

Instruction manual: Configuring CI/CD with Bamboo for Microsoft Azure

Building and deploying a Azure Function Visual studio application on Azure via a Bamboo
server

Table of contents

1	Introduction.....	4
1.1	Purpose	4
1.2	Scope.....	4
2	Prerequisites	5
2.1	System requirements	5
2.2	Supported platforms	5
2.3	Licenses	5
2.4	Software	5
3	About Bamboo.....	7
4	Setting up a virtual machine on Azure	8
4.1	Configuring RDP and Desktop environment in Centos.....	8
4.1.1	Configuring xRDP	9
4.1.2	Configuring GNOME GUI	9
4.1.3	Configuring Remote Desktop traffic.....	9
5	Installing utilities for CentOS 8	10
5.1	Configuring Java JDK 8.....	10
5.2	Installing Mono.....	10
5.3	Installing Azure CLI	11
5.4	Installing NuGet CLI.....	11
5.5	Installing .NET SDK version 2.1	12
6	Setting up Bamboo	13
6.1	Installing Bamboo	13
6.2	Configuring Bamboo.....	14
7	Creating an Azure function app using Visual Studio	19
8	Creating a GitHub remote repository	21
9	Creating the Azure function app in Azure.....	22
9.1	Configuring the Azure Function settings.....	22
10	Creating and Configuring a Bamboo build plan	24
10.1	Task 1: Creating a Source Code Checkout task.....	26
10.2	Task 2: Creating a NuGet restore script	26
10.3	Task 3: MSBuild task.....	27
10.4	Task 4: Script ZIP file task.....	27
10.5	Creating an artifact.....	28
10.6	Configuring a Bamboo deployment project	29

10.7	Configuring triggers	31
10.8	Adding the trigger to your deployment plan.....	32
10.9	Test the Azure Function app	33

1 Introduction

Bamboo is a continuous integration or CI server that can be used to automate the build, test and release management for a software application. Automating the management of an application creates a continuous integration pipeline.

1.1 Purpose

This manual explains how to set up the continuous integration pipeline on the Cloud using Bamboo in combination with Azure. The purpose of this manual is to provide new users with information to quickly get started with Bamboo.

1.2 Scope

The scope of this manual pertains to downloading and installing Bamboo on a Linux virtual machine running on Azure. Furthermore, Bamboo will be implemented to build and deploy a .NET Visual studio 2017 application on a GitHub repository.

2 Prerequisites

Before downloading and installing a Bamboo server, make sure your machine meets the minimum specifications. In this manual we are going to install Bamboo on a virtual machine (VM) in the Azure Cloud environment, so that the continuous delivery pipeline is located completely on the cloud. Alternatively you can choose to download the Bamboo server on your own device, the procedure will remain the same.

2.1 System requirements

For Bamboo the minimum CPU and memory requirements differ depending on the size and complexity of your plans.

For individual users and small teams the requirements are:

- **CPU:** Quad core 2GHz+ CPU
- **RAM:** 4GB
- **Minimum database space:** 10GB
- **Storage:** 20 GB

For more information on hardware specifications visit the link: [Bamboo Best Practice - System Requirements](#)

2.2 Supported platforms

Make sure you're using an operating system that is supported. This means only on x86 and 64 bit x86 platforms. Bamboo can run on Windows, Linux, Mac OS X and Solaris.

For more information on supported platforms visit the link: [Supported platforms](#)

2.3 Licenses

Bamboo offers a 30 day trial license for consumers to try the software. The pricing is based on required agents¹ rather than the number of users. The more agents you take the higher the price. For small teams you pay 10 dollar and for unlimited local agents. For growing teams you pay 1270 dollar per remote agent.

2.4 Software

- Virtual machine in Azure cloud with Centos8 (*please see chapter 4 Setting up a Virtual Machine in Azure*)
- GUI on your virtual machine in order to configure Bamboo (*please see chapter 4.1.2 Configuring GNOME GUI*)
- Bamboo Server (*please see chapter 5 Setting up Bamboo*)
- Github Remote Repository (or Bitbucket)
- Visual Studio 2017/2019

¹ A bamboo agent is a service that provides capabilities to run a service.

- Installed utilities (JDK 1.8, Mono, Azure CLI, Nuget and .NET sdk version 2.1) – *Please see 5 installing utilities for Centos*

3 About Bamboo

Bamboo is a Continuous Integration (CI) server that can be used to automate the release management for a software application, thus creating a continuous delivery pipeline. This means that builds, unit tests, integration tests and so forth are performed or triggered whenever code is committed to the repository. This methodology ensures that new changes are integrated well into the existing code base. Integration builds provide feedback on the quality of new changes.

Bamboo solves numerous problems for solo and team projects. Solo developers can rely on the automated builds and test processes to check for the quality of new code, therefore allowing them to focus on coding. Another advantage of Bamboo is that the deployments can be automated to a server, such as App Store, Google Play or Azure. Additionally Bamboo is a way to manage different builds that have different targets or requirements. This is useful because production builds have different requirements than development or test builds.

The above advantages also work for teams. In addition, builds and test processes are not dependent on a specific local environment. Lastly, builds and integration tests are triggered automatically as soon as the developer commits code, thus allowing for continuous integration.

Bamboo creates these advantages by implementing the functionalities below:

- Bamboo is a central management server which schedules and coordinates all work.
- Bamboo implements interfaces and plugins for different types of work.

Bamboo works in the following order:

1. Bamboo retrieves your source from a source repository.
2. Bamboo starts the build by calling something like MSBuild to build your Visual Studio solution.
3. Once the solution or project is built, you have artifacts like build results, executable apps, config files, etc.
4. With these artifacts you can zip them up and copy them or install them on a test server to make sure everything runs fine.

4 Setting up a virtual machine on Azure

We recommend setting up a virtual machine through: <https://portal.azure.com/#home> especially for beginners since you can get \$100 credit for free when you have a school account.

First you make an account on Azure or use an existing account. After signing in you click on **Virtual Machines** under the Azure services section. Click on + Add → + Virtual machine.

[Home](#) > [Virtual machines](#) >

Create a virtual machine

Project details

Select the subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription *	Gratis versie
Resource group *	(New) MyBamboo_group

[Create new](#)

Instance details

Virtual machine name *	MyBamboo
Region *	(Europe) West Europe
Availability options	Availability zone
Availability zone *	1
Image *	CentOS-based 8.2 - Gen1
Browse all public and private images	
Azure Spot instance	<input type="radio"/> Yes <input checked="" type="radio"/> No
Size *	Standard_A4_v2 - (€ 112,66/month)
Select size	

Image 1. Creating a virtual machine on Azure

Follow the parameters for availability options, image and size. For size: Standard_A4_v2 click on the dropdown list → all sizes. For inbound ports choose HTTP, HTTPS, SSH. You can also add RDP if you want to use remote desktop. In this manual we will use RDP to access the virtual machine. After this you can click Review + create.

4.1 Configuring RDP and Desktop environment in Centos

We have to configure a Remote Desktop Protocol (RDP) and Desktop environment to graphically control the Centos 8 VM. This will allow us to log in to the remote machine and create a desktop session. Follow the steps below to install and configure RDP and desktop environment.

4.1.1 Configuring xRDP

We will use xRDP to allow for remote control on the CentOS 8. XRDP is a free and open-source implementation of the Microsoft RDP. XRDP is available in the EPEL software repository. If epel is not enabled in your system enable it with the following command:

```
$ sudo yum install -y epel -release
```

Install xRDP:

```
$ sudo yum install -y xrdp tigervnc-server
```

Enable and start the software:

```
$ sudo systemctl enable xrdp  
$ sudo systemctl start xrdp
```

4.1.2 Configuring GNOME GUI

We will use GNOME Desktop GUI (graphical user interface) for the desktop environment. Gnome is a free and open-source desktop environment for Unix-like operating systems.

Install GNOME:

```
$ sudo yum groupinstall "Server with GUI" -y
```

Check the default settings of GNOME:

```
$ systemctl get-default
```

Output:

```
multi-user.target
```

If our target is `multi-user.target`, it means the GUI will not be loaded. To resolve this we have to set the default target to `graphical.target`.


```
$ sudo systemctl set-default graphical.target
```

Run the following command to change to the GUI immediately:

```
$ sudo systemctl isolate graphical.target
```

4.1.3 Configuring Remote Desktop traffic

Create a network security group rule for Remote Desktop traffic. To allow for Remote Desktop traffic to reach your CentOS VM, a network security group rule needs to be created that allows for TCP on port 3389 to reach your VM.

Run the following command in Azure CLI (Click on the  icon in the top right corner of the Azure portal to open the Azure CLI). Change myResourceGroup and myVM according to your own resource group and VM names:

```
az vm open-port --resource-group myResourceGroup --name myVM --port 3389
```

5 Installing utilities for CentOS 8

Before we can set up Bamboo, we have to install some software on the virtual machine so that Bamboo can deploy without errors.

5.1 Configuring Java JDK 8

Before we can run Bamboo we need to download and install the supported JDK (not JRE). In Bamboo's case the supported JDK version is 8. Bamboo will use the JDK to run its application. Follow the steps below to install and configure the JDK.

1. Open the terminal.
2. Install the JDK with the following command (The '\$' is used to indicate a command and is therefore not part of the command):

```
$ sudo dnf install java-1.8.0-openjdk-devel
```

3. Set up the JAVA_HOME path variable. This is an environment variable that will point to the correct Java installation directory. Type in the following commands:

```
$ export JAVA_HOME=$(dirname $(dirname $(readlink $(readlink (which javac)))) )
```

```
$ sudo nano /etc/profile.d/java.sh
```

```
$ JAVA_HOME="/usr/lib/jvm/java-1.8.0-openjdk-1.8.0.265.b01-0.el8_2.x86_64"
```

4. Refresh the file with the following command:

```
$ source /etc/profile.d/java.sh
```

5. Test if the path is set successfully with the following command:

```
$ echo $JAVA_HOME
```

The result should show the java version 8. You can find more information about the JDK configuration here: <https://linuxize.com/post/install-java-on-centos-8/>.

5.2 Installing Mono

Mono is a free and open-source project of Microsoft's .NET Framework based on the standards for C# and the Common Language Runtime. We can use mono to compile C# code on Linux. Follow the steps below to install Mono:

1. Open the terminal.
2. Type in the following commands to add the Mono repository to your virtual machine:

```
$ rpmkeys --import http://pool.sks-keyservers.net/pks/lookup?op=get&search=0x3fa7e0328081bffa6a14da29aa6a19b38d3d831ef
```

```
$ su -c 'curl https://download.mono-project.com/repo/centos8-stable.repo | tee /etc/yum.repos.d/mono-centos8-stable.repo'
```

3. Install Mono with the following command:

```
$ yum install mono-devel
```

The package mono-devel should be installed to compile code. Alternatively you can choose mono-complete to install everything. This can resolve errors like ‘assembly not found’

5.3 Installing Azure CLI

The Azure command-line interface (CLI) is a set of commands used to create and manage Azure resources. In Bamboo we will use an Azure CLI script to log into Azure. Therefore we need to install the Azure CLI on the virtual machine.

1. Open the terminal
2. Import the Microsoft repository key with the following command:

```
$ sudo rpm --import https://packages.microsoft.com/keys/microsoft.asc
```

3. Create local Azure CLI repository information

```
$ sudo sh -c 'echo -e "[azure-cli]
name=Azure CLI
baseurl=https://packages.microsoft.com/yumrepos/azure-cli
enabled=1
gpgcheck=1
gpgkey=https://packages.microsoft.com/keys/microsoft.asc" >
/etc/yum.repos.d/azure-cli.repo'
```

4. Install Azure CLI with yum install command:

```
$ sudo yum install azure-cli
```

5.4 Installing NuGet CLI

The NuGet CLI provides the full extent of NuGet functionalities.

1. Open the terminal
2. Download the latest stable nuget.exe to ‘usr/local/bin’ using the following command:

```
$ sudo curl -o /usr/local/bin/nuget.exe https://dist.nuget.org/win-x86-commandline/latest/nuget.exe
```

3. Create an alias by adding the following script to the nuget.exe file

```
$ alias nuget="mono /usr/local/bin/nuget.exe"
```

4. Reload the terminal (exec bash). Test the installation by entering the following command

```
$ nuget
```

The output should display NuGet CLI help.

5.5 Installing .NET SDK version 2.1

We will use .NET Core for our Function application because .NET Core is supported on CentOS. Bamboo will use the SDK to run the application.

1. Open the terminal
2. Install .NET SDK version 2.1 with the following command:

```
$ sudo dnf install dotnet-sdk-2.1
```

6 Setting up Bamboo

6.1 Installing Bamboo

1. Open the terminal.
2. Install wget to download files on the internet. Wget is a program that retrieves contents from web servers.

```
$ sudo yum install -y wget
```

3. Download Bamboo using wget. First create an installation directory in the terminal. This directory will contain Bamboo's application files. We will call this directory 'BambooInstallation' with the following command:

```
$ mkdir BambooInstallation
```

4. Locate the directory using the cd command and download the Bamboo installation file using wget:

```
$ wget  
https://www.atlassian.com/software/bamboo/downloads/binary/atlassian-bamboo-7.1.2.tar.gz
```

5. Unpack the .tar file using command:

```
$ tar -xvzf atlassian-bamboo-7.1.2.tar.gz
```

6. Create a Bamboo home directory. This directory contains Bamboo's configuration data. Go back to your home directory using the following command:

```
$ cd ~
```

Use the pwd command to find out your current path add this path name into the bamboo-init.properties file with nano. This file contains the path name of the bamboo home directory.

```
$ nano BambooInstallation/atlassian-bamboo-7.1.2/atlassian-bamboo/WEB-INF/classes/bamboo-init.properties
```

Now we can uncomment the second line (by removing # in front of the bamboo.home line). Add the path name from the pwd command after the = sign (replace the existing file path name).

7. Move to your Bamboo installation directory.

```
$ cd BambooInstallation/atlassian-bamboo-7.1.2/
```

Now we can start Bamboo by executing the start-bamboo.sh file.

```
$ ./bin/start-bamboo.sh
```

We can now open an internet browser like Firefox to configure Bamboo by typing <http://localhost:8085/> in the URL.

6.2 Configuring Bamboo

On the first time you load the localhost:8085 page it will look like image 5.

Welcome to Atlassian Bamboo continuous integration server!

Please enter your license information and choose a setup method below to complete the installation of Bamboo.

Enter your license

Server id **B4K6-9GN5-GR3M-G0BE**

License key*

Please enter your Bamboo license key above - either commercial or evaluation. Contact [Atlassian](#) if you require a license key.

Select setup method

Express installation

Installs Bamboo with default settings and an embedded database.

Recommended if you are evaluating or demonstrating Bamboo, as it will get you up and running as quickly as possible.

Express installation

Custom installation

Installs Bamboo but allows you to configure Bamboo with an external database, customize the default settings, and/or initialize the server with your own data.

Recommended if you are setting up a production instance.

Custom installation

Image 5: The localhost:8085 Bamboo page on first load.

On this page a unique **Server id** has been generated for you as a customer. If you have purchased a license on the Bamboo Atlassian website, please type in your **License key**. This should have been emailed to your email address. Please check your SPAM box if you didn't find anything in your mailbox.

Bamboo also offers a 30 day trial version for the customers who would want to explore the benefits of using Bamboo before buying it. In order to activate your trial version, please click on the [Atlassian](#) hyperlink that has been marked with a red line (see image 5).

My Atlassian

New Trial License

Product

License type

Bamboo (Server)

Organization

Your instance is: ☒ up and running
☐ not installed yet

Server ID

Please note we only provide trial support for 90 days per product.

By clicking "Generate License" below, you agree to the Atlassian [Software License Agreement](#) and [Privacy Policy](#).


 [Cancel](#)

Image 6: Atlassian website: Requesting new trial license

Choose your product as Bamboo, because you want your license to be valid for Bamboo. Type in your **Organization** name. Your instance is **up and running**, because you have connected to Bamboo through localhost and the **start-bamboo.bat** is currently active. As you might have noticed, the **Server ID** has automatically been filled with your own unique one (see image 6). Now it is time to generate your trial license by clicking the green button “**Generate License**”.

New Trial License

Filter by SEN, product, or name ☐ Licenses ☒ Trials 3 ☐ Archived

Collapse All

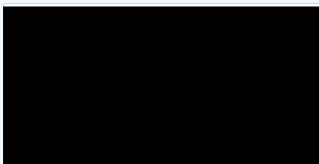
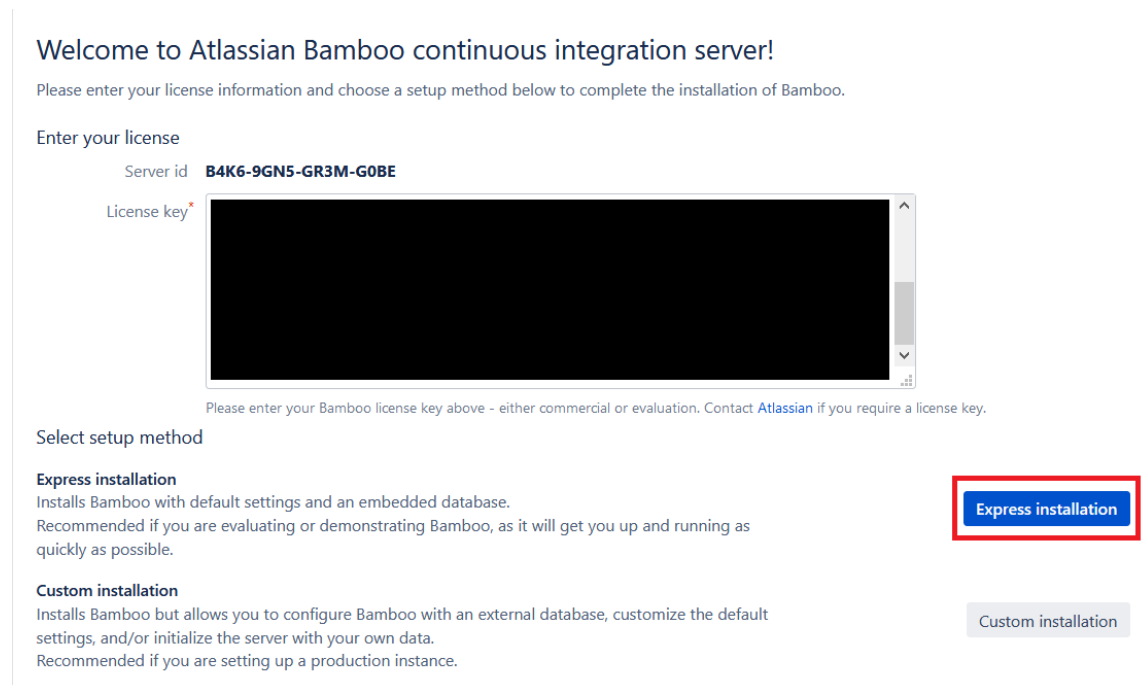
SEN	Product	Name	Support Expires	Support
<div> <div>⌵</div> <div>⌚ SEN-L16103782</div> </div>	Bamboo (Server): Trial	Cloud minor	28 okt 2020	Request Support
<div> <div> <div>➡</div> <div> <div>Server ID</div> <div>SEN ?</div> </div> <div>B4K6-9GN5-GR3M-G0BE</div> </div> <div> <div>License Key</div> <div></div> </div> <div> <div>Actions</div> <div>Buy Download Bamboo (Server) Archive</div> </div> </div>				
<div> <div>➤</div> <div>⌚ SEN-L16036275</div> </div>	Run CLI Actions in Bamboo: Trial	Cloud minor	14 okt 2020	Request Support
<div> <div>➤</div> <div>⌚ SEN-L16031826</div> </div>	Bamboo (Server): Trial	Cloud minor	14 okt 2020	Request Support

Image 7: Atlassian website: Overview of products

The site will redirect to the overview of products on which you have a valid license for. You can also find the expiry date of each license. To find your unique **License key** you need to search for the right product, in this case **Bamboo (Server): Trial**. When opening this product, you will find the corresponding **Server ID** and its generated **License key** (see image 7). Please make sure the **Server ID** matches with the one from the setup screen (see image 5),

When everything is correct, copy the whole **License Key** and go back to the setup screen. Paste it in the empty '**License key**' box.



Welcome to Atlassian Bamboo continuous integration server!

Please enter your license information and choose a setup method below to complete the installation of Bamboo.

Enter your license

Server id **B4K6-9GN5-GR3M-G0BE**

License key *

Please enter your Bamboo license key above - either commercial or evaluation. Contact [Atlassian](#) if you require a license key.

Select setup method

Express installation
Installs Bamboo with default settings and an embedded database.
Recommended if you are evaluating or demonstrating Bamboo, as it will get you up and running as quickly as possible.

Custom installation
Installs Bamboo but allows you to configure Bamboo with an external database, customize the default settings, and/or initialize the server with your own data.
Recommended if you are setting up a production instance.

Express installation

Custom installation

Image 8: The localhost:8085 Bamboo page on first load with license key.

You can install bamboo in 2 different ways:

Express installation: This type of installation is recommended for customers who are trying out Bamboo. It requires only a minimum of configuration information. It sets up Bamboo with default settings and an embedded database (H2).

Custom installation: The 'Custom Installation' method takes longer, but allows you to configure Bamboo with an external database, customize the default settings, and/or initialize the server with your own data.

For the ease of this manual, we will go with the **Express installation** method.

Bamboo – express installation

Please wait while Bamboo sets up your instance



Image 9: Bamboo - express installation: setting up instance.

Bamboo is currently setting up your instance. This may take a while.

Bamboo – express installation

Set up administrator user

Please enter the details of the administrator user for this installation of Bamboo.

Username*

Password*

Confirm password*

Full name*

Email*

[Finish](#)

Image 10: Bamboo - express installation: Administrator user set up

When finished setting up the instance, Bamboo asks one more thing before finishing the installation. Please fill in the minimum information of the administrator user for Bamboo.

Please click on [Finish](#) in order to complete the installation setup.

Bamboo – express installation

Please wait while Bamboo finalises your setup



Image 11: Bamboo - express installation: Finishing setup

If everything has been correctly set up, Bamboo is now **fully** installed and ready for use! It will redirect you to your **Build dashboard** where you can create your first build plan (see image 12).

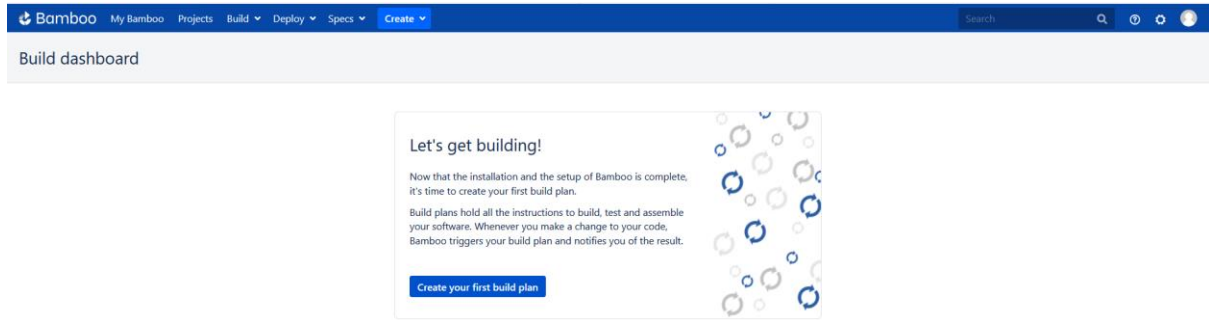
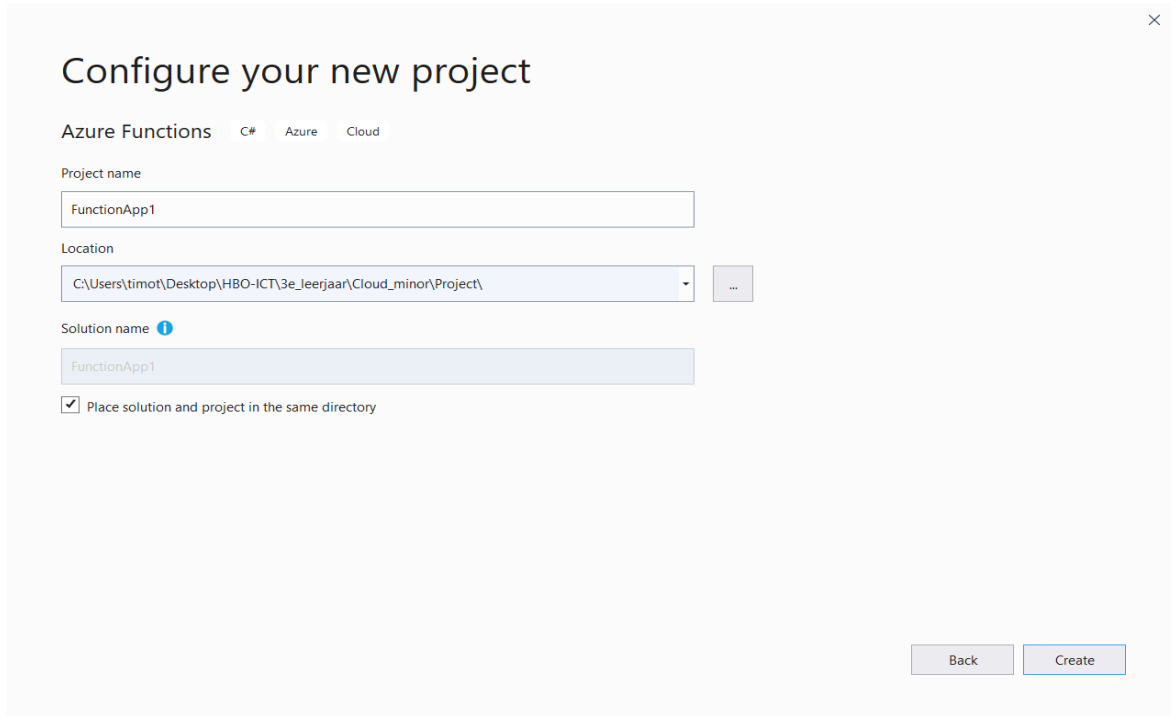


Image 12: Bamboo: Build dashboard

7 Creating an Azure function app using Visual Studio

After configuring Bamboo we have to create an application for Bamboo to work with. In this manual we will create an Azure Function app in Visual Studio. We do this by opening Visual Studio to create a new project (see image 13).



The screenshot shows the 'Configure your new project' dialog box in Visual Studio. The title bar is 'Configure your new project' with a close button (X) in the top right corner. Below the title, there are three tabs: 'Azure Functions' (selected), 'C#', and 'Cloud'. The 'Project name' field contains 'FunctionApp1'. The 'Location' field shows a file explorer path: 'C:\Users\timot\Desktop\HBO-ICT\3e_jeerjaar\Cloud_minor\Project\'. To the right of the location field is a button with three dots. The 'Solution name' field also contains 'FunctionApp1'. Below this, there is a checkbox labeled 'Place solution and project in the same directory' which is checked. At the bottom right, there are two buttons: 'Back' and 'Create'.

Image 13: Creating a new project

Click on 'Create' button. It will redirect you to the screen below (image 14).

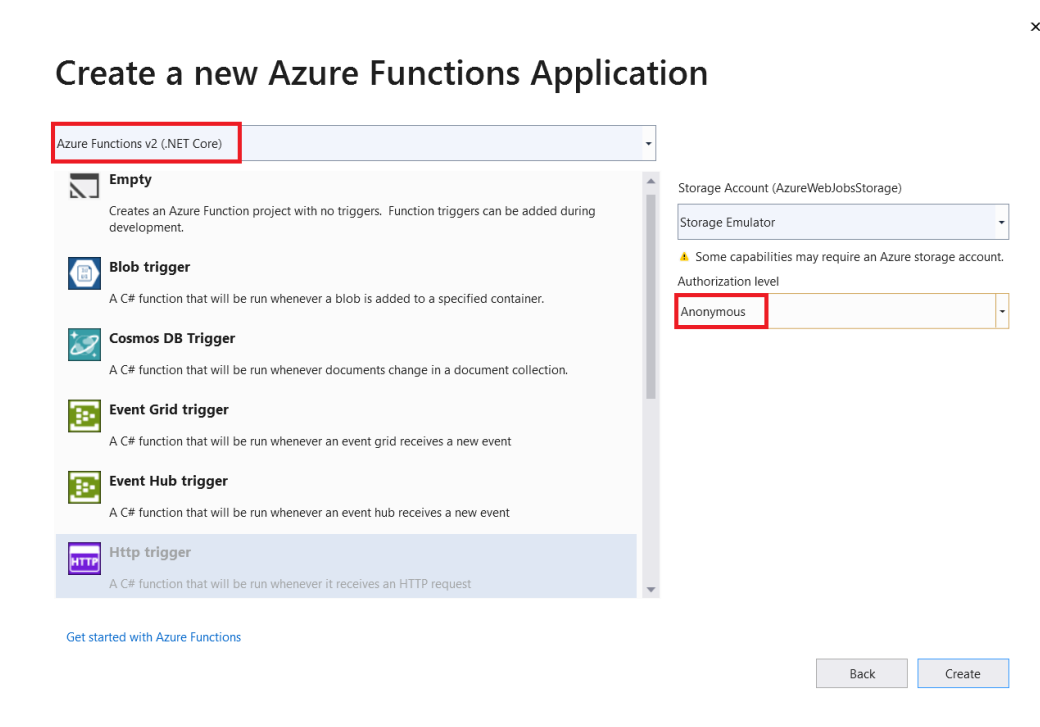


Image 14: Creating a new Azure Functions application

Select the 'Http trigger' function and click 'Create'. The HTTP trigger will allow us to invoke functions with an HTTP request. Make sure you are using Azure Functions v2 (.NET Core). Lastly select Anonymous for the Authorization level.

Now we have a new Azure Functions application. We can add this application to source control by navigating to the bottom right corner and clicking on 'Add to source control' → select 'Git' → click on the 'Team Explorer' tab (image 15). This will give you the option to push the code to remote repositories. In this manual we will use GitHub as our remote repository.

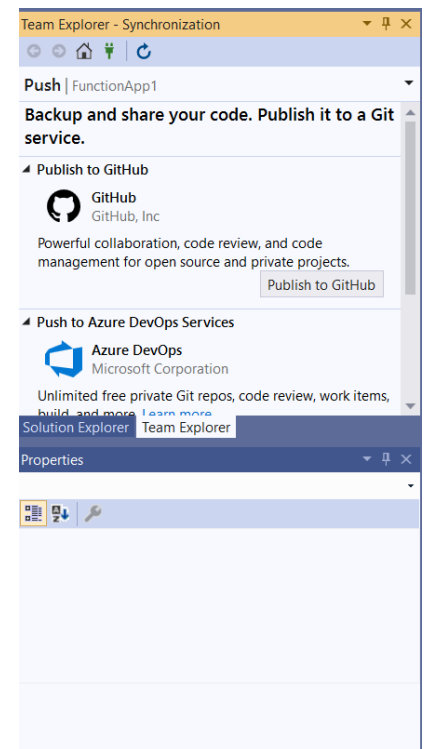


Image 15: Team Explorer

8 Creating a GitHub remote repository

Navigate to [GitHub](#) and log in with your account. On the left side of your GitHub portal, click on to create a new repository. Fill in a repository name and set the access on private, click on the 'Create a repository' button.



You will now be redirected to your new repository. Copy the Git link by clicking the button next to the URL (see image 16).

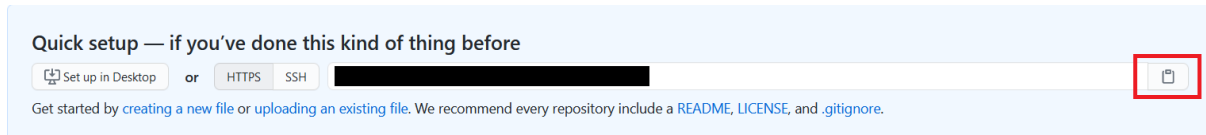


Image 16: GitHub repository setup.

It is now time to push the existing Azure Function application from your local repository (in Visual Studio) to your remote repository in GitHub. To sync the repositories you will need to paste the copied link from image 16 to the 'Push to remote repository' option from the 'Team Explorer' tab in Visual Studio. Now we can publish the local repository into the remote repository by clicking on the 'Publish' button. You can now refresh your remote repository in GitHub to see the synchronization.

9 Creating the Azure function app in Azure

Navigate to the [Azure portal](#) and login with your account. Go to services to create a new 'Function app'. Fill in the mandatory fields. Select '.NET Core' for Runtime Stack with version 3.1 (see image 17).

Create Function App

Basics Hosting Monitoring Tags Review + create

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * ⓘ Azure for Students

Resource Group * ⓘ (New) CloudMinor
[Create new](#)

Instance Details

Function App name * MinorCloudSample2 .azurewebsites.net

Publish * ☒ Code ☐ Docker Container

Runtime stack * .NET Core

Version * 3.1

Region * West Europe

[Review + create](#) < Previous Next : Hosting >

Image 17: Create Function app

Click on the button 'Review and create' and then again on 'Create'. After creating the Function app you will be redirected to page where Azure will execute the deployment. When its finished click on 'Go to resource'.

If everything went successfully, you can look up your newly created Function App in Azure.

9.1 Configuring the Azure Function settings

Navigate to the 'Configuration' page under the settings tab on the left side. Set **SCM_DO_BUILD_DURING_DEPLOYMENT** to **true**. This will execute build steps on your site during deployment, such as npm install. Also set **Website_RUN_FROM_PACKAGE** to **0**. This option allows for deploying the files directly in the D:\home\site\wwwroot directory of your Azure Function application. When this setting is changed to 1 (true) the files are read-only and cannot be updated nor deleted (see image 18).

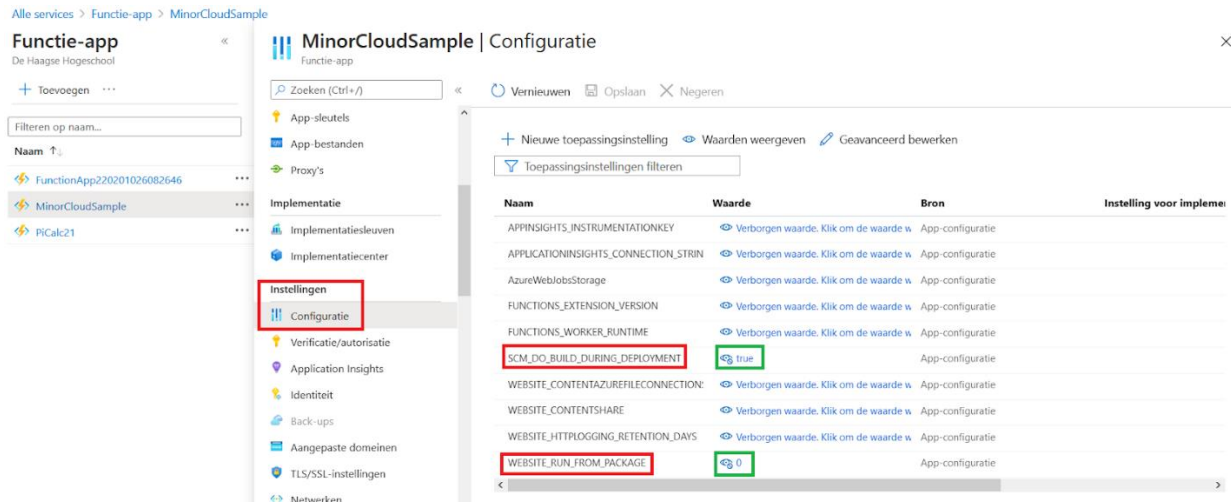


Image 18: The Configuration settings of the Function App

Navigate to the 'Function runtime settings' tab and select `runtime-version ~2`. Now we have an empty Azure Function Application without any functions in Azure and a GitHub repository with an Azure Function app template. In the next step we will use our Bamboo server to deploy the code to Azure.

10 Creating and Configuring a Bamboo build plan

We will create a build plan in Bamboo that defines everything about our continuous integration build process.

Navigate to your Bamboo server and create a new plan. You can find your Bamboo server by typing <http://localhost:8085/> in the URL, your `start-bamboo.sh` file must be running (see chapter 5.2 for more information).

Click on create plan (see image 19).

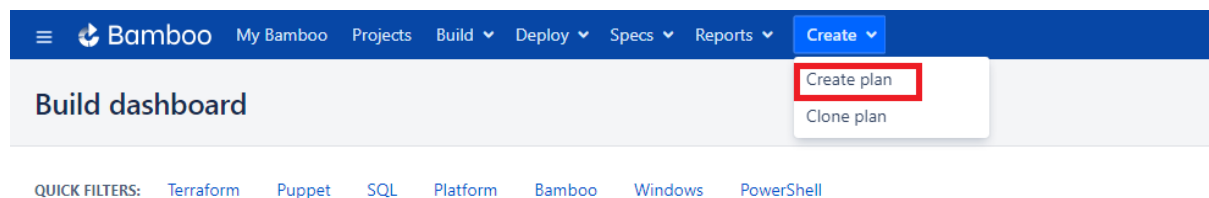


Image 19: Bamboo's navigation menu

Fill in all the necessary details and link the repository to your 'Github repository'. Fill in your credentials and select the repository where you stored the Azure Function app. Test the connection and click the button 'Configure plan' (see image 20.1 and 20.2). Your access token is your GitHub password.

The image shows the 'Create plan' form in Bamboo. The form is titled 'Configure plan' and contains the following fields and options:

- Project and build plan name:**
 - Project name*: Cloud Project
 - Project key*: CP
 - For example AT (for a project named Atlassian)
 - Project description: [Empty text box]
- Plan details:**
 - Plan name*: Azure Function App
 - Plan key*: AFA
 - For example WEB (for a plan named Website)
 - Plan description: [Empty text box]
- Plan access:**
 - Plan access: ☐ Allow all users to view this plan. Applies to new project as well.
- Link repository to new build plan:**
 - Repository host*: ☒ Link new repository
 - GitHub repository: [Dropdown menu]
 - Display name*: Azure functions

Image 20.1: Creating a plan page

Repository host* ☒ Link new repository

GitHub repository ▾

Display name* Azure functions

GitHub repository details

Username* [redacted]

The GitHub user required to access the repositories.

Access token [redacted]

The access token required by the GitHub username.

☐ Using GitHub Enterprise?

Repository [redacted] AzureSample ▾ Load Repositories

Select the repository you want to use for your Plan.

Branch master ▾

Choose a branch you want to check out your code from.

Test connection

✓ Connection successful

Repository access ☒ Allow all users to reuse the configuration of this repository

☐ Only you are allowed to reuse the configuration of this repository

☐ None

Configure plan Cancel

Image 20.2: Creating a plan page

After creating our plan we have to configure the default job. Each plan has a job that will consist of tasks. These tasks will define our plan's behaviour. Select the 'Agent environment' for the build and create the job by clicking on 'Create'.

Now it's time to set up a build plan with a few tasks. Go to actions and select 'Configure plan' from the dropdown list (see image 21).

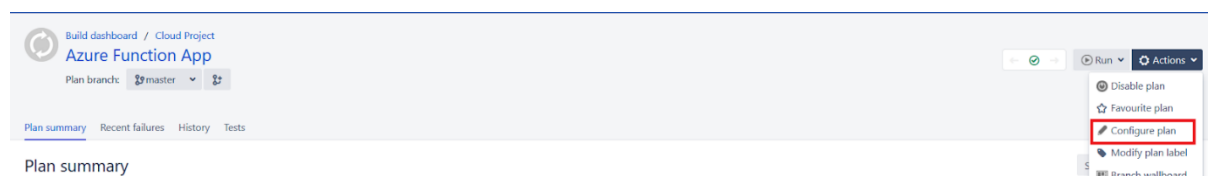


Image 21: The dropdown list to configure the plan

Select the default job as shown in image 22.

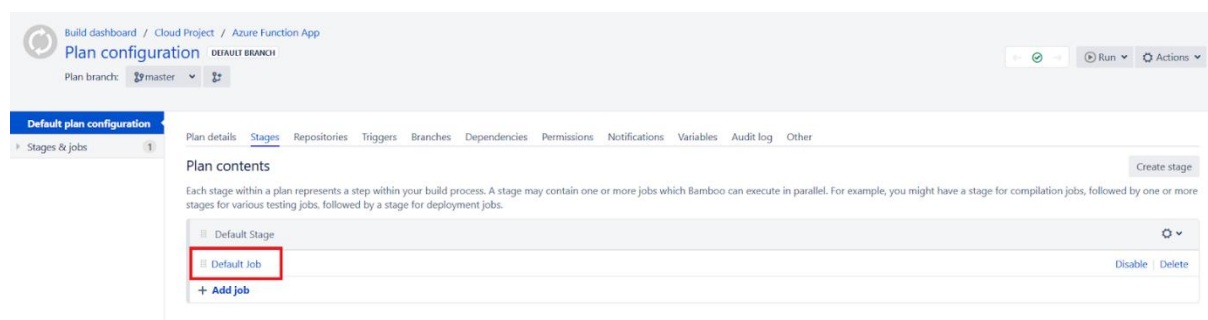


Image 22: Selecting the default job

In the next steps you will learn how to configure the build process tasks for your Azure Function application.

10.1 Task 1: Creating a Source Code Checkout task

This task checks out the current default branch of the code repository in Bamboo. By default, repositories are checked out to the Bamboo working directory.

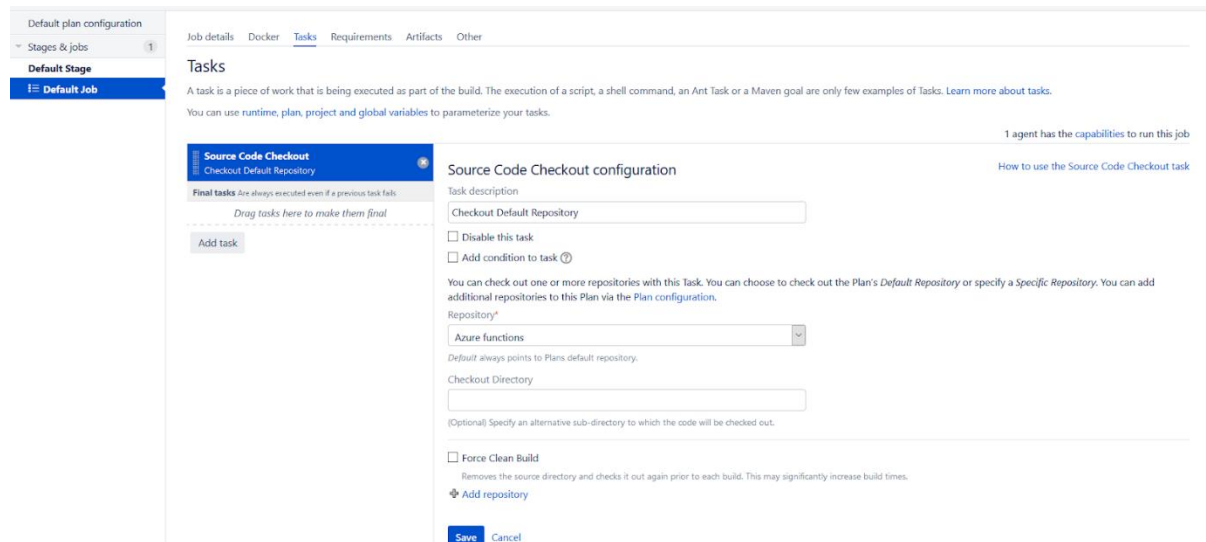


Image 23: Source Code checkout task

10.2 Task 2: Creating a NuGet restore script

Create a new task using the 'Add task' button and select a script. Add a task description and choose your desired interpreter. In this example we have chosen for /bin/sh or cmd.exe. The script location is inline so we can type our command in the script body.

Type in the script body the following:

```
nuget restore "${bamboo.build.working.directory}/FunctionApp1.sln"
```

Change the 'FunctionApp1' part to your solution name if you've changed it when creating the Function App in Visual studio (see image 24). Create the script task using the 'Save' button.

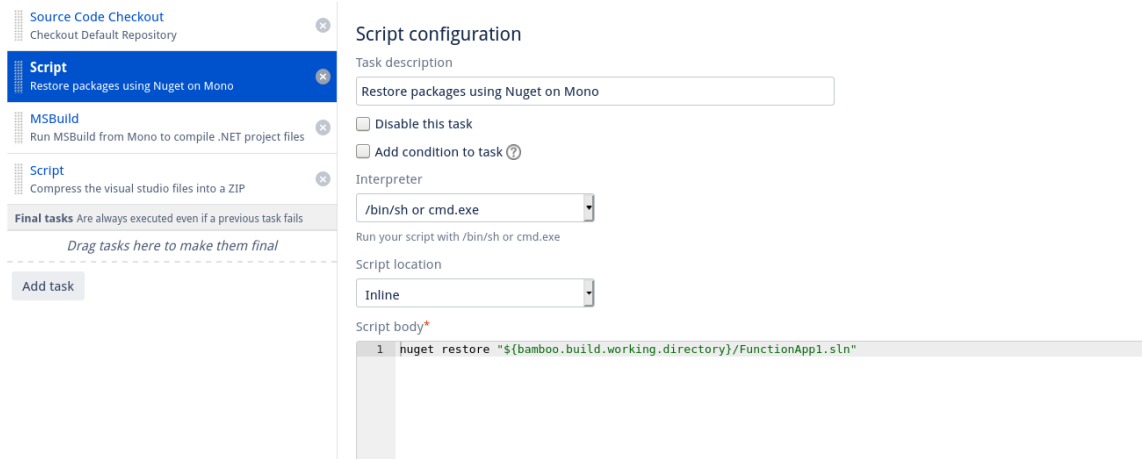


Image 24: Script task

10.3 Task 3: MSBuild task

This task will build our solution on the Bamboo agent in release mode. Add a new task and choose for MSBuild. Fill in the MSBuild configuration and select the latest MSBuild executable or ‘Add a new executable’. Furthermore you have to change the project file name to the correct one on your local computer.

In CentOS MSBuild is integrated within Mono since version 5.0.0. It is the same MSBuild that is used on .NET in Windows. You can use it with the ‘msbuild’ command in the terminal shown in image 25.

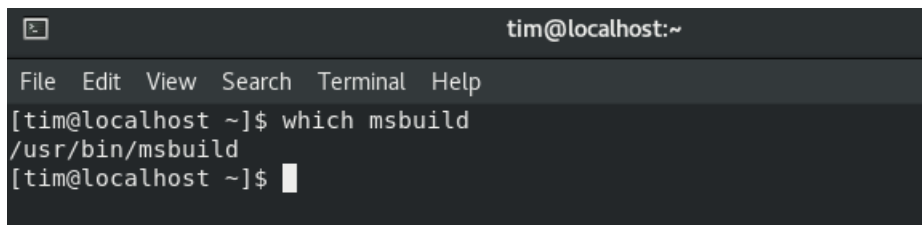


image 25: msbuild command

The output is your path. Copy the path and paste it in the path field. Click on ‘Add’ to add the executable.

After adding the executable we have to use the value below for the options field:

```
/property:Configuration=Release /p:DeployOnBuild=true /p:PublishProfile=FolderProfile
```

Save the task by using the ‘Save’ button.

10.4 Task 4: Script ZIP file task

We will create a zip file task to zip our solution. Firstly we will create a folder called ‘artifacts’ in the Documents folder. You can do this by navigating to the File Explorer or by typing in the commando `mkdir artifacts` (be sure to navigate to the documents folder with `cd Documents`).

Create a new script task with the ‘Add task’ button and choose ‘/bin/sh or cmd.exe’ as your interpreter. Use the following script body and change the source and destination locations:

```
zip -r function.zip .
```

```
mv function.zip <path to artifacts folder>
```

Script configuration

Task description
Compress the visual studio files into a ZIP

☐ Disable this task
☐ Add condition to task ?

Interpreter
/bin/sh or cmd.exe

Run your script with /bin/sh or cmd.exe

Script location
Inline

Script body*

```
1 zip -r function.zip .
2 mv function.zip /home/tim/Documents/artifacts
```

Image 26: the Script task parameters

Create the script task.

10.5 Creating an artifact

Navigate to the artifacts tab and create a new artifact with the settings shown in image 27.

Edit artifact definition

Name* Function1 artifact
If the artifact is shared, the name must be unique within the plan

Location
Specify the directory (relative path) to find your artifact. e.g. target

Copy pattern* */**
Specify the name (or [Ant file copy pattern](#)) of the artifact(s) you want to keep. e.g. **/*.jar

☒ Shared
Make the artifact available to be used in other builds and deployments. If you subscribe to an unavailable shared artifact, the build will fail.

☒ Required
Build fails if the artifact cannot be published.

Save Cancel

Image 27: the Artifact parameters

Click on 'Save' to save the artifact.

Now we can run the plan by clicking on the run dropdown list → ‘Run plan’. If the build plan has run without errors it will build successfully. The artifact has been made when executing the plan. We can publish this artifact to Azure with a deployment project.

10.6 Configuring a Bamboo deployment project

We have reached the last step of configuration. This step will pertain the deployment of the Azure Function app. Go to the ‘Deploy’ tab and navigate to ‘All deployment projects’. Create a new deployment project. Fill in a name for your project and select your desired Build plan and branch. Click on ‘Create deployment project’ (see image 28).

Create deployment project [How deployments work](#)

A deployment project defines which build plan you get your artifacts from, and contains the environments you want to deploy to.

Deployment project details

Name*

Description

Access ☐ Allow all users to view this deployment project

Link to build plan [How deployment releases work](#)

Shared artifacts of the selected plan will be bundled into releases. Releases will be deployed to the environments.

Build plan*

Start typing the plan name or use the down arrow to select a plan. The selected plan will be used as the source for artifacts that will be deployed as a release.

☒ Use the main plan branch

Currently

☐ Use a custom plan branch

Create deployment project [Cancel](#)

Image 28: Deployment project parameters

Create a new environment with ‘Add environment’ and give it a corresponding name. Choose for ‘Agent environment’. Click on ‘Create’.

You will be redirected to the set up task page. You will notice that Bamboo already created 2 tasks. ‘Clean working directory task’ and ‘Artifact download’ task. Navigate to the ‘Artifact download’ task and select the correct artifact from the build plan in the previous step. Click on ‘Save’.

Now we have to add a new script task (see image 29). In this script task we will connect to Azure and pass the zip deployment using /bin/sh or cmd.exe command. Choose ‘/bin/sh or cmd.exe’ as the Interpreter and use the following script:

```
az login -u '<username>' -p '<password>' --tenant <tenant-id>
```

```
az webapp deployment source config-zip --resource-group "<resource-group name>" --name "<function app name>" --src <path to the zip file (function.zip)>
```

The <username> and <password> credentials are from your Azure account. You can find your tenant-id by logging into your Azure portal and opening up the Azure CLI. Type in the following command:

```
echo $tenantId = (Get-AzContext).Tenant.Id
```

The output will be your tenant-id.

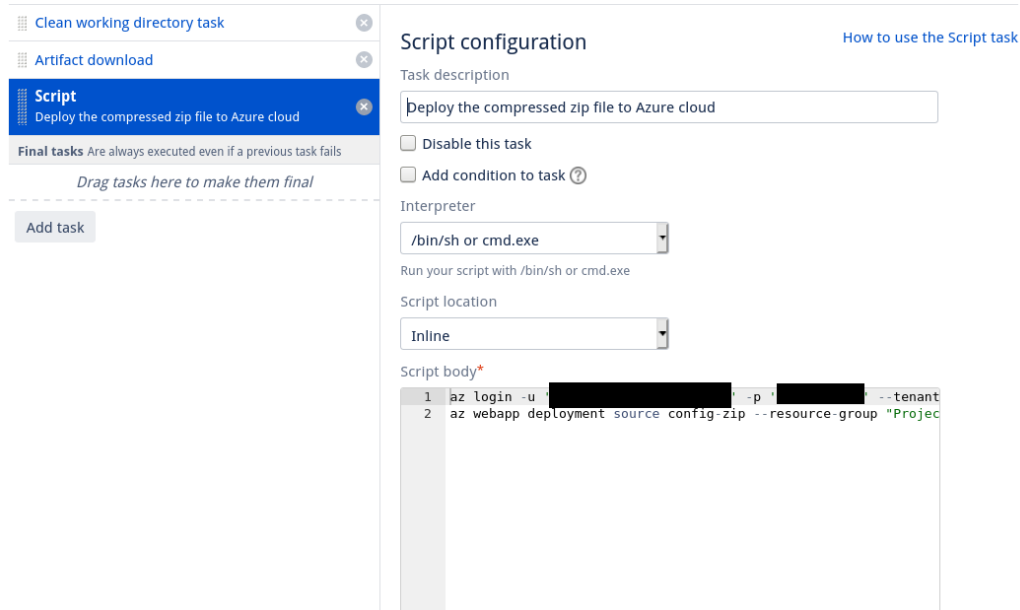


Image 29: Script configuration

Now it is time to deploy our project. Go to your deployment projects and select the correct deployment. Navigate to 'Deploy' and select your project (see image 30).

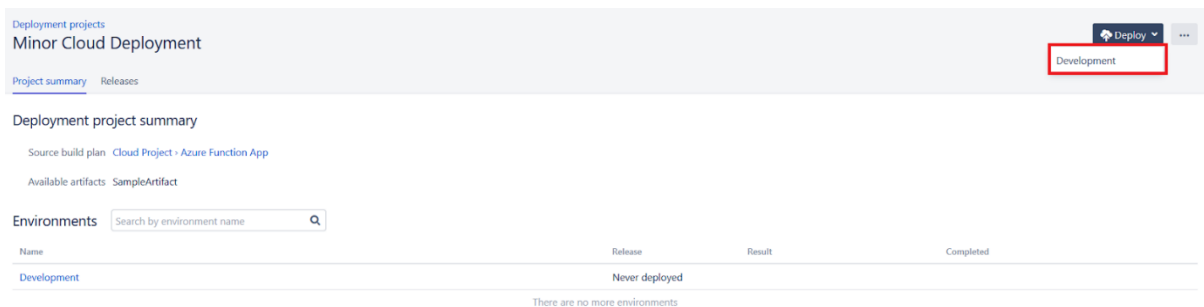


Image 30: Development build

Review your deployment and select your latest Build result. Give your release a name (image 31).

Deployment preview: Development

[How deployment releases work](#)

Select a release to deploy

☒ Create new release from build result

Build result* 🟢 #6 37 minutes ago ⓘ

Release* release-1 ⓘ

☐ Promote existing release to this environment - There are no existing releases

Artifacts provided by
🟢 #6
This is your first release in this deployment project.
Therefore, we cannot calculate the number of issues and commits in this release.
Execution details
[3 tasks](#)

Start deployment

Cancel

Image 31: Deployment preview

If the deployment has been successful, you will see something like image 32.

Deployment projects / Minor Cloud Deployment / Environment: Development
Deployment: release-1 on Development

Deploy ...

🟢 Success: Deployment of release-1 to Development

Details

Release: release-1

🔗 master

Trigger: Manual run by Tim

Completed: 14 Sep 2020 10:33 PM

Duration: 3 seconds

On agent: Default Agent

Status: success

The deployment generated 35 lines of output. [Download or View full deployment log](#)

```

14-sep-2020 22:33:48 Build Deployment of 'release-1' on 'Development' started building on agent Default Agent, bamboo version: 7.1.2
14-sep-2020 22:33:48 Build working directory is C:\Users\Tim\Desktop\Bamboo\xml-data\build-dir\1212417-1310721
14-sep-2020 22:33:48 Executing build Deployment of 'release-1' on 'Development'
14-sep-2020 22:33:48 Starting task 'Clean working directory task' of type 'com.atlassian.bamboo.plugins.bamboo-artifact-downloader-plugin:cleanWorkingDirectoryTask'
14-sep-2020 22:33:48 Cleaning working directory 'C:\Users\Tim\Desktop\Bamboo\xml-data\build-dir\1212417-1310721'
14-sep-2020 22:33:48 Finished task 'Clean working directory task' with result: Success
14-sep-2020 22:33:48 Starting task 'Download release contents' of type 'com.atlassian.bamboo.plugins.bamboo-artifact-downloader-plugin:artifactDownloadTask'
14-sep-2020 22:33:48 Preparing to download plan result CP-AFA-d artifacts: Non required shared artifacts: [SampleArtifact], pattern: [/**]
14-sep-2020 22:33:48 Artifact [SampleArtifact] downloaded successfully in 943.0 ms to working directory
14-sep-2020 22:33:48 Finished task 'Download release contents' with result: Success
14-sep-2020 22:33:49 Substituting variable: ${bamboo.build-working-directory} with C:\Users\Tim\Desktop\Bamboo\xml-data\build-dir\1212417-1310721
14-sep-2020 22:33:49 Starting task 'Download from Azure and deploy to Development' of type 'com.atlassian.bamboo.plugins.scripttask:task.builder.script'
14-sep-2020 22:33:49 Beginning to execute external process for Build 'Deployment of "release-1" on "Development"'
... running command line:
... cmd C:\Users\Tim\Desktop\Bamboo\temp\1212417-1310721-scriptBuildTask-599295409071593719.ps1
... using extra environment variables:
bamboo_planRepository_1_branchName=master
bamboo_deploy_environment=Development
bamboo_deploy_projectRef=Cloud deployment
bamboo_planRepository_1_branchName=master

```

Image 32: Deployment output

10.7 Configuring triggers

Triggering in Bamboo allows plan builds to be started automatically. You can choose between triggering a build if the code has changed, based on a schedule or depending on the outcome of other plans. Note that plans can also be started manually independent of triggers like we did in section 9.5.

Go to your desired build plan and select 'Default plan configuration'. Go to triggers and add a trigger. There are four triggers each with a different function:

- Remote trigger: Triggers the build when changers are committed in the repository.
- Repository polling: Bamboo polls the source repository and builds when new changed are found.
- Scheduled: Run according to a schedule.
- Single daily build: Runs once a day.

In this example we will choose for 'Repository polling'. Select the repository on which the trigger should be applied to. Select the desired polling frequency in seconds. This is how often (in seconds) Bamboo should check for changes in the repository (see image 33). Click on save trigger.

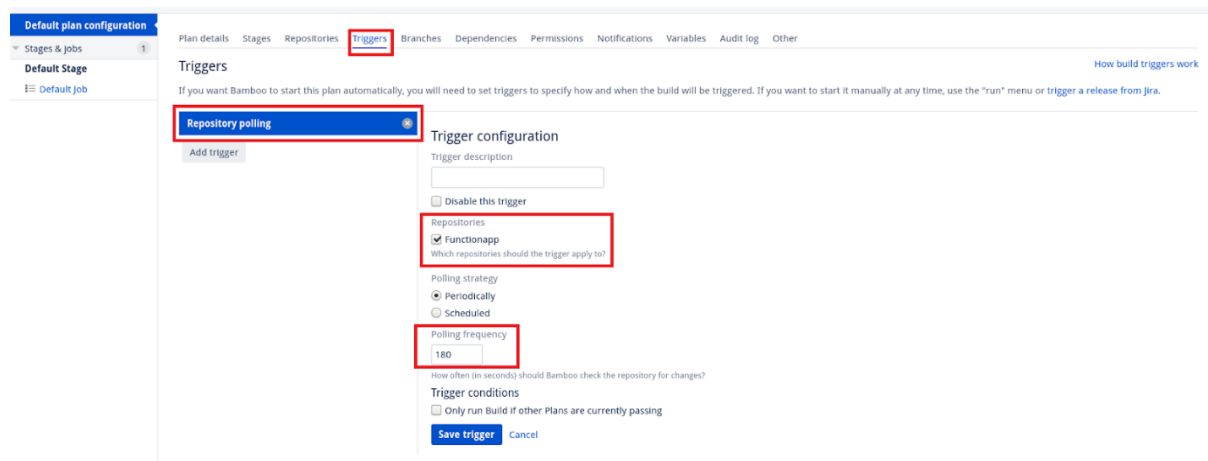


Image 33: The trigger configuration

10.8 Adding the trigger to your deployment plan

Go to your deployment project and click on 'triggers'.

What you want to deploy

Source build plan [Centos8 Cloud Minor](#) > [Azure Cloud](#)

Available artifacts [Function1 artifact](#)

[Edit build plan](#) [Release versioning](#) [Project permissions](#) [Bamboo Specs repositories](#) [Audit log](#)

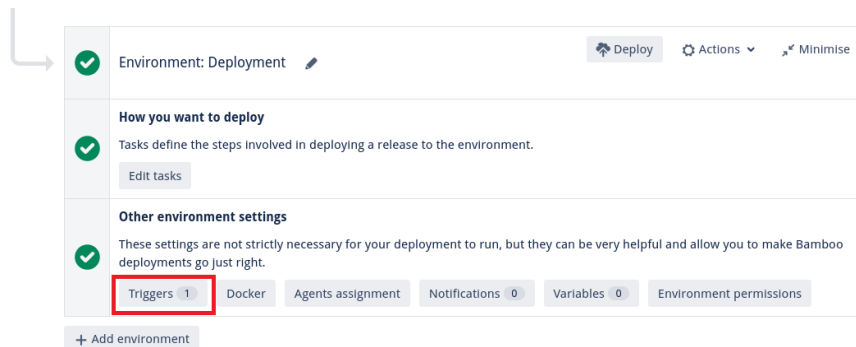


Image 34: Adding a trigger

Clicking on the Trigger button in image 34 will bring an overview of triggers that are currently active in that deployment plan. Here you can add, modify or delete a trigger.

Add a trigger and select your desired trigger. Bamboo gives you 3 options:

- After succesful build plan
Deployment is started after a plan is successfully build
- After succesful stage
Deployment is started after a stage is successfully build
- Scheduled
Run according to schedule

In this tutorial we will choose the first option.

Edit triggers: Deployment

Set triggers to specify how and when the deployment will be triggered automatically. When a deployment is automatically triggered, a new release is created from the latest successful build result of the linked plan.
If you want to deploy to this environment manually, use the "Deploy" action at any time.

After successful build plan

Add trigger

Trigger configuration

Trigger description

☐ Disable this trigger

Save trigger

Cancel

Back to deployment project

Image 35: Edit trigger

Add a trigger description and save the trigger (image 35). It is now currently active as it periodically checks for successful build plans.

10.9 Test the Azure Function app

Go to Azure portal and look up your Function App. On the left side search for "Functions" and click on it. If you see a function then you have successfully created it with http trigger which you deployed using the Bamboo CI/CD server.

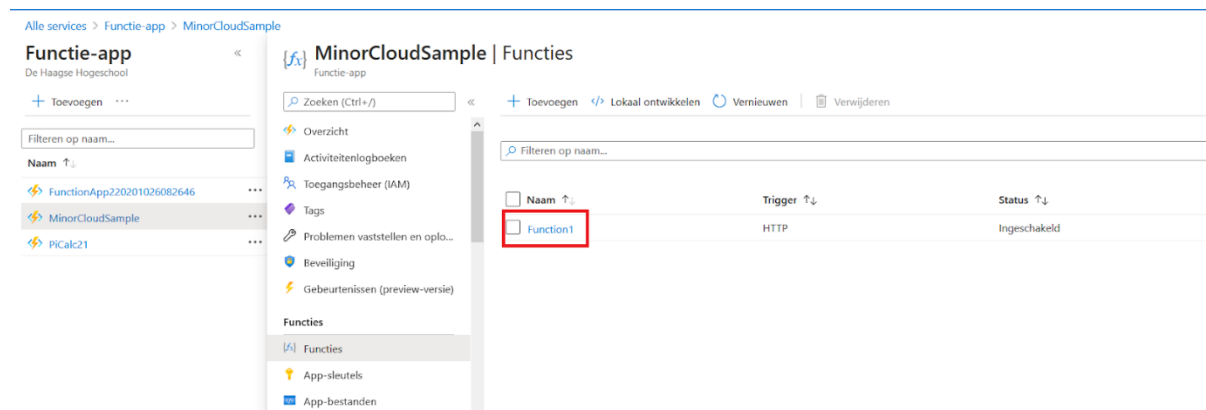


Image 36: Functions overview

Click on the Function and retrieve the Function Url using the "Get Function Url" tab.

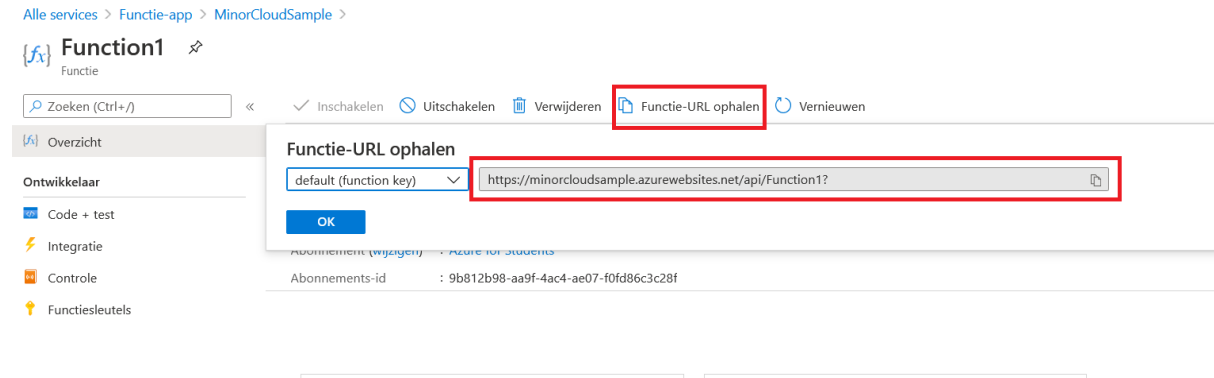


Image 37: The Function app URL.

Paste it on your browser and check if the function works.

If it works, congratulations! You have successfully configured Bamboo. You can now add more plans, tasks, triggers as you wish.