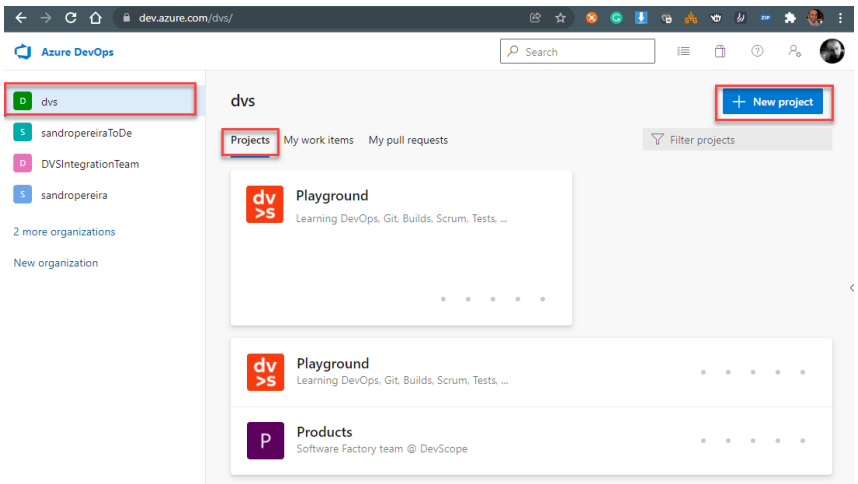
This chapter describes how to use Azure DevOps to create a project and establish a repository for source code. You can manage and structure each project to support your business needs. Each project you create provides boundaries to isolate data from other projects

To create a project you need to:

- Sign in to Azure DevOps Organization (https://dev.azure.com/{your-organization-name}).
- If it's the first project in that organization, on the main page you will see a form asking for the project name, description and visibility in order to create the project



- Alternatively, if your organization already contains one or more projects, select the Projects page, and then select **New project.**



• On the Create new project window, provide the following information:

o **Project name:** Provide a name for your project. Your project name must have 64 or fewer characters but it can not:

• Contain special characters, such as / : \ ~ & % ; @ ' " ? < > | # $ * } { , + = [ ]

• Begin with an underscore.

• Begin or end with a period.

o **Description:** provide a description to your project. This is an optional field.

o **Visibility:** Choose the visibility to your project:

• **Public:** Anyone on the internet can view the project. In this visibility type certain features like TFVC are not supported.

• If the **Public** option is not available, you need to change the policy.

• **Private:** Only people you give access to will be able to view this project.

o In the **Advanced** features you can setup the:

• **Version control:** initial source control type:

• **Git** is a distributed version control system. Each developer has a copy of the source repository on their dev machine. Developers can commit each set of changes on their dev machine and perform version control operations such as history and compare without a network connection. Branches are lightweight.
When you need to switch contexts, you can create a private local branch. You can quickly switch from one branch to another to pivot among different variations of your codebase. Later, you can merge, publish, or dispose of the branch.

• **Team Foundation Version Control** (TFVC) is a centralized version control system. Typically, team members have only one version of each file on their dev machines. Historical data is maintained only on the Server. Branches are path-based and created on the server.

• **Work item process:** Choose the process that provides the best fit for your team:

•**Basic:** Choose Basic when your team wants the simplest model that uses Issues, Tasks and Epics to track work.

- Tasks support tracking Remaining Work.

• **Agile:** Choose Agile when your team uses Agile planning methods, including Scrum, and tracks development and test activities separately. This process works great if you want to track user stories and (optionally) bugs on the Kanban board, or track bugs and tasks on the taskboard.

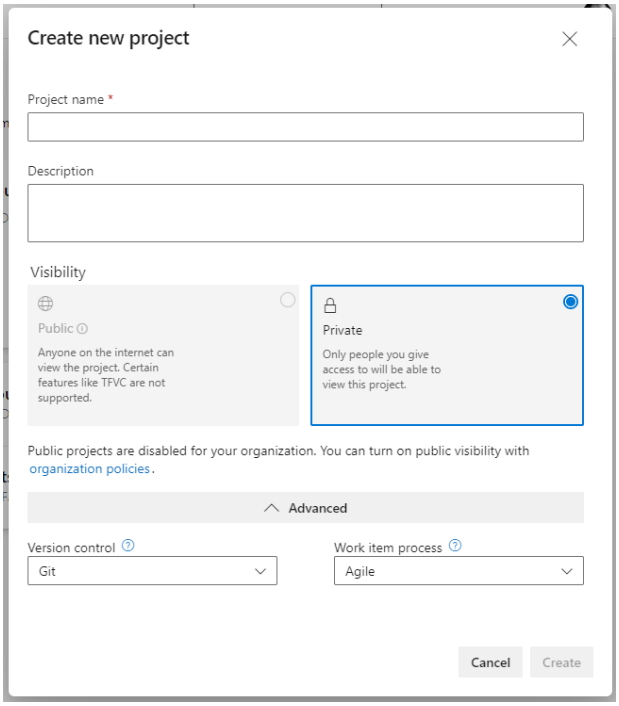
- Tasks support tracking Original Estimate, Remaining Work and Completed Work.

• **Scrum:** Choose Scrum when your team practices Scrum. This process works great if you want to track product backlog items (PBIs) and bugs on the Kanban board or break down PBIs and bugs into tasks on the taskboard.

- Tasks support tracking remaining work only.

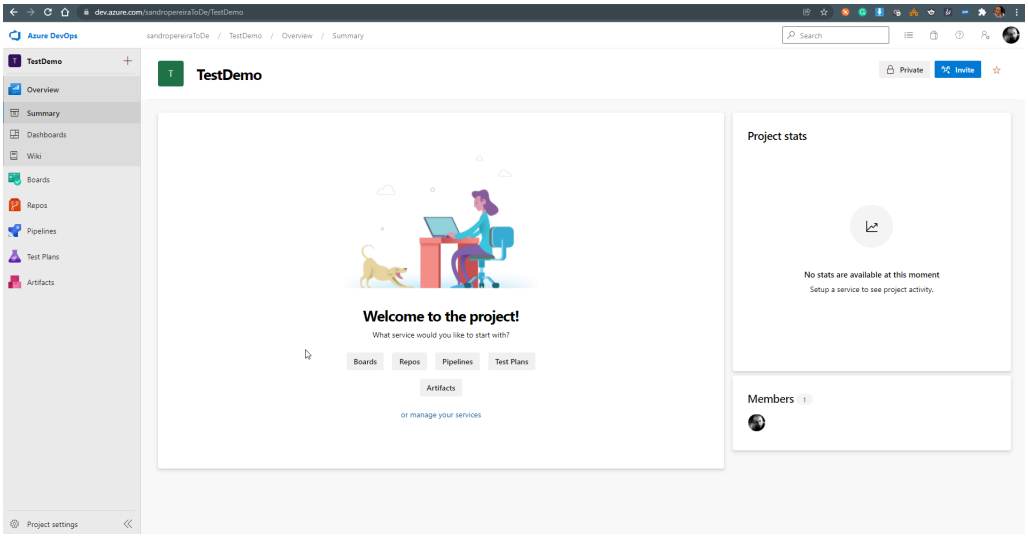
• **CMMI:** Choose CMMI (Capability Maturity Model Integration) when your team follows more formal project methods that require a frame work for process improvement and an auditable record of decisions. With this process, you can track requirements, change requests, risks and reviews.

- This process supports formal change management activities. Tasks support tracking Original Estimate, Remaining Work and Completed Work.



o Select **Create.** The welcome page appears.

• The project will take a few seconds to be created. After the creation process finished you will be redirected to the project page.



After that, you can select one of the following options to continue:

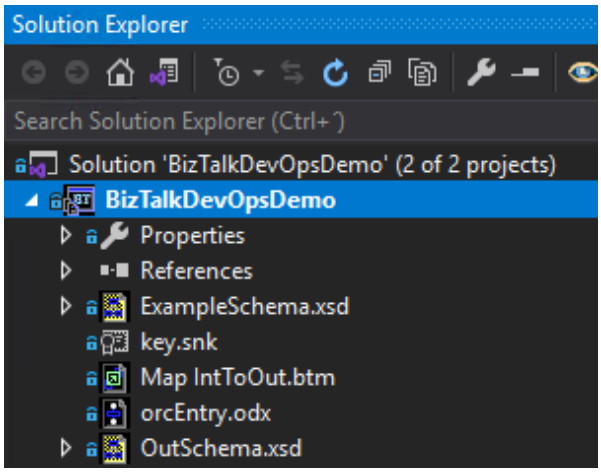
- **Invite:** add others users to your project or invite users who are already in your organization. For more information, see Add users to a project.
    - o https://docs.microsoft.com/en-us/azure/devops/organizations/security/add-users-team-project?view=azure-devops#add-users-to-a-project
- **Boards:** add work items. See View and add work items using the Work Items page.
    - o https://docs.microsoft.com/en-us/azure/devops/boards/work-items/view-add-work-items?view=azure-devops
- **Repos**: clone or import a repository or initialize a README file for your project summary page. See Clone an existing Git repo.
    - o https://docs.microsoft.com/en-us/azure/devops/repos/git/clone?view=azure-devops
- **Pipelines:** define a pipeline. See Azure Pipelines documentation.
    - o https://docs.microsoft.com/en-us/azure/devops/pipelines/?view=azure-devops
- **Test Plans:** define test plans and test suites. See Create test plans and test suites.
    - o https://docs.microsoft.com/en-us/azure/devops/test/create-a-test-plan?view=azure-devops
- **Artifacts:** discover, install, and publish NuGet, npm and Maven packages. See the Azure Artifacts overview.
    - o https://docs.microsoft.com/en-us/azure/devops/artifacts/start-using-azure-artifacts?view=azure-devops
- **Manage your services:** disable the visibility of services. See Turn a service on or off.
    - o https://docs.microsoft.com/en-us/azure/devops/organizations/settings/set-services?view=azure-devops

Preparing your Visual Studio BizTalk Server project for CI/CD

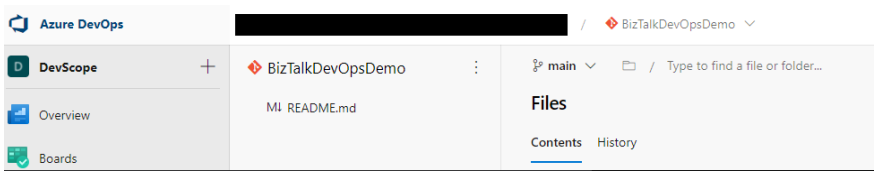
The standard BizTalk Deployment task does a decent job in deploying the application, but it doesn't handle changing tokens or registering DLLs in the Global Assembly Cache(GAC).

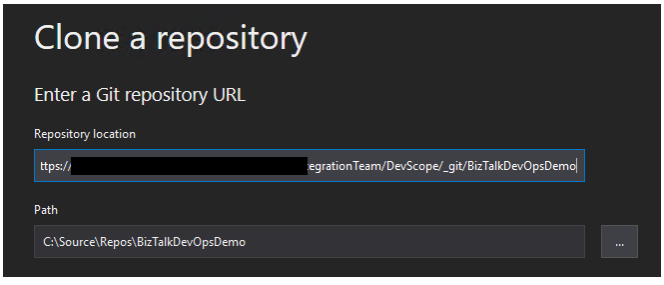
To deploy in multiple machines or to change your Bindings according to your environment, you have to make your file dynamic. This means, replacing your connections with variables or having multiple binding files – a binding file for each environment that can be difficult to manage, sometimes we end having a huge XML file for a specific application.

Let's start with the basics. Notice that we will start from an existing BizTalk Server project – a very basic one – that contains two Schemas, one map and one orchestration.



As always, it is better to first create the DevOps repository and clone it in your machine.





or add your project to a DevOps existing repo.

Having the project created, you need to make sure your project is working properly and then have a Deployment Project as well. This will contain the needed DLLs and Binding file pointers to your BTS project. This also contains the Application name to be deployed and some other configurations.

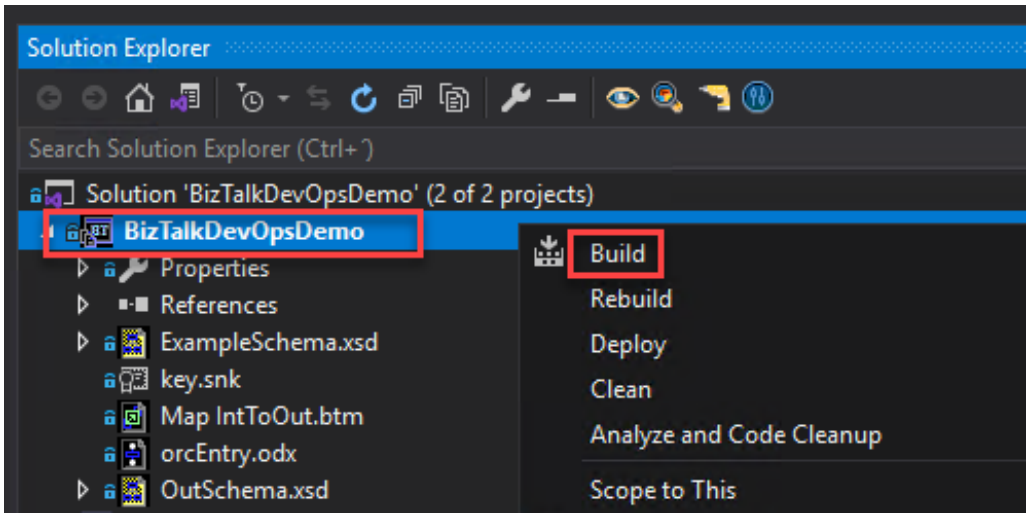
Creating a BizTalk Server Deployment Project

When you build your applications using DevOps, a new BizTalk project file has to be created – .btaproj. This new project holds all the BizTalk applications that you build and deploy using the Azure DevOps build and release features.

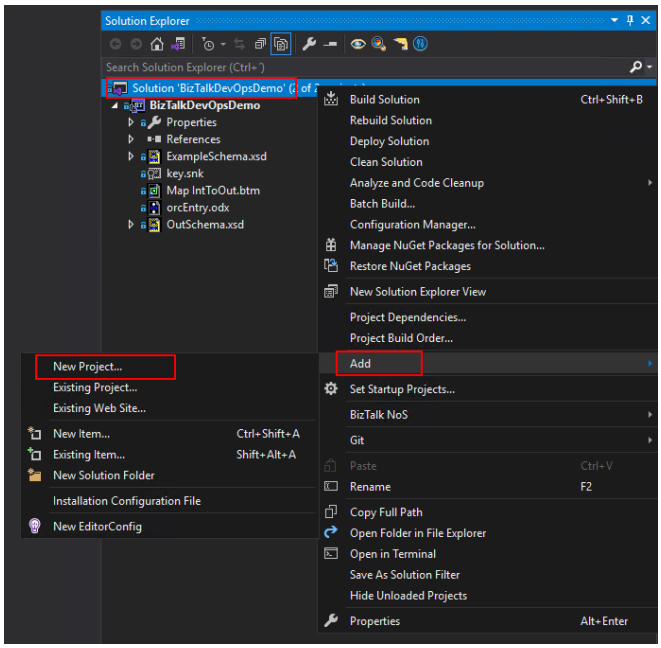
Add the application project

To add a application project we need to:

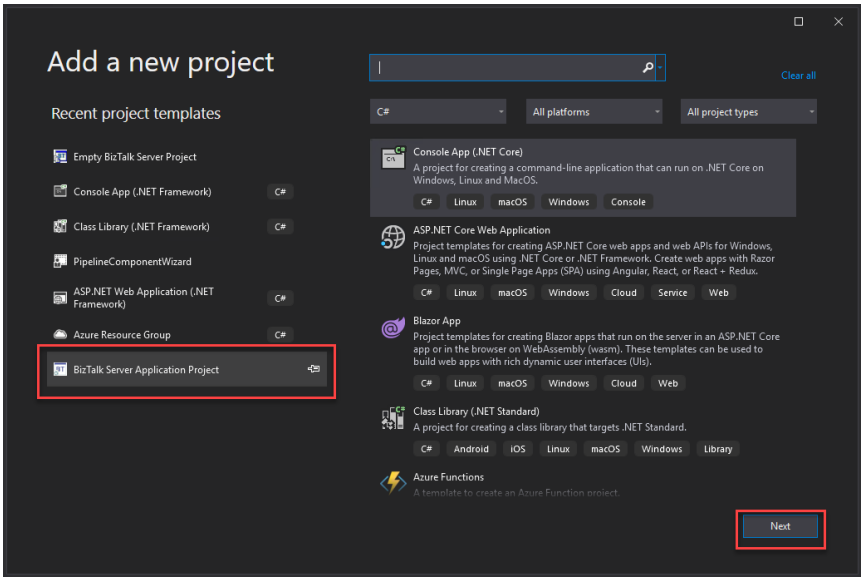
- On the BizTalk Server, open your solution (ProjectName.sln) in Visual Studio.
- In the Solution explorer, right-click your **project > Build**. Be sure your build succeeds. Right-click your **project > Deploy**. Be sure your deploy succeeds



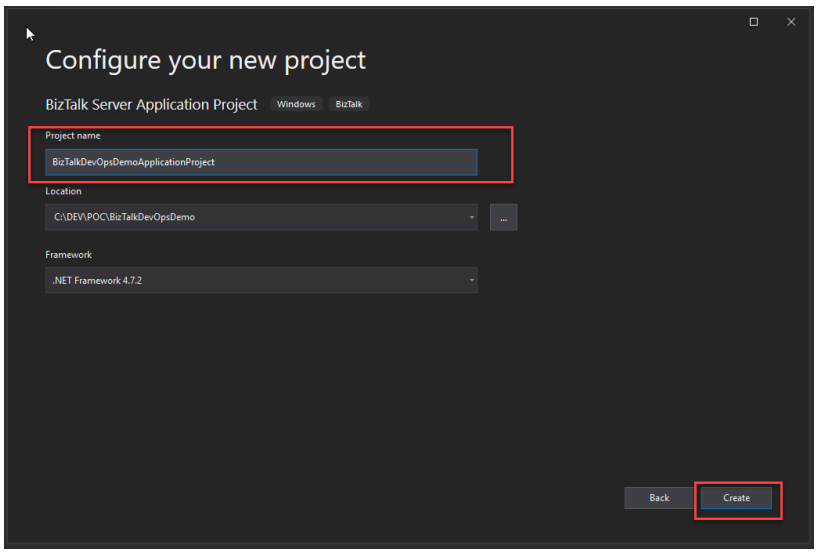
• Right-click your **solution > Add > Add New Project.**



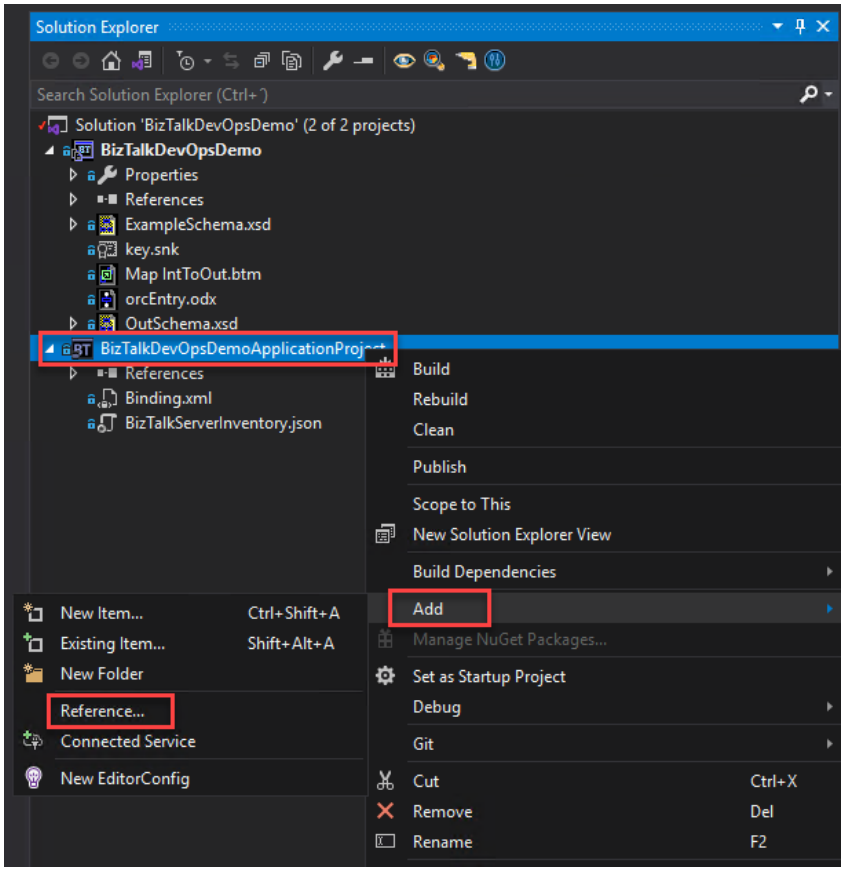
• Select **BizTalk Server Application Project > Next**. Enter a project name, such as BizTalkDevOpsDemoApplicationProject > Create.



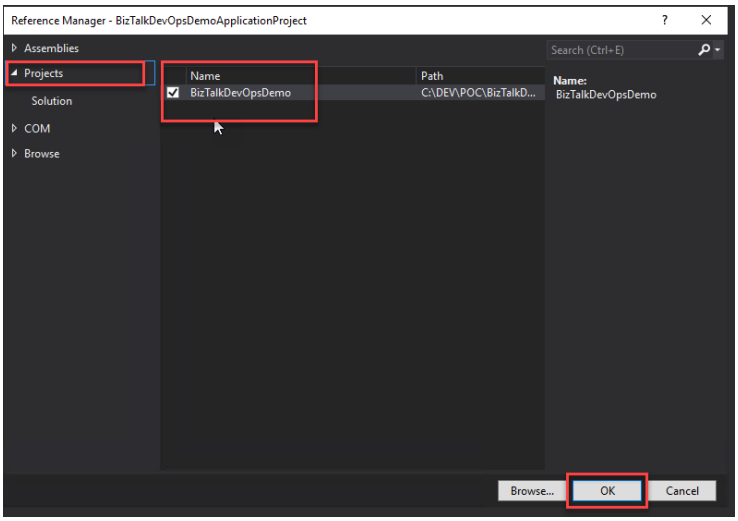
• Add application project in BizTalk Server



• In Solution Explorer, right-click your **newly-added application project (.btaproj) > Add > Reference.**



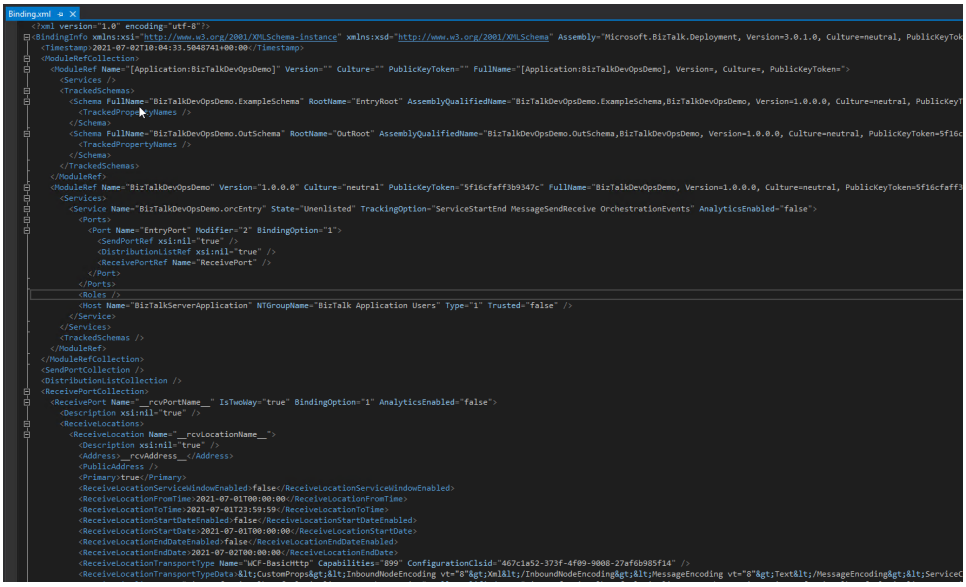
• Expand the Projects tab, and check your BizTalk project (the project you're deploying using Azure DevOps). Select **OK.**



Note: When referencing your BT projects, do make sure that the Application Project is using the same .NET framework version as your other projects. If it's not the same version, it will not be able to copy the DLLs to the referenced Path and will not build successfully.

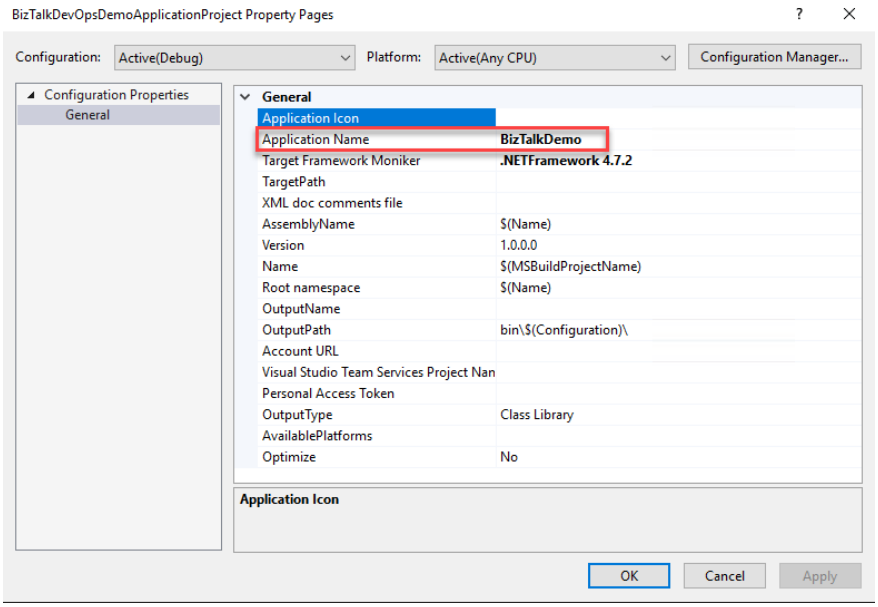
• Once added, expand References under your application project (e.g. BizTalk DevOpsDemoApplicationProject) to see the BizTalk project you just added.

• In Solution Explorer, right-click your **application project (.btaproj) > Add > Existing Item,** and then add your binding file of your application that you can export from BizTalk Server Administration Console.



Note: The bindings file that is created with the project is just an empty template, so you'll want to deploy your application in your Dev Environment and create those bindings. It will make a difference if you export your application bindings when it's started and when it's stopped, so keep that in mind.

• Optional. Right-click your **newly-added application project > Properties**. Customize the Application Name you want to be shown in the BizTalk Adminis tration console:



Making your Bindings dynamic for deployment

Your standard Bindings export will carry the ports and URIs/connections straight from the Admin console. Through a little magic, we will configure these values to be dynamic and it's super easy.

Once you've exported the bindings and you may want to make it ready for DevOps and to accept multiple configurations. To do that we need to tokenize some properties:

• In our sample, we will tokenize the ReceiveLocation and ReceivePort names that are static – this can also be applicable for, for example, the URI of the receive location.

o If we tokenize this, you can call it whatever you want, therefore reducing the risk of colliding with other existing ports in your end systems.

```
<ReceivePortCollection>
  <ReceivePort Name="ReceivePort" IsTwoWay="true" BindingOption="1" AnalyticsEnabled="false">
    <Description xsi:nil="true" />
    <ReceiveLocations>
      <ReceiveLocation Name="Receive Location1">
        <Description xsi:nil="true" />
        <Address>/testservice.net/service.svc</Address>
        <PublicAddress />
        <Primary>true</Primary>
        <ReceiveLocationServiceWindowEnabled>false</ReceiveLocationServiceWindowEnabled>
        <ReceiveLocationFromTime>2021-07-01T00:00:00</ReceiveLocationFromTime>
        <ReceiveLocationToTime>2021-07-01T23:59:59</ReceiveLocationToTime>
        <ReceiveLocationStartDateEnabled>false</ReceiveLocationStartDateEnabled>
        <ReceiveLocationStartDate>2021-07-01T00:00:00</ReceiveLocationStartDate>
        <ReceiveLocationEndDateEnabled>false</ReceiveLocationEndDateEnabled>
```

• So, keeping the desired token in mind, we are going to replace these values, ReceiveLocation address included, with a variable and token identifier. With a few magic touches, we end up with something like this:

```
eceivePortCollection>
<ReceivePort Name="__rcvPortName__" IsTwoWay="true
  <Description xsi:nil="true" />
  <ReceiveLocations>
    <ReceiveLocation Name="__rcvLocationName__">
      <Description xsi:nil="true" />
      <Address>__rcvAddress__</Address>
      <PublicAddress />
```

And that's it. Of course, this is a very small and simple example, but even with a goliath project, it will still be the same pattern. You find what you want to make dynamic, tokenize it, save and upload your changes to your Repo.

Configure the BizTalkServerInventory JSON template

The BizTalk Application Project includes the BizTalkServerInventory.json file. In this file, add your BizTalk assemblies, add the binding files for your BizTalk application, and then set a deployment sequence.

You will see that you can set the BizTalk Assemblies path as well as other Assemblies, Pre and Post processing scripts and the Deployment Sequence. This is one of the most important steps, because, as you know, it does matter in which order you deploy your BizTalk Assemblies.

To configure the JSON template you need to:

- In Visual Studio, in your application project (.btaproj), open the BizTalkServer Inventory.json file.
- The template includes the following sections:
  - BizTalkAssemblies: The assemblies used in your applications.
  - BindingFiles: The binding files you're referencing.
  - Assemblies: External assemblies like C# helper classes.
  - PreProcessingScripts: Pre processing scripts.
  - PostProcessingScripts: Post processing scripts.
  - Files: other files like Business rules.
  - DeploymentSequence: The sequence for the elements to be installed.
- In our case we need to:
  - In BizTalkAssemblies, add the assemblies used by our BizTalk project:

```
  "BizTalkAssemblies": [
    {
      "Name": "BizTalkDevOpsDemo",
      "Path": "bin\\BizTalkDevOpsDemo.dll"
    }
],
```

  - In BindingsFiles, add the binding files for your BizTalk project:

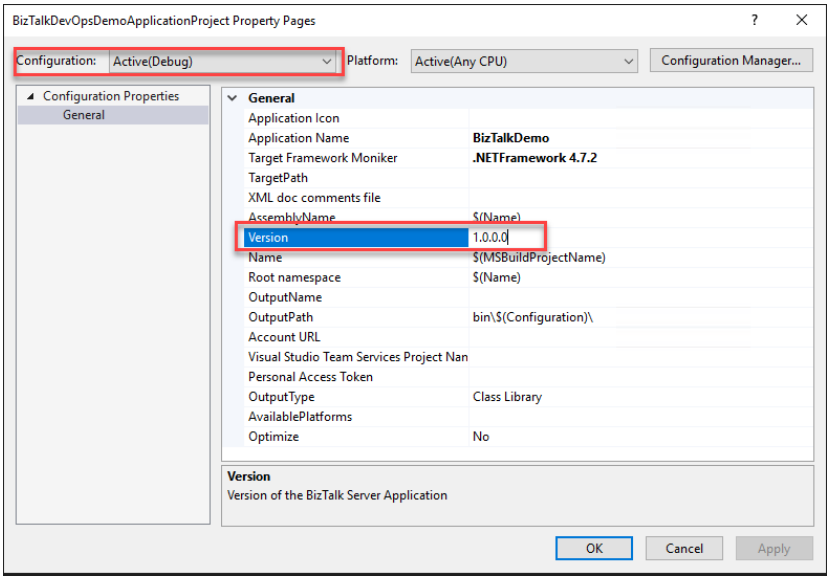
```
  "BindingsFiles": [
    {
      "Name": "BindingFile",
      "Path": "bindings\\Binding.xml"
    }
  ],
```

  - In DeploymentSequence, add the application names in the order you want them to be deployed and installed on the BizTalk Server:

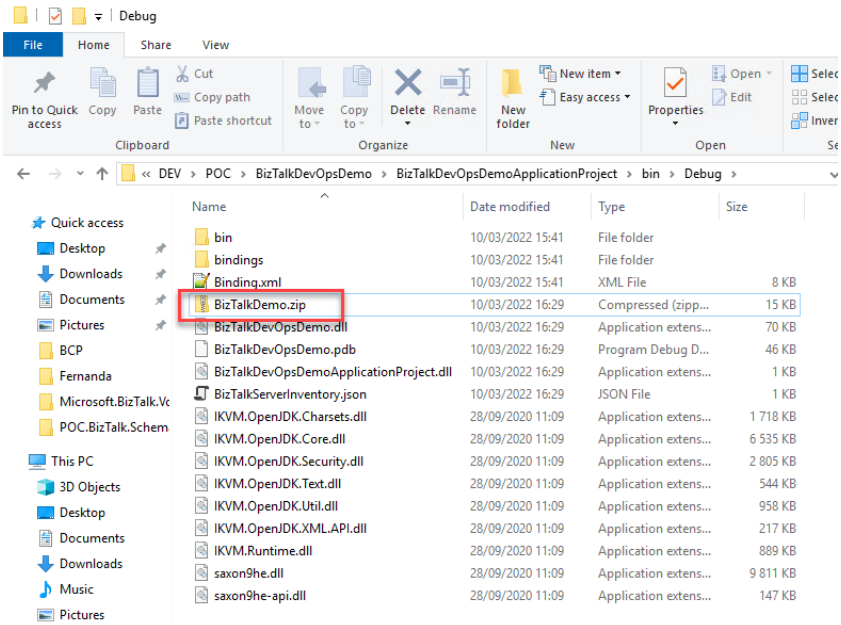
```
"DeploymentSequence": [
    "BizTalkDevOpsDemo",
    "BindingFile"
  ]
```

o Note that the last one is always the BindingFile.

• Optional. Right-click your application project **(e.g. BizTalkDevOpsDemo ApplicationProject) > Properties**. You can set the Debug or Release version to a new value.



• Right-click your **application project (e.g. BizTalkDevOpsDemoApplication Project) > Build**. If it succeeds, a zip file is created in **yourApplicationProject\ bin\debug** folder.



Building this project will generate a ZIP file that contains all that is needed. You can try to publish it directly, after configuring the application.

At this import, it is important to remember that we need to have our solution present in DevOps, otherwise you need to connect it.

Publish your code

Last but not the least, if you didn't already, we need to add our Visual Studio solution to an Azure DevOps repo. You can do this by:

- Connect to an Azure DevOps repo by right-clicking the solution name in Solution Explorer, and then selecting **Create Git Repository...**
- On the **Create a Git repository** window, because we already create ao Repo earlier on this article, select Existing remote and provide the URL on the **Push your code to an existing remote > Remote URL** property. Finally click **Create and Push**.
- On the **Git Changes** window, add a description and select the option **Commit all and Push** to add the solution to Azure DevOps repo.

Create a Personal Access Token

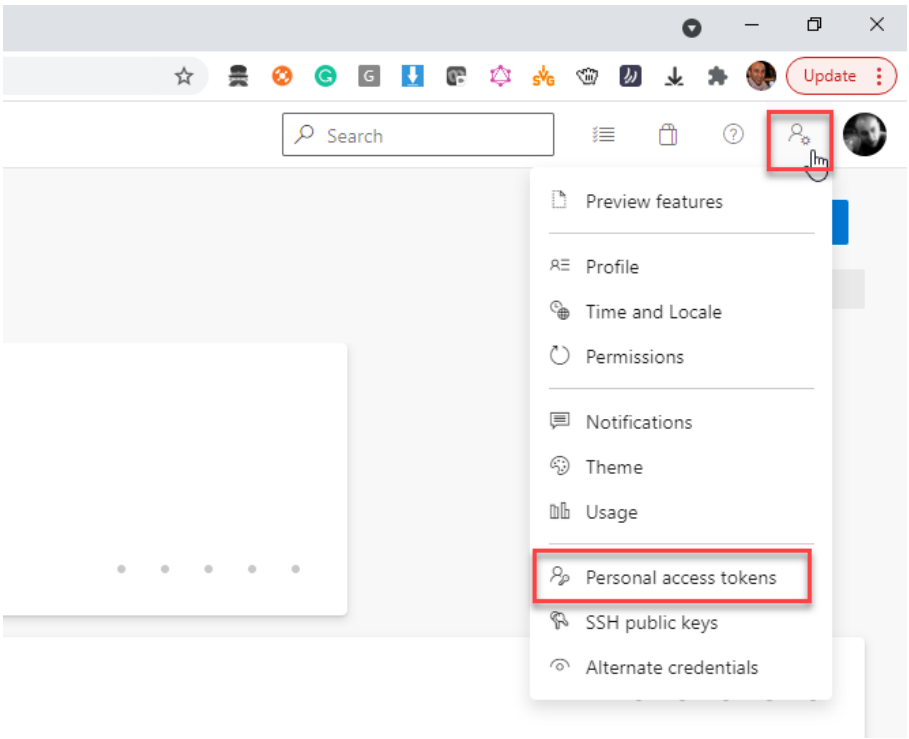
A **personal access token (PAT)** is created in DevOps. This token is your password and is used by the DevOps build agent to authenticate. The token is only shown when you create it. After that, it isn't shown anymore. Once you create it, you should save it to another file in a rememberable location.

To accomplish that:

• Sign in to Azure DevOps Portal (https://app.vsaex.visualstudio.com/) using your work or school account.

o If you do not have an account, select **Create new account**, and enter a name. To manage your code, choose your personal preference between Git or Team Foundation Version Control. When finished, your new account is created, you will be able to access Azure DevOps Portal.

• Select your DevOps organization and then click the top second right-side corner icon **– User settings –** and select **User settings > Personal access tokens.**



• The Personal Access Tokens page will be presented a list of all existing personal access tokens.

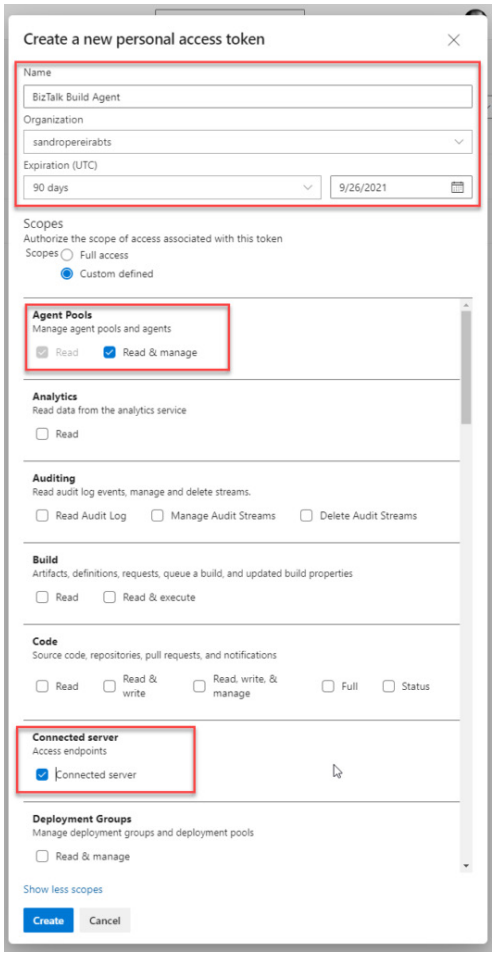
o If you don't have an existing PAT for your agent, select **Add,** and on the **Create a new personal access token** page, enter the following configuration:

• On the **Name** property, enter a name for your PAT, for example, **BizTalk Build Agent.**

• On the **Organization** property, leave the default organization.

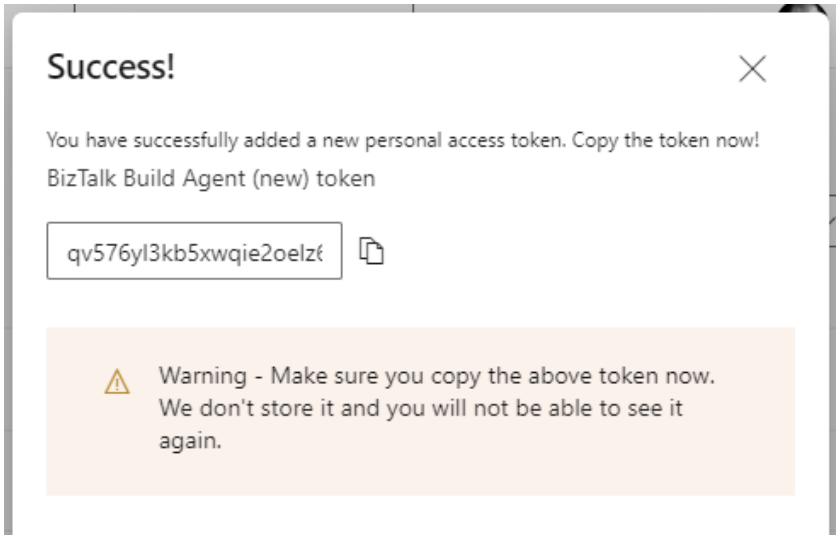
• On the **Expiration (UTC)** property, set an expiration date, for example, **90 days.**

• In **Scopes,** select **Show all scopes,** and then select **Agent Pools – Read & manage** option and **Connected server – Connected server.**



• Select **Create** to finish the PAT creation.

o **Important Note:** You need to save the token value. You will need it in future steps. If you don't know the access token value and didn't take note of it anywhere, it cannot be retrieved. In this case, you need to create a new PAT.



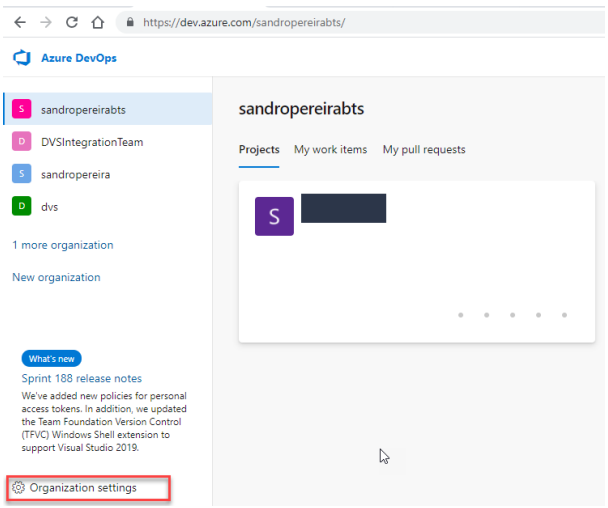
Install the Build Agent

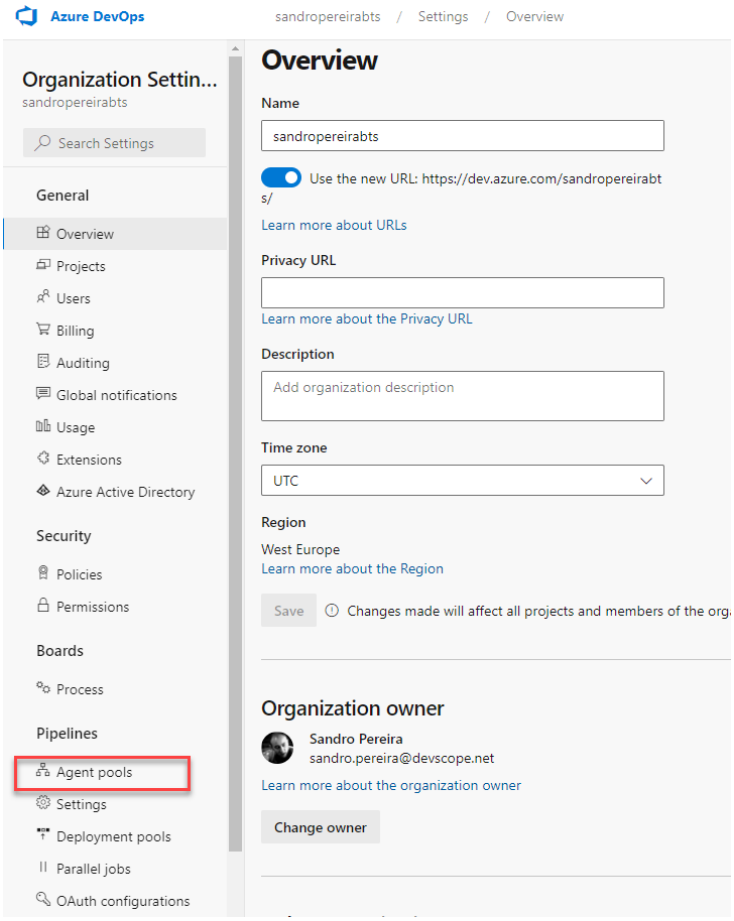
The build agent is installed on the BizTalk development computer. If using deployment groups, the build agent is installed on all the BizTalk servers you want to deploy to. Also, use these same steps to add a build computer, which might be different than the BizTalk development computer.

The following steps show you how to install the build agent on a single computer:

• Open your Azure DevOps organization and then select the **Organization settings** icon and then **Agent Pools.**

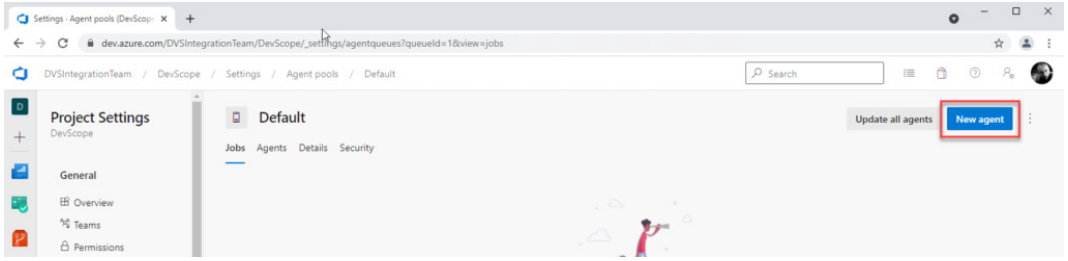
o Optionally, you can choose a Project inside your Organization, and then select the **Project settings** icon and select **Agent Pools.**





• Open Agent pools page, select the **Default (Azure Pipelines)** agent.

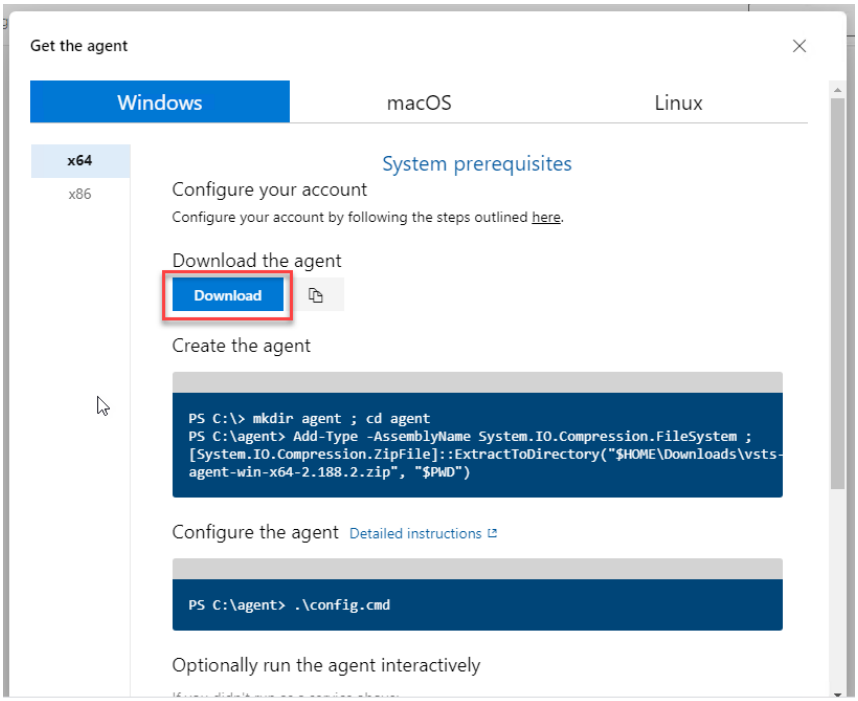
• On the Default agent page, select **New agent.**



On the Get the agent pop-up window, select your OS (Operating System), and on the Download the agent section, select Download.

It is important for you to save the file to your Downloads folder on your BizTalk Server Development machine since the scripts will be referencing that folder.

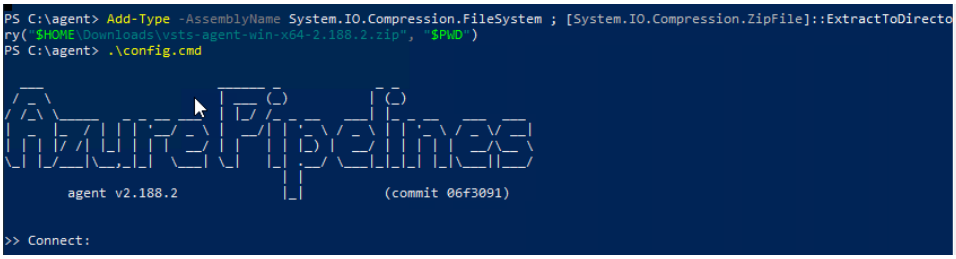
Depending on your OS, this will download a zip file, for example, vsts-agent-win-x64-2.188.3.zip, that you will need to create the agent on the BizTalk Server Development machine.



• The first step is to create the agent on your BizTalk Server Development machine. To do that open Windows PowerShell as Administrator and type the following command:

```
PS C:\> cd /
PS C:\> mkdir agent ; cd agent
PS C:\agent> Add-Type -AssemblyName System.IO.Compression.FileSystem ; [System.IO.Compres-
sion.ZipFile]::ExtractToDirectory("$HOME\Downloads\vsts-agent-win-x64-2.188.3.zip", "$PWD")
```

• **Note:** The vsts-agent file version changes. Make sure the zip file name is the correct one.

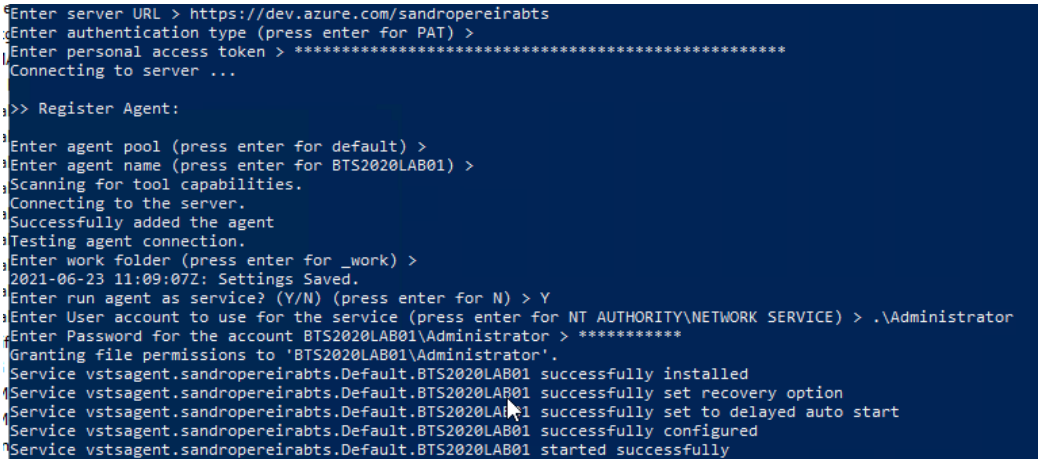


• The second step, as you also see in the picture is to configure the agent. To do that type the following command:

```
PS C:\agent> .\config.cmd
```

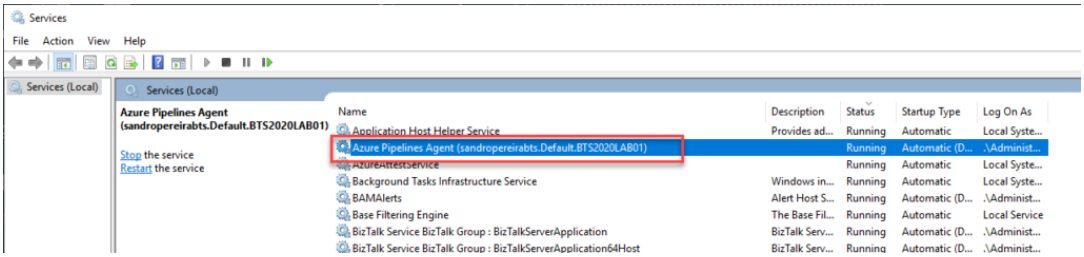
• Enter the following details:

- **Server URL:** Type https://dev.azure.com/{your-organization}.
- **Authentication Type:** Enter **PAT.**
- **Personal access token:** Paste your Azure DevOps token.
- **Agent pool:** Click Enter for assuming the default value.
- **Agent name:** Click Enter for assuming the default value.
  - **Replace:** Only displays if you have an existing agent.
- **Work folder:** Click Enter for assuming the default value.
- **Run agent as a service:** Enter Y.
- **User account:** This value is up to you, but you may run into a permissions issue. Consider entering your current logged-on account, which is a local admin.

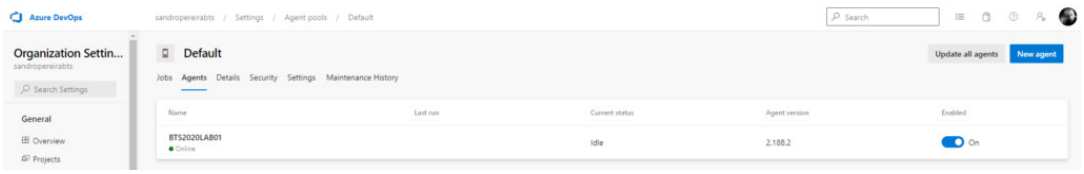


• To validate if the agent was properly installed, Open **services.msc** to see the new service called Azure Pipelines Agent (<organization>.<agent pool>.<server>). The job should be running, otherwise type the following command:

```
PS C:\agent> .\run.cmd
```



Now, if we go back to our DevOps organization > Organization settings > Agent pools > Default (Azure Pipelines) > Agents, you will see your BizTalk Server Development server on the list:



Building your Azure Pipeline

In this chapter, we show you how to create your Azure Pipeline and preparing it for any environment you need to deploy your Logic Apps.

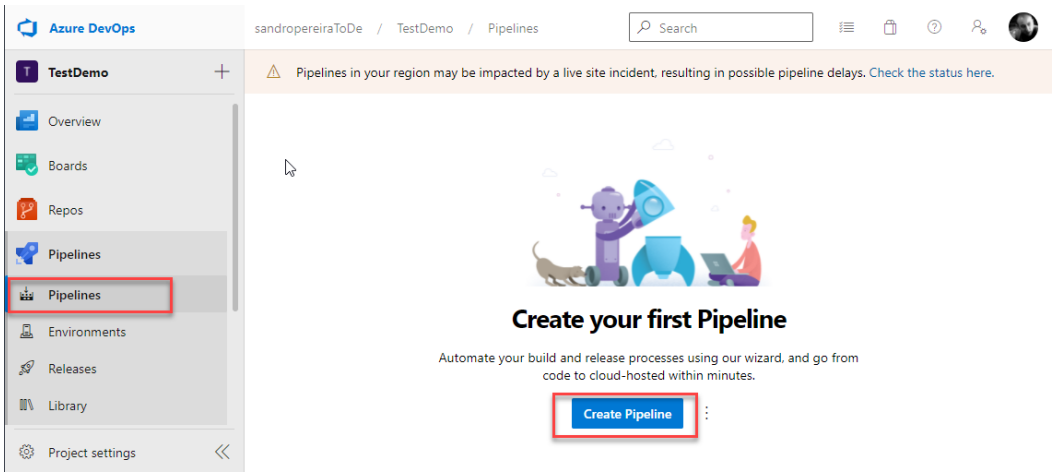
Building the Pipeline

Assuming you already have your repository configured, building the pipeline is fairly simple and quick. We are not a big fans of using YAML, we find it easier to use the classic editor, having the GUI seems more appealing to us, so in this article we will be using the classic editor.

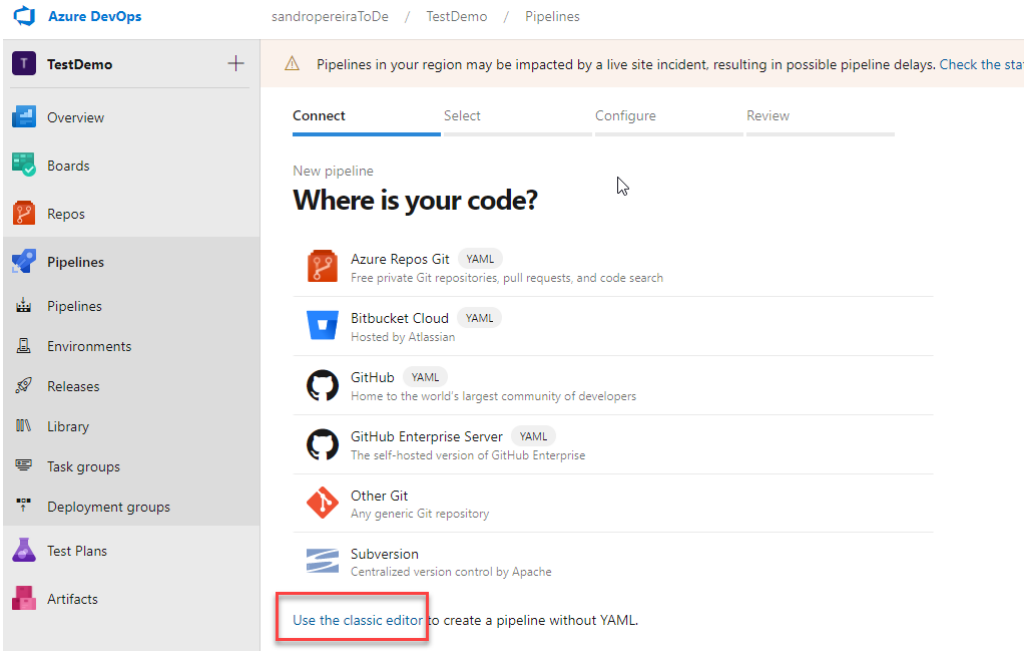
Microsoft Azure Pipelines is a cloud service you can use to automatically build, test and deploy your code project. You can also make it available to other users and it works with just about any language or project type.

In order for us to create a Azure Pipeline we need to:

- In Azure DevOps, go to your project.
- Either from the project page or from the left pane, select Pipelines.
- Select Create Pipeline.



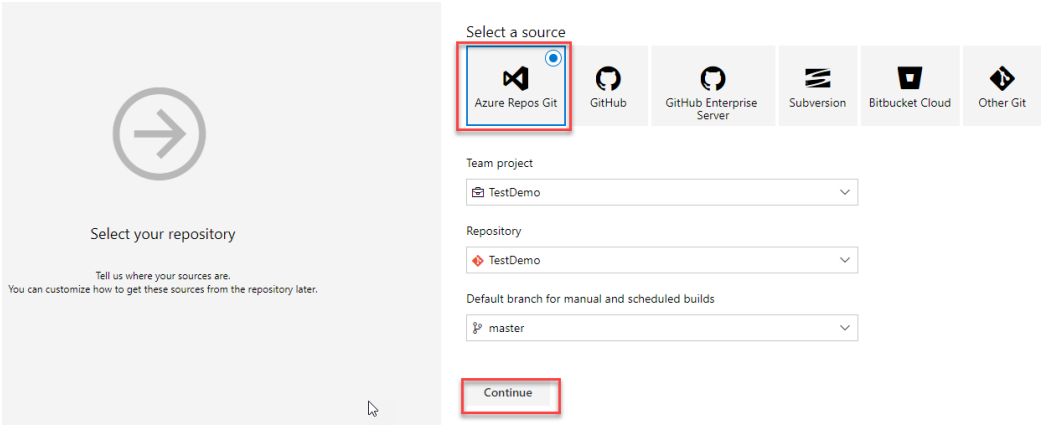
On the Connect tab, select Use the classic editor.



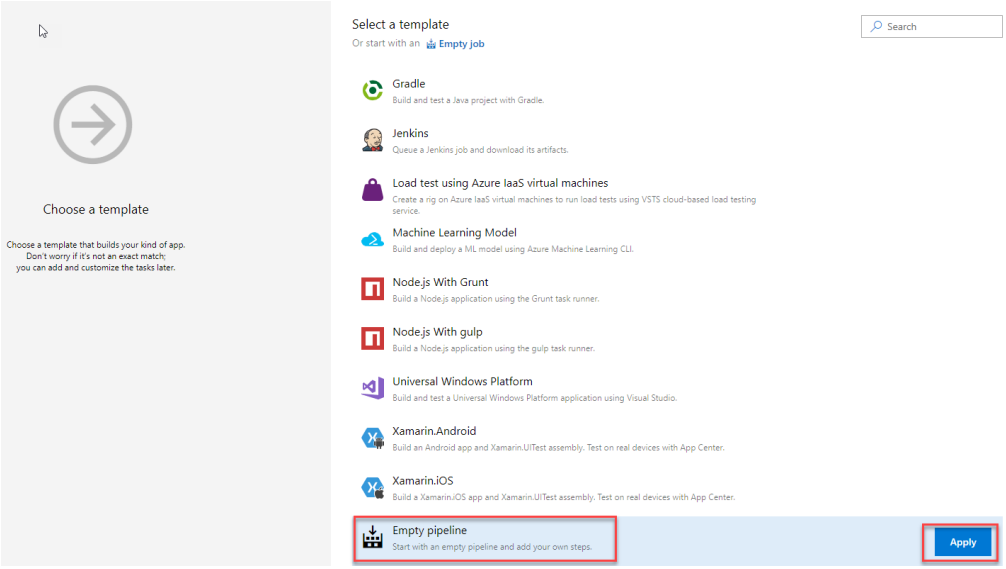
• On the **Select your repository** page, provide the following information and click **Continue**:

o On the **Select a source** property choose **Azure Repos Git.**

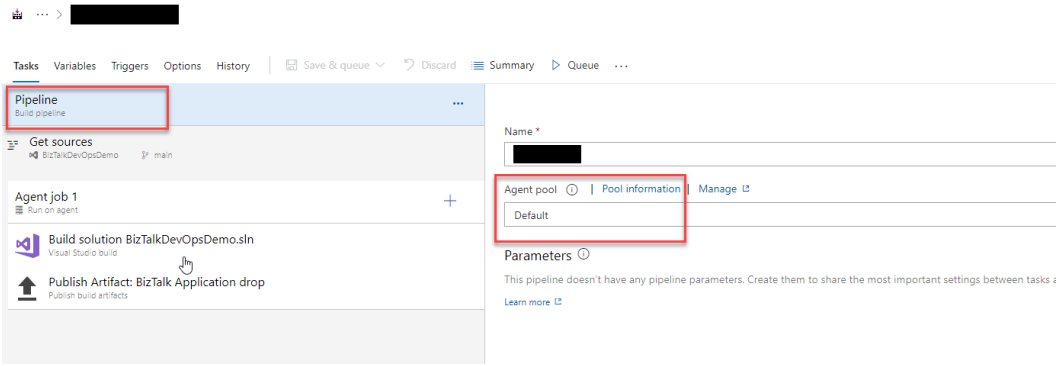
o On the **Team project** and **Repository** properties, select your current project and repo.



• On **Choose a template** page, scroll down to the end **Select a template** section and select **Empty pipeline** and then click **Apply.**

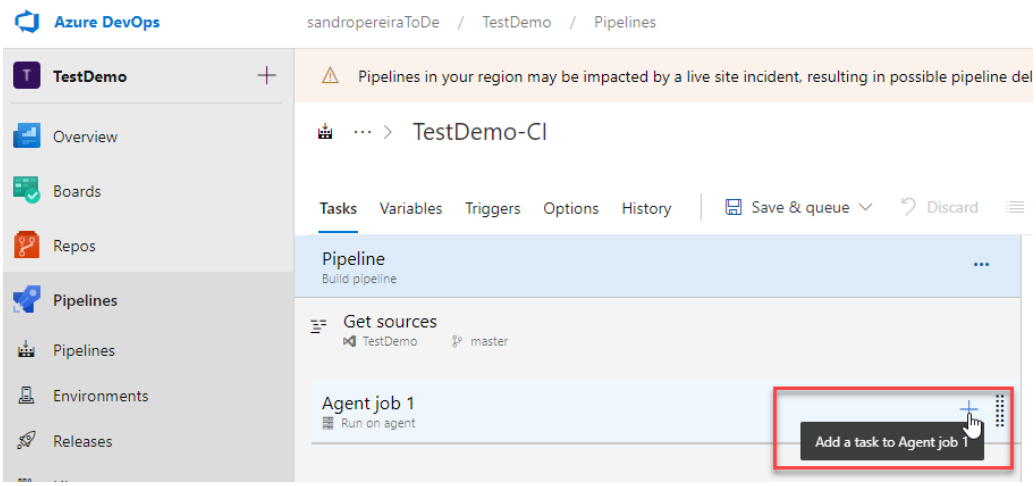


• Select the **Pipeline** and make sure that the Agent pool is configured as **Default.**

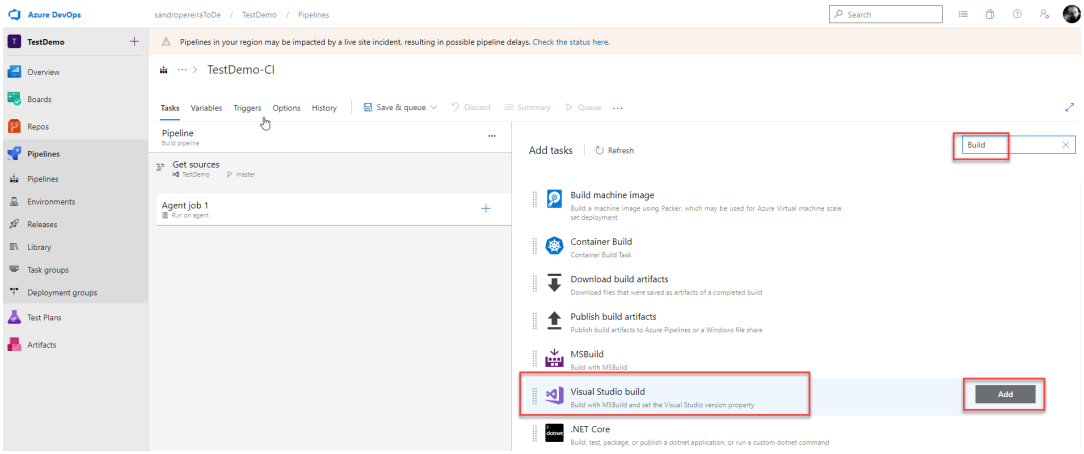


• We are now going to add the tasks to our pipeline, so on the **Tasks** tab

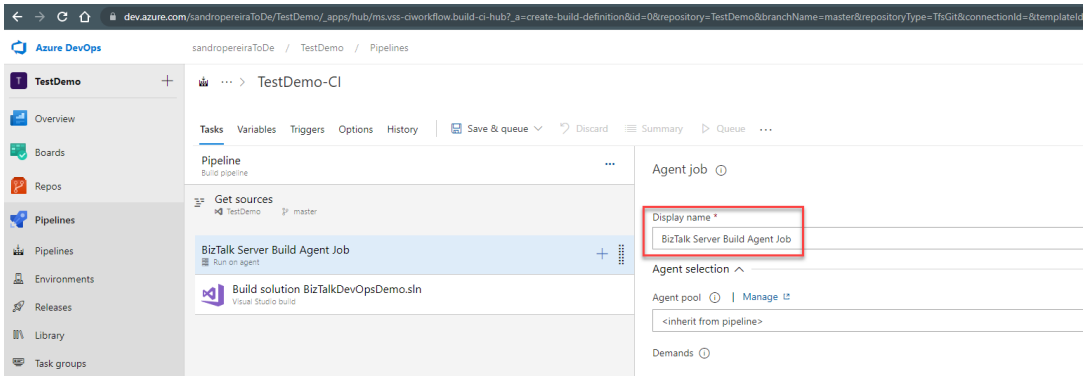
o On the **Agent job 1,** click on **Add a task to a Agent job 1**



o On the **Add tasks** panel, search for **Build** and select **Visual Studio build** task and click **Add.**



o Before going further, with the configuration of our Build task, let's get a step back and select the Agent job 1 and rename it to a decent and descriptive it name, for example: BizTalk Server Build Agent Job



o Once that done, let's select our Build solution ***.sln task and provide the following configuration:

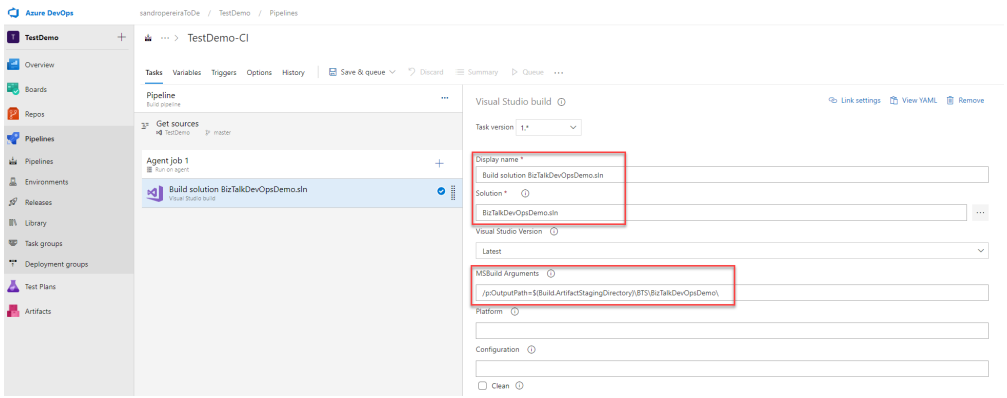
• On the Display name: give a name to your task, for example, Build solution BizTalkDevOpsDemo.sln

• Optional: On the Solution name, set the solution name BizTalkDevOps Demo.sln

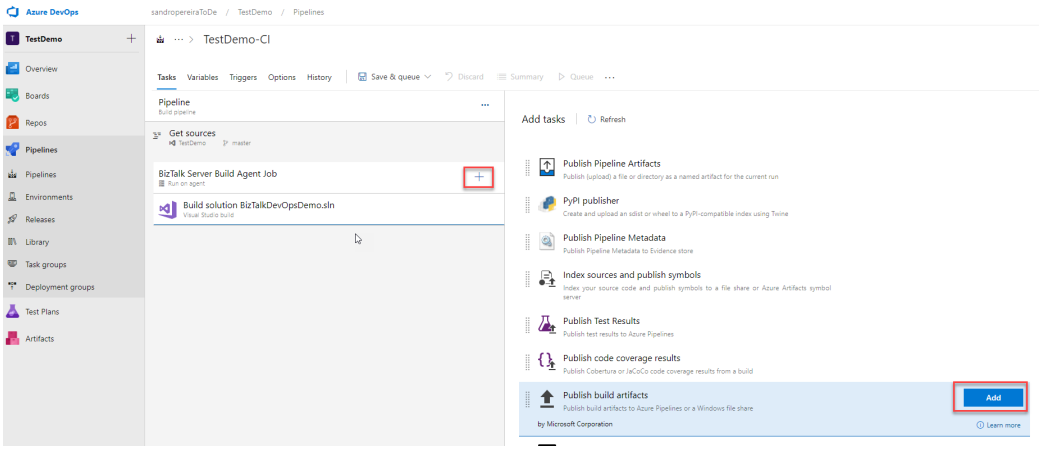
• Optional: On the MSBuild Arguments property let's force the Output Path just to be sure.

• /p:OutputPath=$(Build.ArtifactStagingDirectory)\BTS\BizTalkDevOpsDemo\

• Leave the rest of the default values.

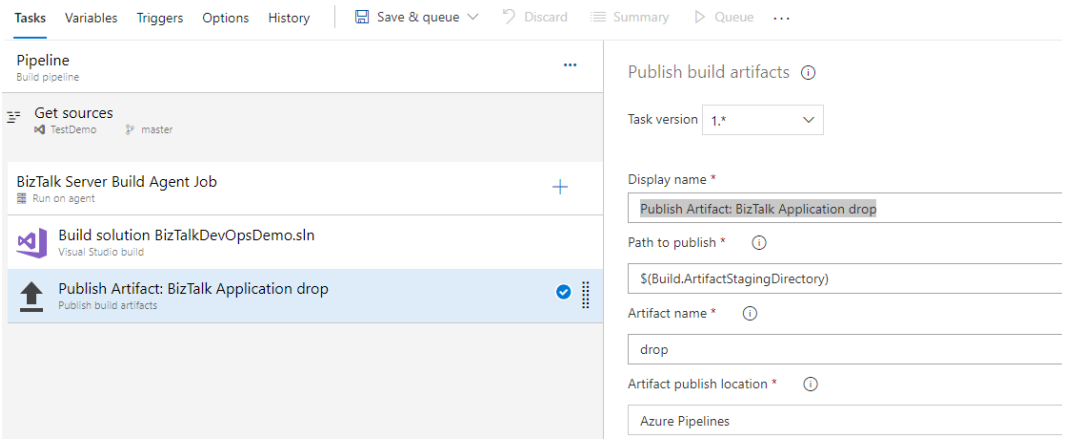


o On our Agent, click on Add a task, and from the Add tasks panel, search for Publish, select Publish buid artifacts task and click Add.

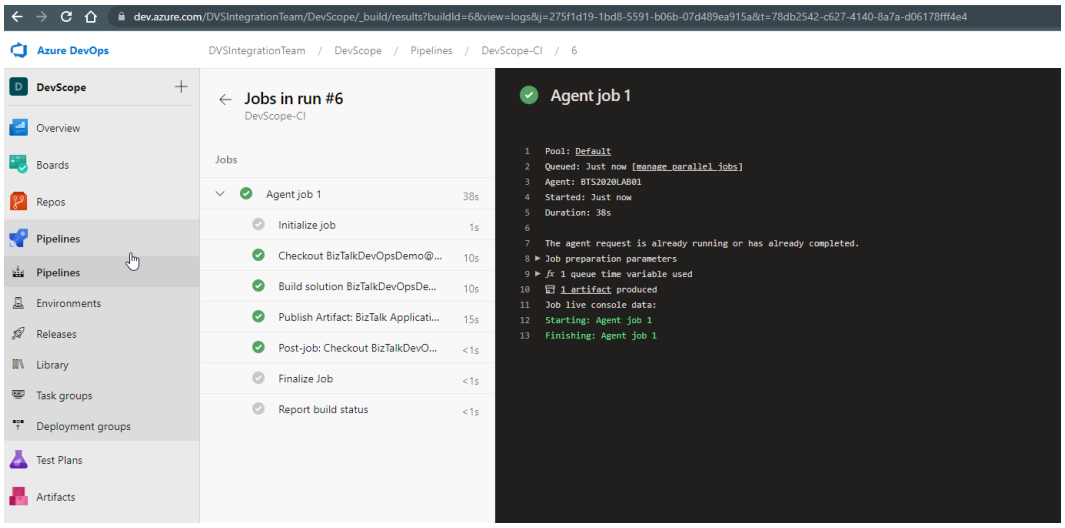


o On the **Publish Artifacts** task provide the following configuration:

- On the **Display name:** give a name to your task, for example, **Publish Artifact: BizTalk Application drop**
- Leave the rest of the default values.



- We can now select **Save & queue** to run the pipeline.



If everything goes well, all tasks should have a green icon.

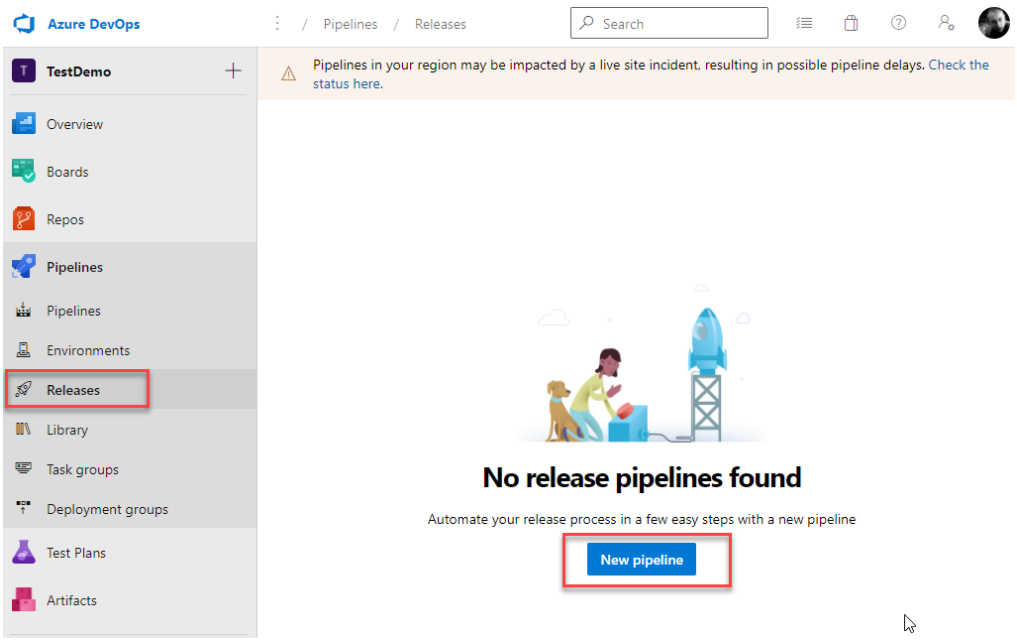
Building the Release Pipeline

Azure Pipelines provide a highly configurable and manageable pipeline for releases to multiple stages such as Development, Staging, QA and Production. it also offers the opportunity to implement gates and approvals at each specific stage.

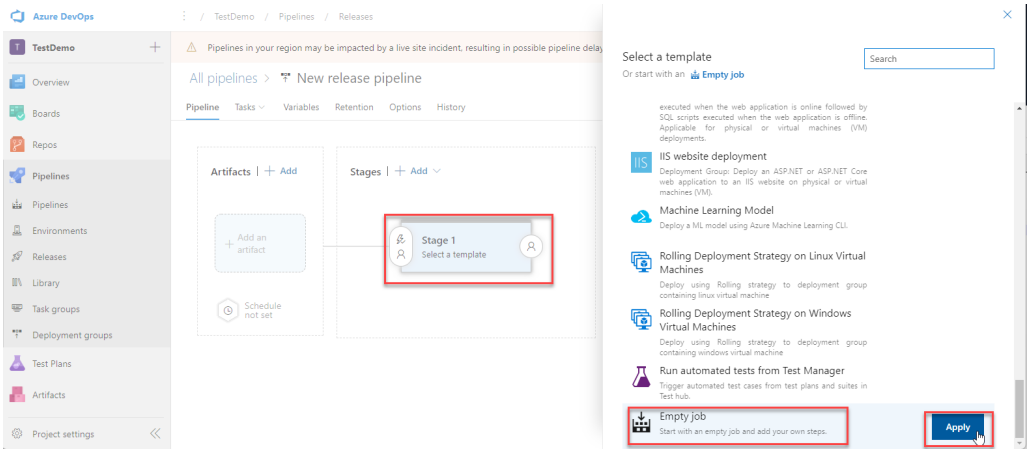
What we'll need is a release pipeline that contains at least one stage. You must choose stage names that are unique, but it's a good idea to include for example QA in the name of one and Production in the name of the other, so that you can easily identify for where you are deploying.

To do this we need to:

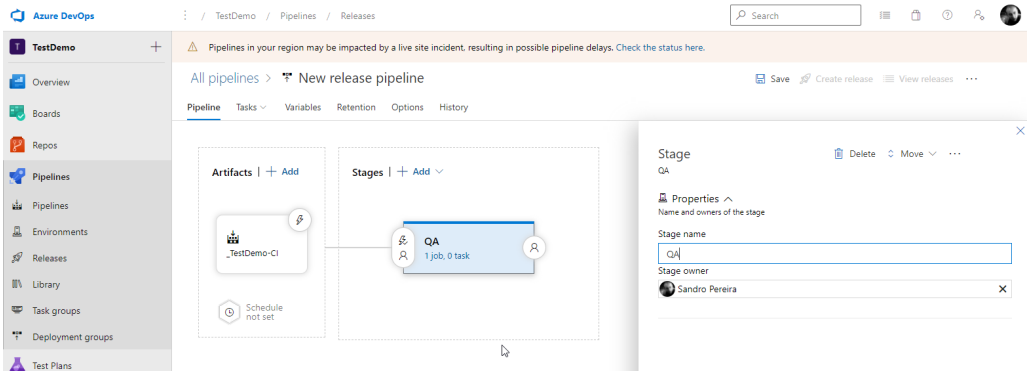
- In Azure DevOps to create a release go to Pipelines and then Releases. Since this is our project's first Release we have a New pipeline button to click to start the creation process.



• The New pipeline button will start the creation process by showing a Select a template panel. Scroll down to the end and select the Empty job option and then click Apply.



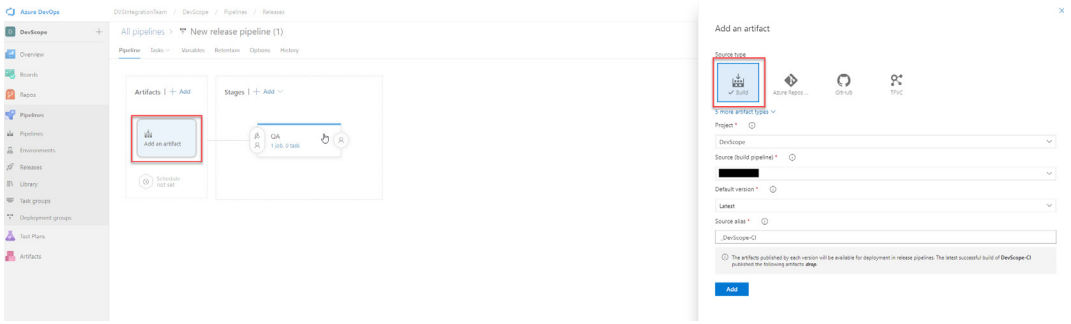
• On the Stage properties, let's go and change the **Stage name** property to **QA.**



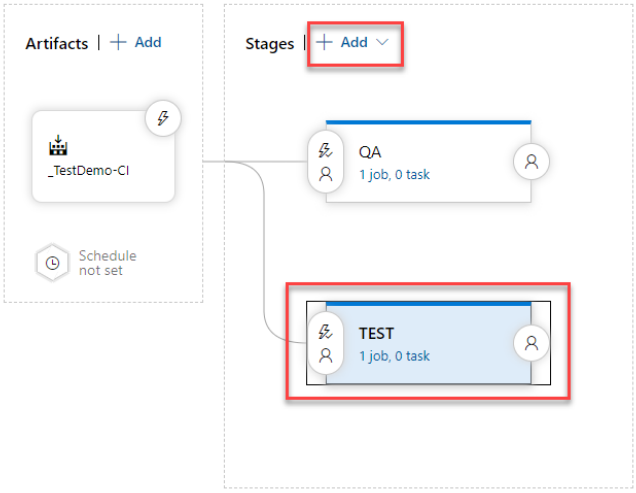
• Next, we want to add the artifacts from our build Pipeline to this Release. Select the **Add an artifact** box, on the **Add an artifact** panel perform the following configurations and then click **Add:**

- o On the **Source type:** select **Build** since our artifacts are the result of an Azure DevOps Build.
- o On the **Project:** select the project that contains the build pipeline
- o On the Source (build pipeline): select the build pipeline that publishes the artifact
- o Leave the rest of the default values. You can choose a specific drop version, but remember, it will be a static release.

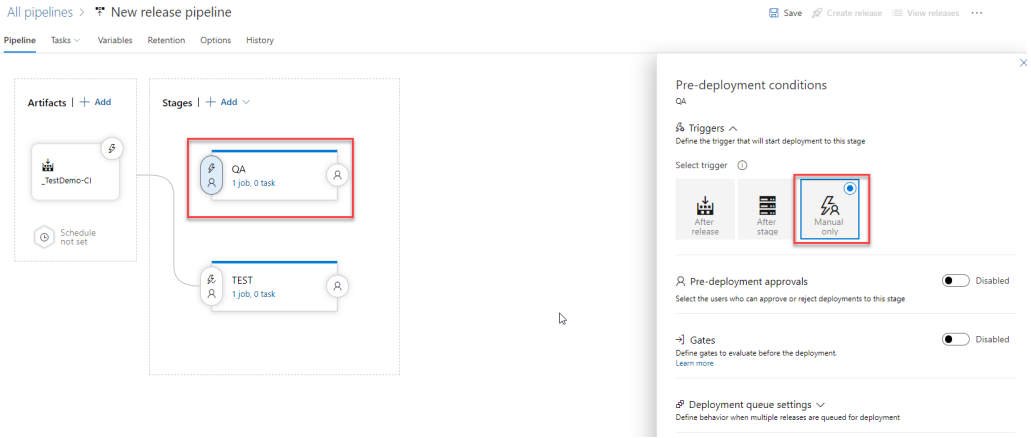
o Take note of the Source alias, as you will use it in the stages.



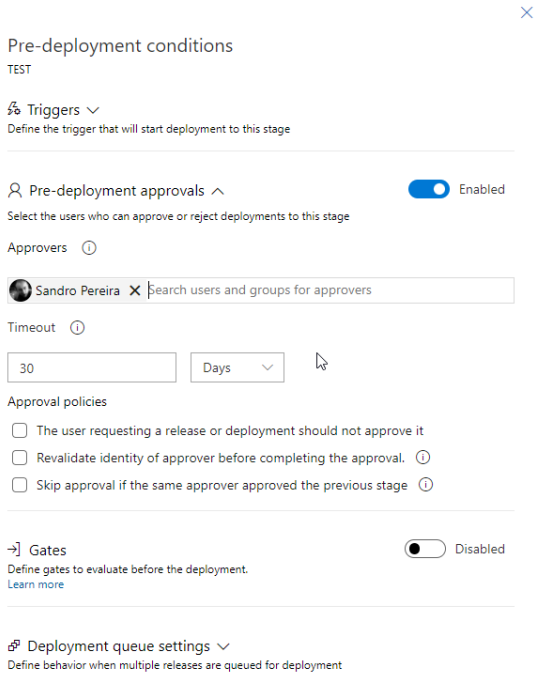
o You can then add new **Stages** like **TEST** or PRD. You can add empty stages or Clone previously created ones.



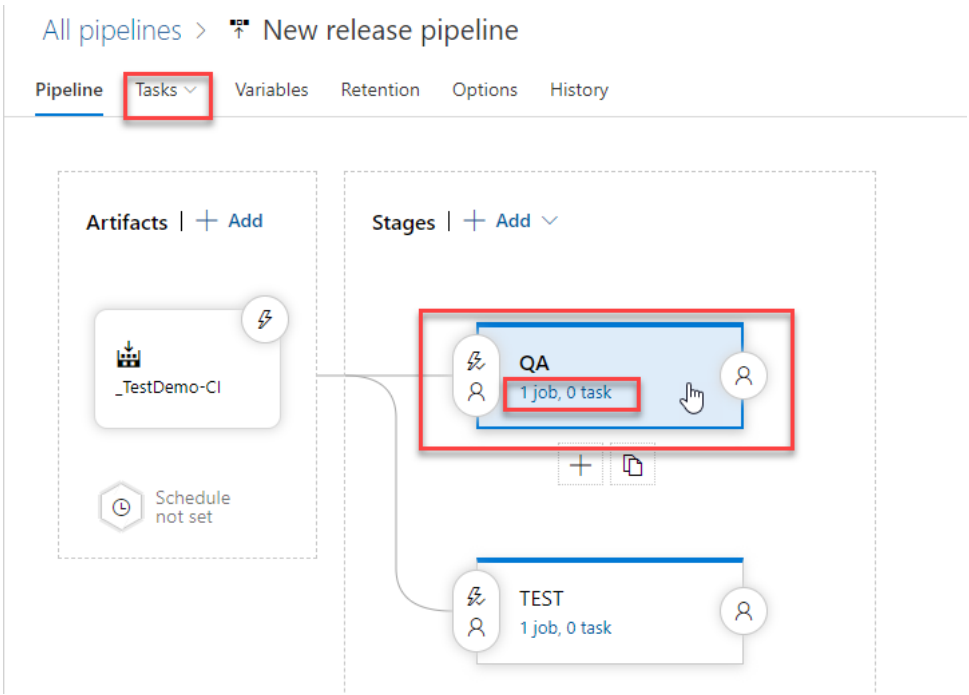
• At this moment, because we want to control the deployment, we will configure it to run manually on demand. To do that, select the **QA** Stage and on the **Triggers** under **Pre-deployment conditions** choose Manual only.



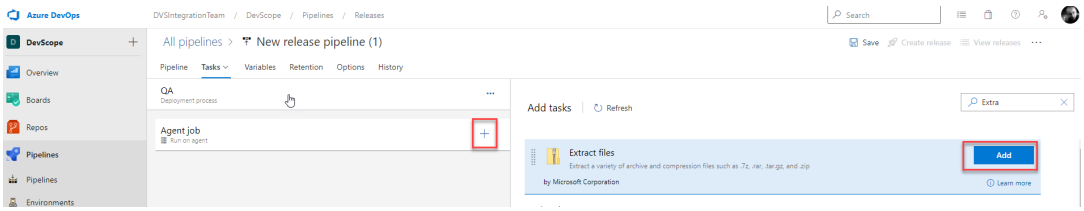
o You can always configure also the Pre-deployment approvals. This can be important for deployment into production environments.



• Now that we define the Stages, we need to define the tasks for each stage. For that, under **Pipeline** tab, select the **QA** stage and click the **Tasks** tab (or click on the link **1 job, 0 task** inside the QA stage).

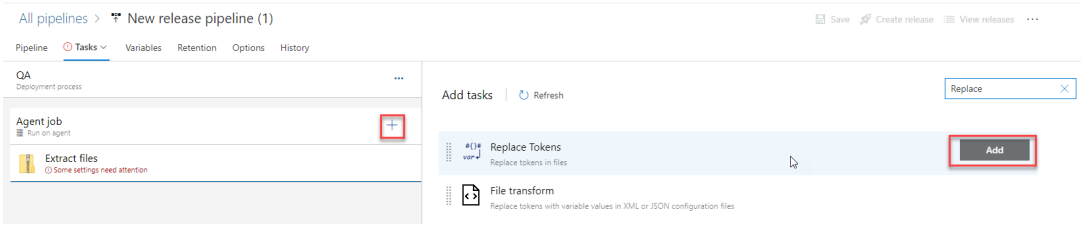


• On the **QA stage Tasks** panel, click on the **+** (plus) button under the **Agent job.** From the **Add tasks** panel, search for **Extract,** select **Extract Files** task and then click **Add.**

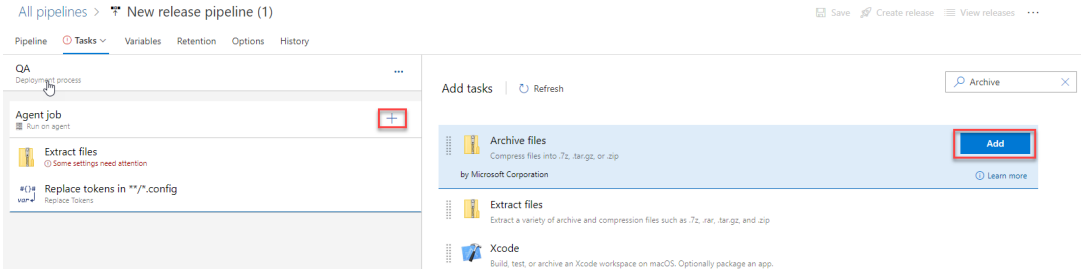


o Do the same for the tasks:

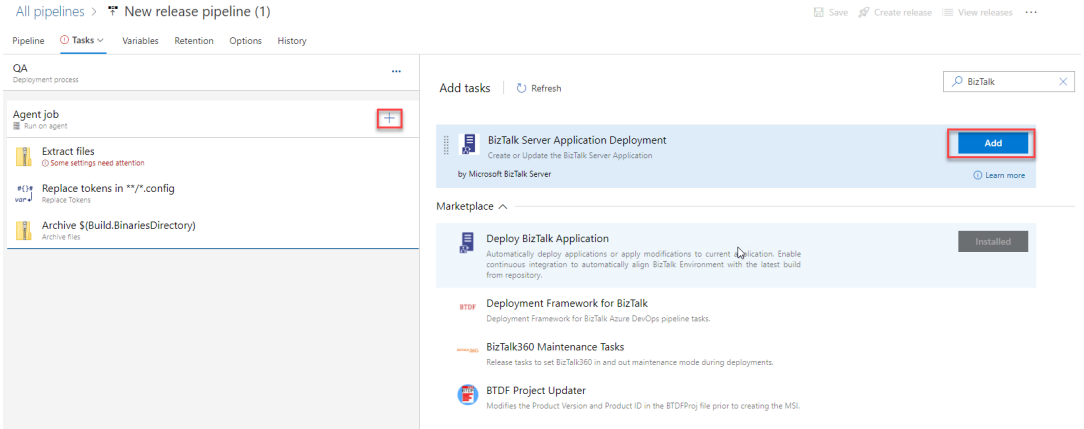
- **Replace Tokens**



- **Archive Files**



- **BizTalk Server Application Deployment.**
- we recommend this task, but you can easily do it with PowerShell

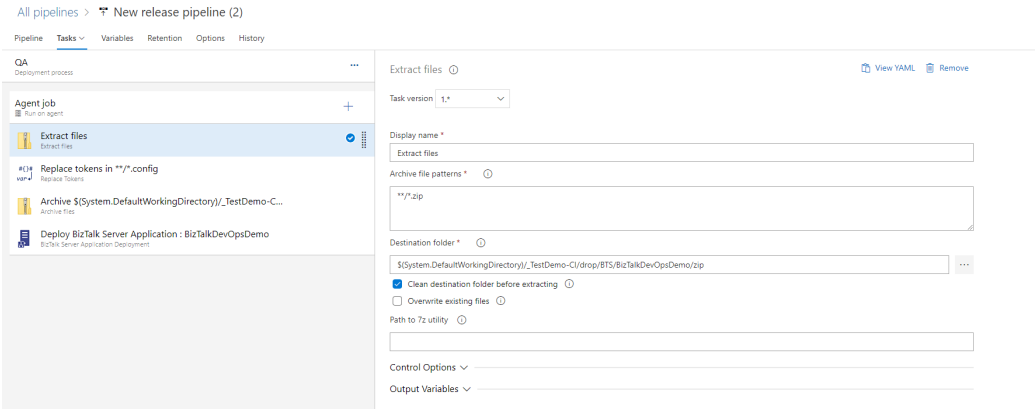


• Extracting your file contents is a straight forward task, you just need to select your zip utility and your destination folder. Keep in mind that you will need to know where it lands, to zip it back. On the **Extract Files** task provide the following configuration:

o On the **Destination folder**: Provide the destination folder into which archive files should be extracted. Use variables if files are not in the repo. Example: $(agent.builddirectory).

• $(System.DefaultWorkingDirectory)/_TestDemo-CI/drop/BTS/BizTalkDevOpsDemo/zip

o Leave the rest of the default values.



• Replacing the Tokens is just as before, you select the *.XML mask or point directly to your bindings and select the Token that it should be looking for. Remember, that the variables you define are case sensitive. You can also use a Variable Group, it is a great way of defining your environment specific variables or common variables that you might have. On the **Replace Tokens** task provide the following configuration:

o On the Display name: give a name to your task, for example, **Replace tokens in BindingInfo.xml**

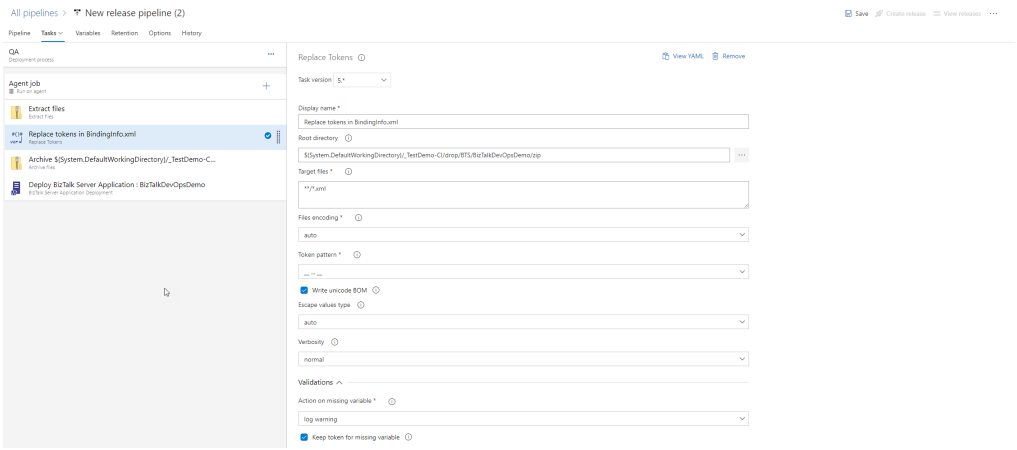
o On the **Root directory:** specify the base directory for searching files. If not specified the default working directory will be used. In our case: **$(System.DefaultWorkingDirectory)/_TestDemo-CI/drop/BTS/BizTalk DevOpsDemo/zip**

o On the **Target files:** we need to change that for json instead config. In this case: ****/*.xml**

o On the **Token pattern**: Specify the token pattern. We will be using __...__

o On **Validations** section, enable the option **Keep token for missing variable.**

o Leave the rest of the default values.



• Once this is done you can proceed to recreate the Zip file and it's contents. The destination folder you've selected when Unzipping will now be the Root folder you are pointing to. On the Archive Files task provide the following con figuration:

o On the Root folder or file to archive, set the same value you defined on the destination folder on the Extract Files task

• $(System.DefaultWorkingDirectory)/_TestDemo-CI/drop/BTS/Biz TalkDevOpsDemo/zip

o Unselect the Prepend root folder name to archive paths option.

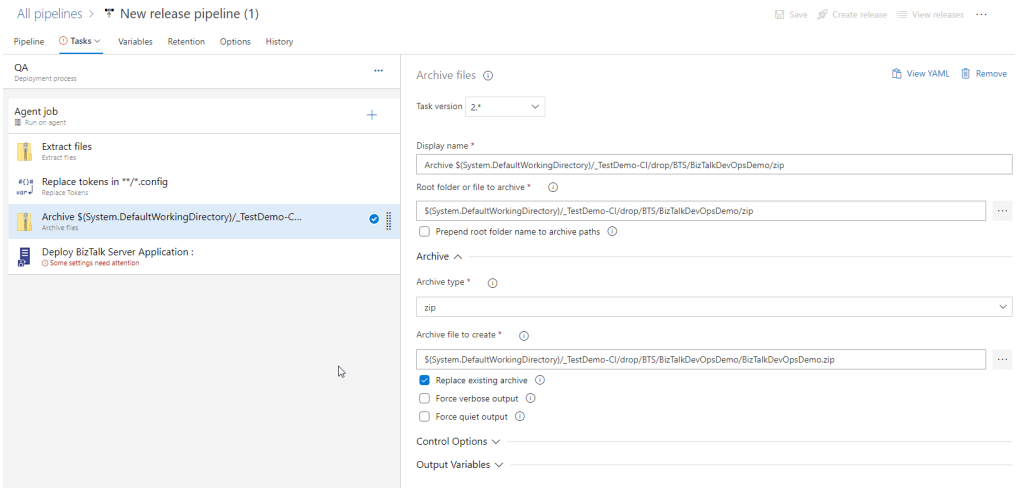
• If you keep this selected, your file will end up with a structure like "Zip / bindings" instead of just "bindings" and the deployment will fail, be cause it's not the expected folder structure.

o On the Archive file to create, specify the name of the archive file to create. For example, to create foo.tgz, select the tar archive type and gz for tar compression.

• $(System.DefaultWorkingDirectory)/_TestDemo-CI/drop/BTS/Biz TalkDevOpsDemo/BizTalkDevOpsDemo.zip

o Make sure you select the Replace existing archive option. By doing that, if an existing archive exists, specify whether to overwrite it. Other wise, files will be added to it.

o Leave the rest of the values as default.



• For the final step, the Deployment Task. I chose to use the standard task instead of PowerShell, because I didn't want to handle scripts at this point. On the Deploy BizTalk Server Application task provide the following configuration:

o On the Display name, set the same a proper name to this task.

• Deploy BizTalk Server Application : BizTalkDevOpsDemo

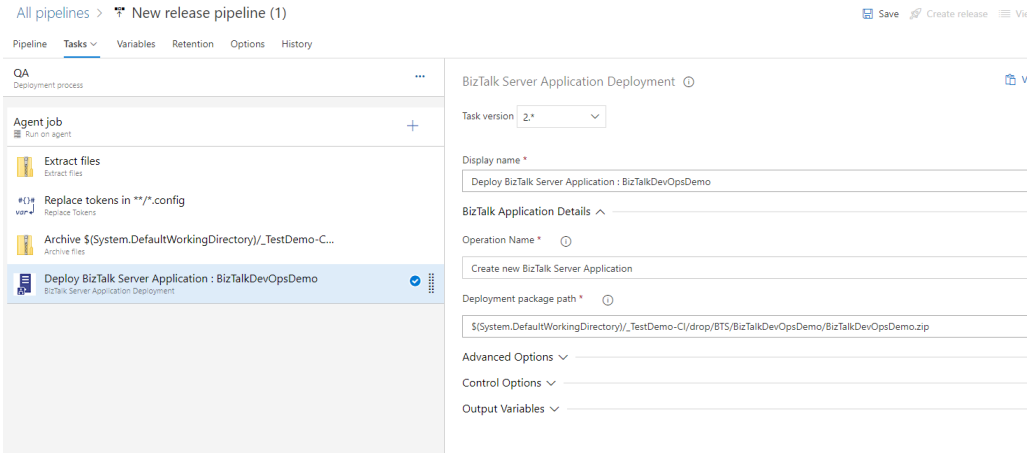
o On the Operation Name you need to select the Action to be performed on the BizTalk Application deployment. In this case we want:

• Create new BizTalk Server Application.

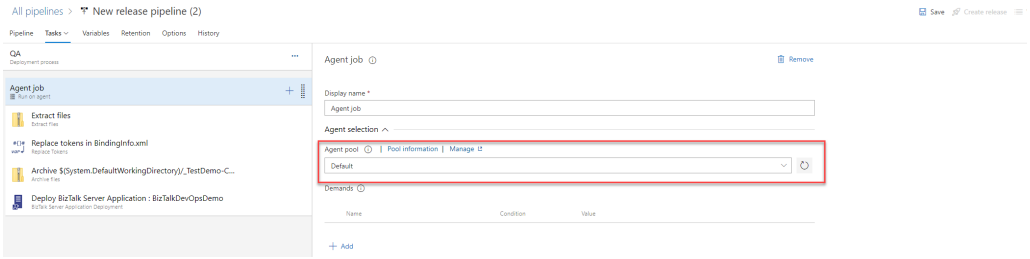
o On the Deployment package path, specify the path of BizTalk Application package in zip format..

• $(System.DefaultWorkingDirectory)/_TestDemo-CI/drop/BTS/BizTalkDevOpsDemo/BizTalkDevOpsDemo.zip

o Leave the rest of the default values.

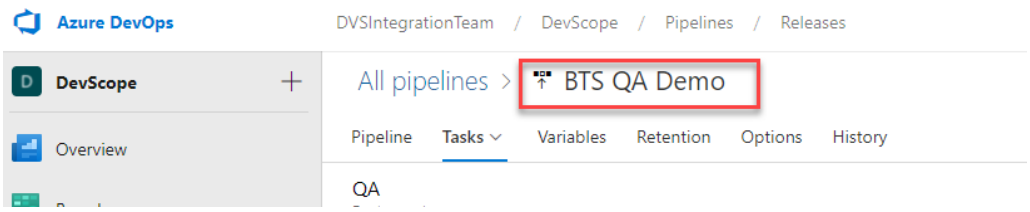


• Finally, we need to select the Agent Job and specify which agent to be executed.



• If you have multiple BizTalk Server machines, you may need to create different Agents on this release pipeline in order to deploy into different machines.

• Do not forget to rename your Pipeline release:

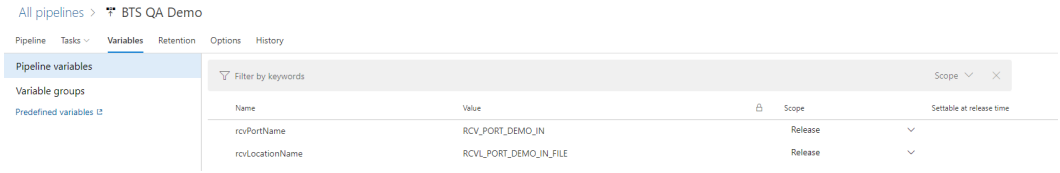


• Do the same steps to your other Stages like the TEST stage that we create previously.

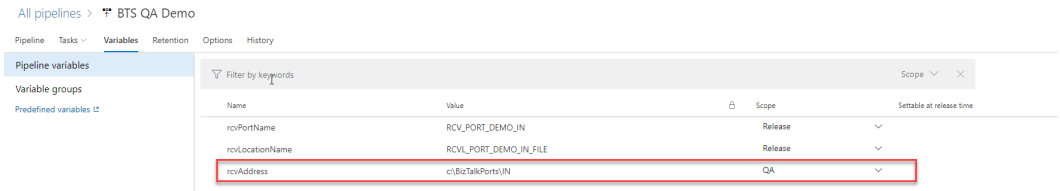
Defining the Variables

Now that we already defined the tasks, it is time for us to define the Variables, similar to what we did previous in the pipeline.

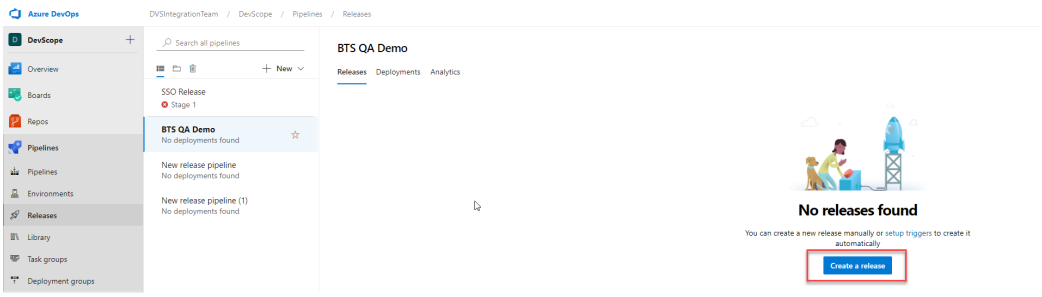
- Here we are going to add all the variables that exists on the Binding.xml that exists inside you Visual Studio BizTalkDevOpsDemoApplicationProject project but we also have to define the scope of each variable:
    - o We need to add two **Pipeline variables** and define the scope to **Release, QA** and/or **TEST** with different values;
    - o Add one variable called **rcvLocationName**, define the scope to **RELEASE** and set the value to **RCVL_PORT_DEMO_IN_FILE**.
    - o Add one variable called **rcvPortName**, define the scope to **RELEASE** and set the value to **RCV_PORT_DEMO_IN**.
- These two variables are set as release since we are not changing the name per environment. Actually there was no need for us to tokenize these values but we used the to serve as demonstration.



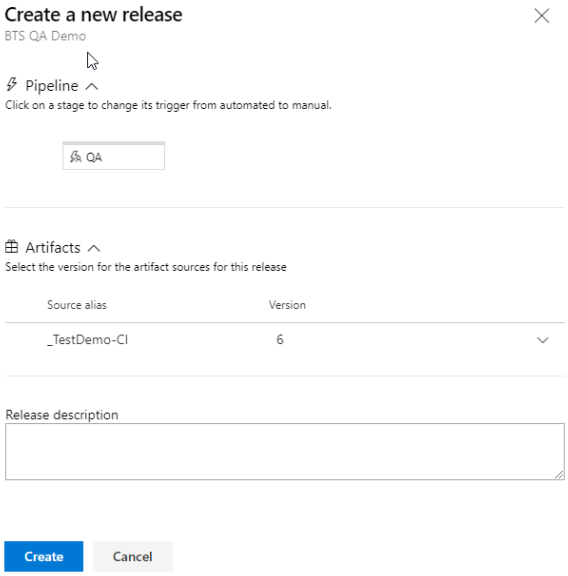
- o Then create another variable called rcvAddress, define the scope to **QA** and set the value to **c:\BizTalkPorts\IN.**
- You should create this variable and define the value for each environ ment you have



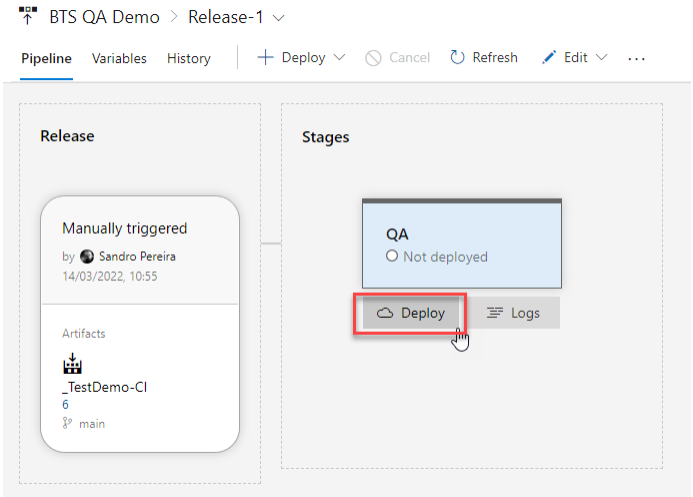
- Click **Save.**
- Finally, let's go back to the **Pipeline** tab and click on **Create release.**



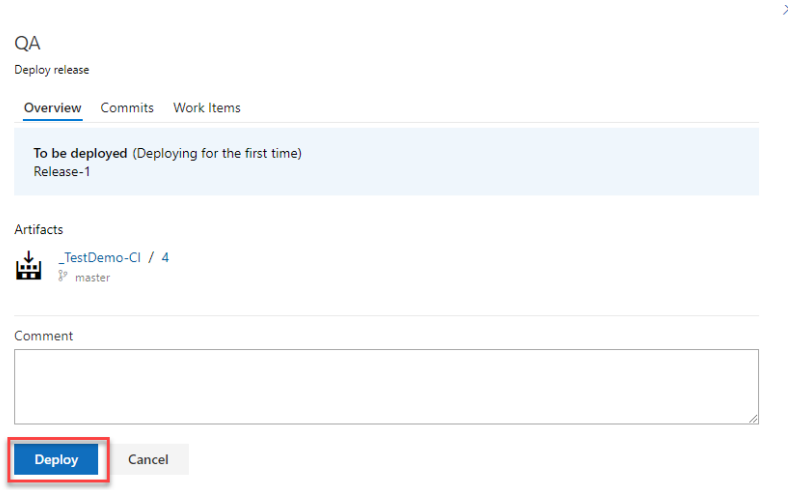
• On the **Create a new release** panel, add a **Release description** and click **Create.**



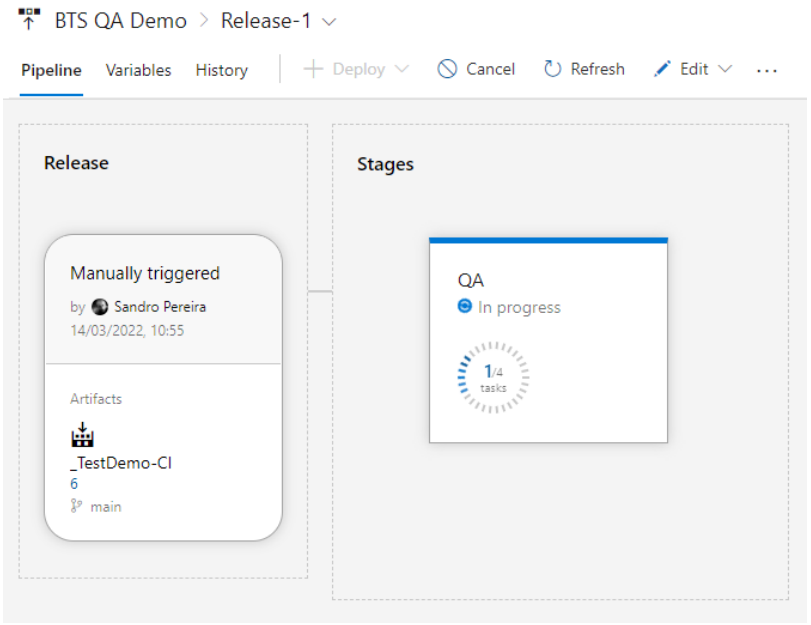
• Under the **Release-1** panel, go to the **QA** stage and click **Deploy.**



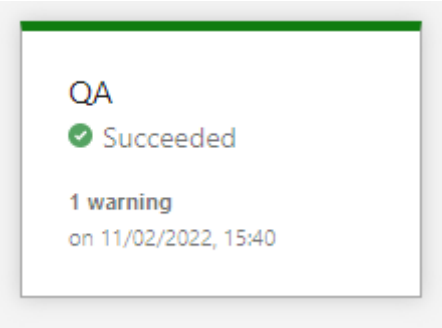
• On the **QA Deploy release** window, insert a **Comment** to your deployment (a good best practice) and click **Deploy.**



• After that, the deployment starts, as soon as the Agent is ready to, and you will see the progress.



• If everything goes according you will end with a Succeeded status.



Indeed preparing our BizTalk Server solution for deployment can be a challenge and with a huge amount of manual configurations we need to do but trust us when we say that this amount of work will save not only time in the future but also a lot of headaches. This is a must-do task if you are working with multiple servers BizTalk Server environments in an enterprise scenario.

Plus, it also takes out the need for every developer and/or administrator that works with a given project, to create the deployment tasks and spread this across environments.

So if you think of it and put it in numbers, although it has a development cost, it saves time in the future, because you will not have the 3 hours of deployment time to get everything prepared, every variable renamed, all the work done over and over and over again.

Thinking long term, it is a low-cost high-return scenario.

Using SSO Application Configuration with CI/CD

BizTalk Server leverages the Enterprise Single Sign-On (SSO) capabilities for securely storing critical information such as secure configuration properties (for example, the proxy user ID, and proxy password) for the BizTalk adapters. Therefore, BizTalk Server requires SSO to work properly. BizTalk Server automatically installs SSO on every computer where you install the BizTalk Server runtime.

But it also can keep your own application configuration data in SSO database, let's say the usual configurations that we normally keep in a configuration file ("app.config"). If you've been in the BizTalk world long enough, you've probably faced this challenge or need and until 2009 there wasn't an easy way to achieve that and Richard Seroter's BizTalk SSO Configuration Data Storage Tool was the go to-tool to store and manage Single Sign-On (SSO) applications. his is still a valid tool, and if you rebuild the code in the last version of BizTalk Server it still works perfectly.

Unfortunately, there is no command line tool to allow you to script the deployment SSO Application Configurations or perform CI/CD thru DevOps. However my team and I created an application called SSO Application Configuration CLI Tool that is a command-line tool that provides the ability to import SSO configuration applications – key-value pairs in the SSO database – that can be deployed to different environments. This enables you to script these tasks.

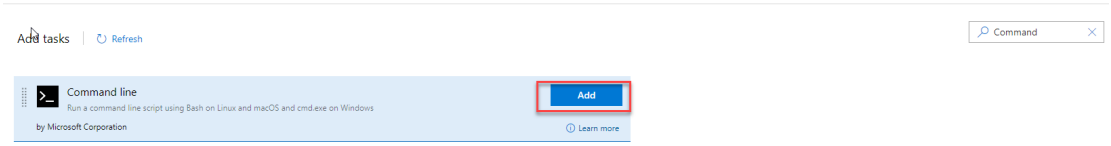
This tool is designed to address this gap on the Azure Pipelines allowing you to securely import Application configurations by using this CLI application;

It mandatory accepts 5 arguments:

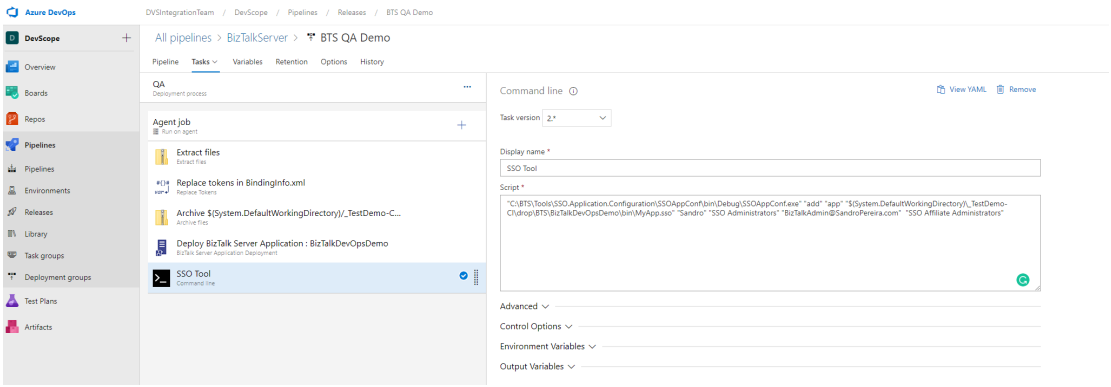
- **Path:** full path to the .sso file
- **Password:** password to unencrypted the .sso file
- **Contact Info:** Internal field that is normally in the format of an email that is used internally in SSO tables for Application Configurations
- **Application User Account:** SSO Affiliate Administrators Group or the Group that will access (read) key.values, for example: BizTalk Application Users.
- **Application Admin Account:** SSO Administrator Group – Administrators of the Enterprise Single Sign-On (SSO) service.

You can download BizTalk Server SSO Application Configuration Tool from GitHub here: https://github.com/BizTalkCommunity/BizTalk-Server-SSO-Application-Configuration-CLI

• In these cases, what you need to do is to add another task to your pipelines. Go to your pipeline and click on the **+** (plus) button under the **Agent job**. And from the **Add tasks** panel, search for **Command**, select **Command line** task and then click **Add.**




```
"C:\BTS\Tools\SSO.Application.Configuration\SSOAppConf\bin\Debug\SSOAppConf.exe" "add" "app" "$(System.DefaultWorking-Directory)\_TestDemo-CI\drop\BTS\BizTalkDevOpsDemo\bin\MyApp.sso" "Sandro" "SSO Administrators" "BizTalkAdmin@SandroPereira.com" "SSO Affiliate Administrators"
```



Notice that the BizTalk Server SSO Application Configuration Tool available on the GitHub needs to be installed (present) on one BizTalk Server machine at least and you only need to execute in one machine.

The SSO application also needs to be included on the drop folder inside your repo.

About BizTalk360

BizTalk360 is a one platform tool for Microsoft BizTalk Server Operations, Monitoring and Analytics. There are over 50 features targeted towards operations and monitoring of BizTalk Server in BizTalk360, making it an invaluable tool for using Microsoft BizTalk server. Time and time again, we came across the same problem each customer was having administering, operating, and monitoring their BizTalk environments. The default tooling (like BizTalk Admin Console) lacked some enterprise qualities like security, auditing, analytics, monitoring capabilities, and so on. It also demanded a high level of BizTalk knowledge to do the very basic things. Majority of the customers (specifically those who could afford) ended up building their own custom management and monitoring solutions. That's when we built a genericmanagement, monitoring, and analytics solution for BizTalk Server and make it affordable to each BizTalk Server customer.

Start your 30 days free trial

Start Free Trial

Visit BizTalk360 for more information