DevOps & Application Stores CA 50%

2018

Computing with Multimedia

Aoife Sayers T00170881

Environment: Production Environment

http://aws.link AWS Windows Server VM

**Production IIS Server**

Environment: Staging Environment

http://azure.link Azure Windows Server VM

**Staging IIS Server**

Nuget Packages

Results Published

CD: Octopus Deploy Server

http://localhost:8008

**Octopus Deploy Server**

Metrics: Sonar Server

http://localhost:9000

**SonarQube**

Octopus Deploy: Promote Release

Octopus Deploy: Deploy Release

Octopus Deploy: Create Release

Octopus Deploy: Create & Push Packages

NUnit

MSBuild

SonarQube Runner

CI: Teamcity CI Server

http://localhost:8111

**Teamcity**

Push

Pull

Commit

Checkout

VCS: Distributed VCS

**GitHub**

VCS: Local VCS

**Git**

IDE: Visual Studio

Project Name: **DevopsCalc**

Release Promoted

Release Deployed

Octopus Tentacle Manager/Agent

Octopus Tentacle Manager/Agent

Octopus Tentacle Manager/Agent

Table of Contents

[1. Software Required: 5](#_Toc510531690)

[2. Calculator ASP Application 6](#_Toc510531691)

[Code for Calculate.cs 6](#_Toc510531692)

[Snippet of Default.aspx 6](#_Toc510531693)

[TestCalculate.cs 7](#_Toc510531694)

[Nuget Packages Installed 8](#_Toc510531695)

[3. Version Control System: Git 9](#_Toc510531696)

[4. Distributed Version Control System: GitHub 12](#_Toc510531697)

[5. CI Server: Teamcity 14](#_Toc510531698)

[5.1 About Teamcity 14](#_Toc510531699)

[5.2 MSBuild 17](#_Toc510531700)

[5.1 Installing Plugins for Teamcity: Octopus Deploy 18](#_Toc510531701)

[5.2 Nunit 20](#_Toc510531702)

[5.3 Octopus Deploy: Create & Push packages 22](#_Toc510531703)

[5.4 Octopus Deploy 23](#_Toc510531704)

[Octopus Deploy 23](#_Toc510531705)

[Machines Roles 23](#_Toc510531706)

[Environments 23](#_Toc510531707)

[Deployment Targets 23](#_Toc510531708)

[Tentacle 25](#_Toc510531709)

[Lifecycle 26](#_Toc510531710)

[Channels 26](#_Toc510531711)

[Variables 26](#_Toc510531712)

[Deployment Process/Step 27](#_Toc510531713)

[Deploy an IIS website process 28](#_Toc510531714)

[5.1 Octopus Deploy: Create release 30](#_Toc510531715)

[5.2 Octopus Deploy: Deploy release 30](#_Toc510531716)

[5.3 Promote release 30](#_Toc510531717)

[5.4 Email & Visual Studio IDE notifiers 30](#_Toc510531718)

[5.5 Tutorials Used 30](#_Toc510531719)

[6. Issues 30](#_Toc510531720)

[References 33](#_Toc510531721)

# Software Required:

* SQL Server Express
* Octopus Deploy Server Manager
* Octopus Deploy Tentacle Manager
* Teamcity Server
* Teamcity Build Agent
* Octopus Deploy Plugin - install zip via web UI (Teamcity zip file)
* SonarQube Server
* SonarQube Scanner for MSBuild
* SonarQube Plugin for Teamcity
* Servers/VMs
  + Windows Server 2016 - AWS & Azure
    - Network Security Groups/Rules / Firewall- Inbound
      * RDP - 3389
      * HTTP - 80
      * HTTPS - 443
      * Listening Tentacle - 10933
      * Polling Tentacle - 10943
* Nuget Package Explorer (If you’re not using Teamcity Octopack to pack solutions)
* Visual Studio
* Microsoft Build Tools/MSBuild
* Nuget Packages
  + Octopack
  + NUnit
  + NUnit Console
  + NUnit Console Runner

# Video of Project working

A full video of the project working from a modification to the code, git commit, Github update and a build kicking off in Teamcity automating the Octopus Deploy deployment etc. can be viewed here.

<https://youtu.be/sMtitmGhLDg>

# Calculator ASP Application

I followed this tutorial <https://www.csharpens.com/asp-net/simple-calculator-asp-net-1080/> to develop a simple ASP.NET calculator.

## Code for Calculate.cs

The code below in Calculate contains functions for add, subtract, multiply, divide and percentage of 2 values

namespace ASP\_1

{

public class Calculate

{

public int add(int Value1, int Value2)

{

return Value1 + Value2;

}

public int subtract(int Value1, int Value2)

{

return Value1 - Value2;

}

public int multiply(int Value1, int Value2)

{

return Value1 \* Value2;

}

public double divide(int Value1, int Value2)

{

return Value1 / Value2;

}

public string percentage(int Value1, int Value2)

{

Value1 = Value1 \* 100;

return divide(Value1, Value2) + "%";

}

}

}

## Snippet of Default.aspx

The code below contains the layout for the calculator – buttons etc.

<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="ASP\_1.Calculator" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">

<head runat="server">

<title>Sajjad's Calculator</title>

<link href="Calculator.css" rel="stylesheet" />

</head>

<body>

<form id="form1" runat="server">

<div class="calculator-holder">

<table class="calculator" id="calc">

<tr>

<td colspan="4" class="calc\_td\_result">

<input type="text" readonly="readonly" runat="server" name="calc\_result" id="calc\_result" class="calc\_result" />

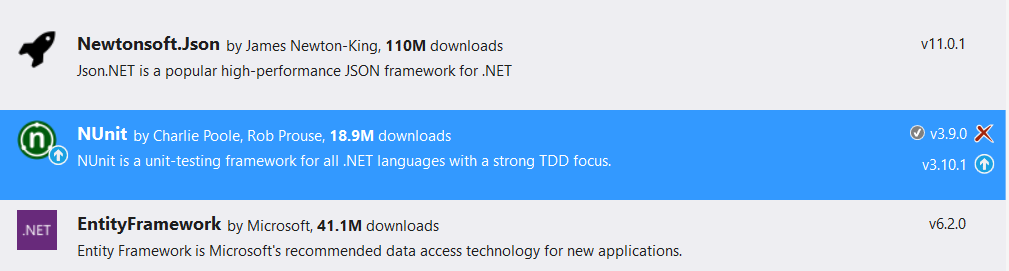
</td>

</tr>

## TestCalculate.cs

A class using NUnit to test the functions in Calculate.cs. **[TextFixture]** is an attribute that marks a class that contains tests and optional methods for setup and teardown. **[TestCase]** is an attribute for marking a method. (NUnit, 2016). NUnit was installed from Nuget Packages in Visual Studio.

Visual Studio > Project > Manage Nuget Packages



A new object is created of the Calculate class and each method is tested (add, subtract, multiply etc.) and Assert.AreEqual() tests to see if the statements are true or Assert.AreNotEqual() to prove the tests are false.

using NUnit.Framework;

namespace ASP2

{

//Class that carries out NUnit Tests on Calculate.cs

[TestFixture]

public class TestCalculate

{

[TestCase]

public void testAdd()

{

ASP\_1.Calculate f = new ASP\_1.Calculate();

Assert.AreEqual(f.add(5, 5), 10);

Assert.AreEqual(f.add(-5, -3), - 8);

Assert.AreNotEqual(f.add(2, 2), 2);

}

[TestCase]

public void testSubtract()

{

ASP\_1.Calculate f = new ASP\_1.Calculate();

Assert.AreEqual(f.subtract(5, 5), 0);

Assert.AreEqual(f.subtract(-5, -3), -2);

Assert.AreNotEqual(f.subtract(2, 2), 2);

}

[TestCase]

public void testMultiply()

{

ASP\_1.Calculate f = new ASP\_1.Calculate();

Assert.AreEqual(f.multiply(5, 5), 25);

Assert.AreEqual(f.multiply(-5, -3), 15);

Assert.AreNotEqual(f.multiply(2, 2), 5);

}

[TestCase]

public void testDivide()

{

ASP\_1.Calculate f = new ASP\_1.Calculate();

Assert.AreEqual(f.divide(6, 2), 3);

Assert.AreEqual(f.divide(-15, -3), 5);

Assert.AreNotEqual(f.divide(2, 2), 2);

}

}

}

## Nuget Packages Installed

**NUnit** – unit testing framework for .NET languages

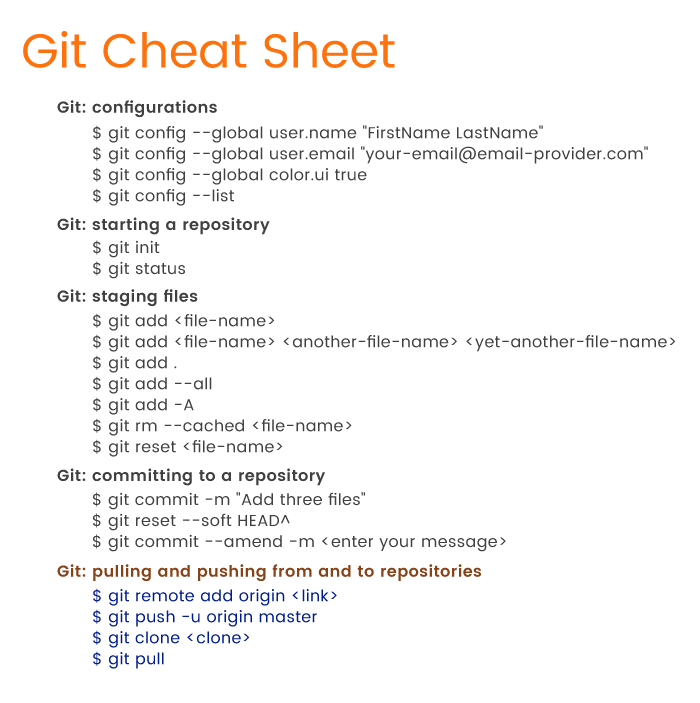
**NUnit.Console** & NUnit.ConsoleRunner – console runner for NUnit 3 unit testing Framework. Used within CI Server

**Octopack** – Nuget package for packaging .NET applications from your CI server. When Octopack is enabled on the MSBuild process in a .NET solution, Octopack packs your project into nupkg artifacts for .NET projects (Deploy, 2018).

# Version Control System: Git

Git was used as a version control system. Git is a free open source version control system (Git, 2018). Git Bash was used as the CLI for using Git.

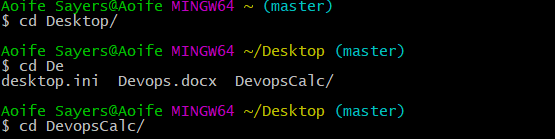
Some of the commands I used were:



Git Cheat Sheet (A., 2018)

The git repo was automatically initialised by Visual Studio. To create a new repository in git, the developer should change directory to the folder to be initialised as a git repository. The command “git init” should then be issued to initialise a git repository.

To commit to the Visual Studio project I changed directory “cd” in to Desktop/DevopsCalc/

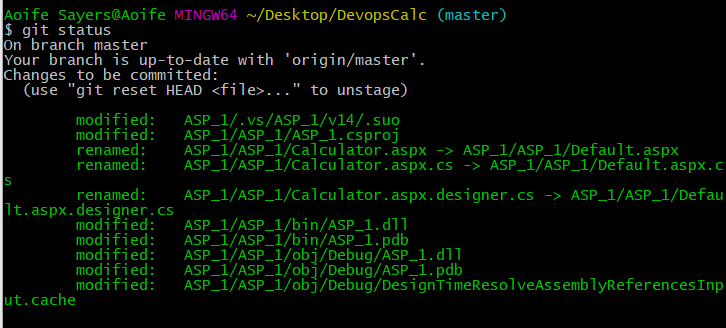


The “git status” command to list the untracked changes of the project since the last commit. Red text denotes unstaged files via Git. Green files denote staged files in Git

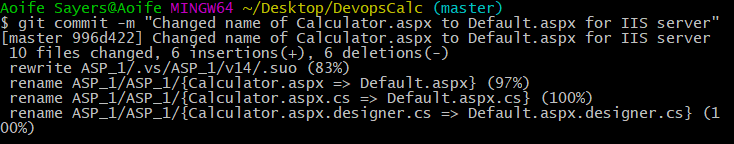
The “git add .” command was used to stage new and modified files via git bash.



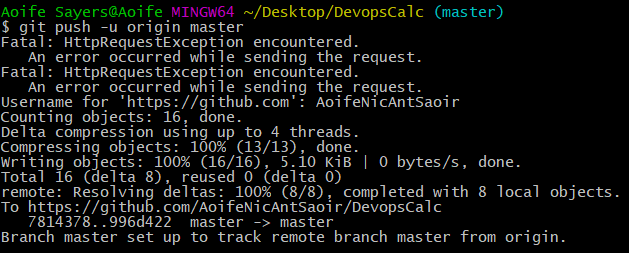
Git status was used again to verify the changes are now staged in Git. The files previously unstaged are now staged as they are coloured green.



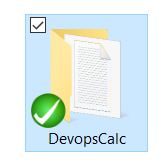
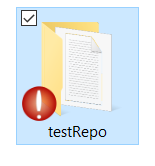
The command “git commit -m “commit message” is now tracked as a revision in source control with a unique ID (SHA or hash) and message of the changes made of the previous revision.



The local commit is now pushed up to a hosted Git repository on Github.com using the “git push -u origin master” command.



Tortoise Git was used to verify in a GUI manner to verify the correct files were staged. Tortise Git denotes all modifications have been committed and tracked in version control using a “tick” as seen below. Alternatively, repositories with modification not committed in version control are denoted by a red “!”.

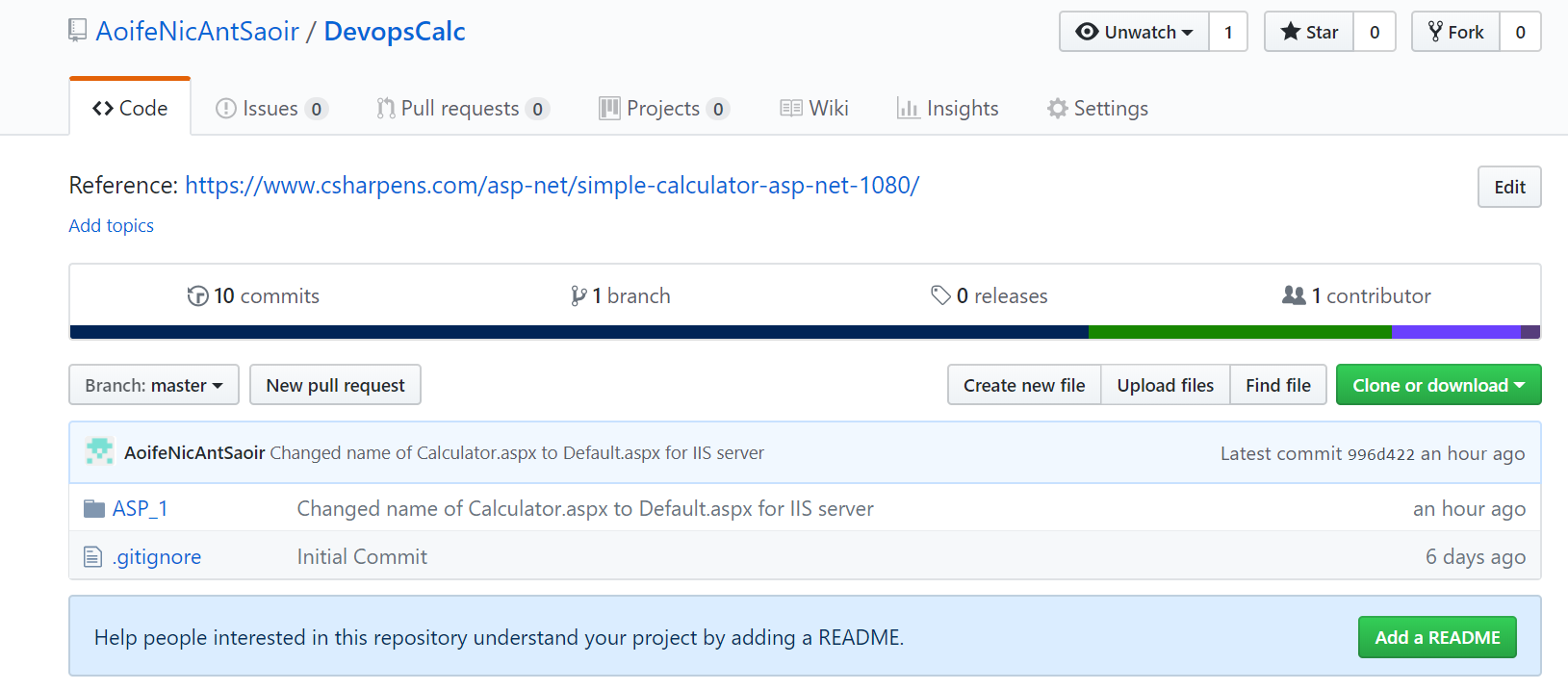
Tutorials Used

I used the Git documentation as a tutorial for Git located here: <https://git-scm.com/docs/gittutorial>

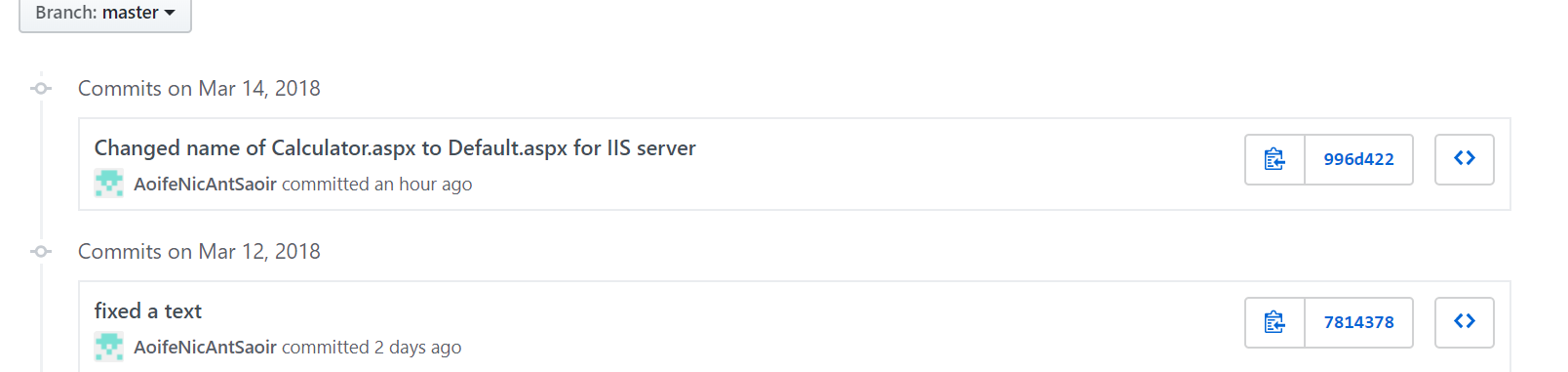
# Distributed Version Control System: GitHub

Github is a distributed and hosted Git version control system. Github encourages the sharing of code and working on building of software with other developers worldwide. Github may be used as a repository in which a CI Server will fetch code from for building and testing etc. applications. Teamcity, Microsoft Team Services and Jenkins use Github to fetch code from for their build process.

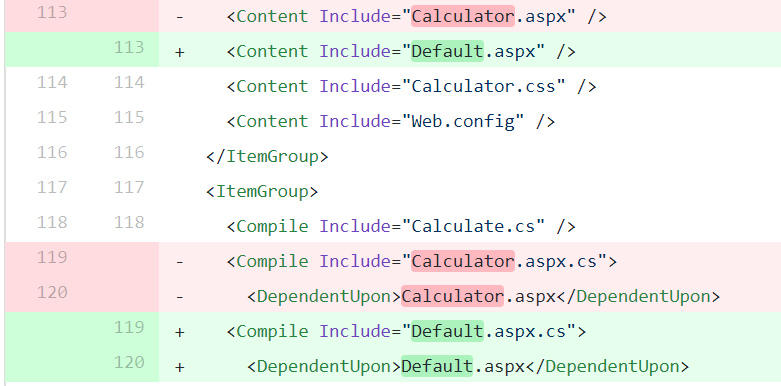
The Github repository for this project is located at the following URL: <https://github.com/AoifeNicAntSaoir/DevopsCalc>



The Commits made to Github can be seen in the image below. The latest commit on March 14 2018 corresponds to the commit made on Git and then pushed to Github as seen below.



The latest commit is compared with the most recent commit when you click on the individual commits. The red denotes the removed or altered code and the green denotes the most recent modifications/additions to the code. For instance in the example below, the .aspx file was renamed from Calculator.aspx to Default.aspx.



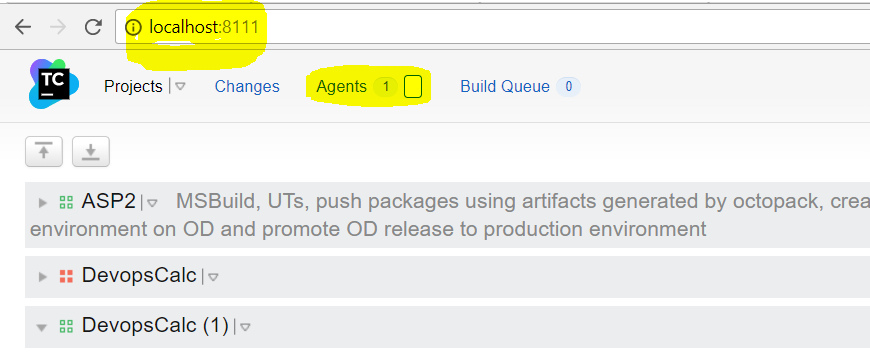
# CI Server: Teamcity

## About Teamcity

Teamcity is a Continuous Integration Server from Jetbrains. Build Agents fetch code from the most recent commits of Version Control Systems. Build Configuration pipelines can be configured from a range of build steps. Other build steps can be added to Teamcity using plugins for additional functionality (Teamcity, 2018).

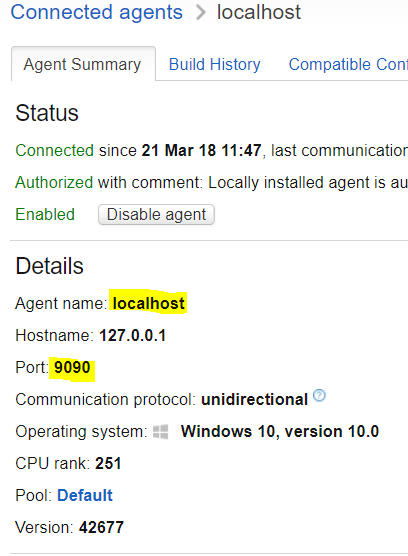
Teamcity is running on localhost:8111 as a Windows Service. Teamcity is running on port 8111 as it was originally running on port 80 and clashing with the Octopus Deploy Server. There is one Build agent configured on localhost:9090

Teamcity Server running on localhost:8111 on a web interface.

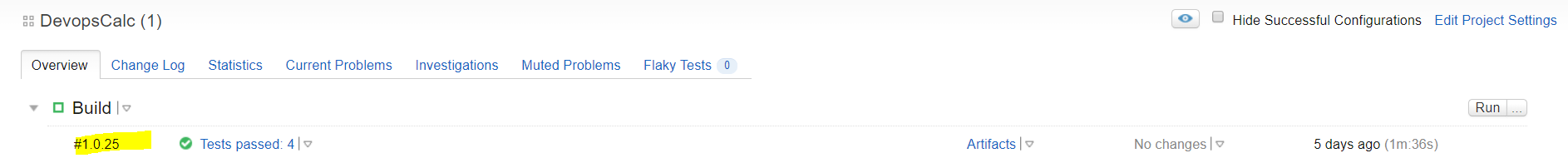
A

**Agents**

Teamcity Agent running on localhost:9090. Teamcity Agents are used to carry out build processes.



Teamcity project created called DevopsCalc(1)



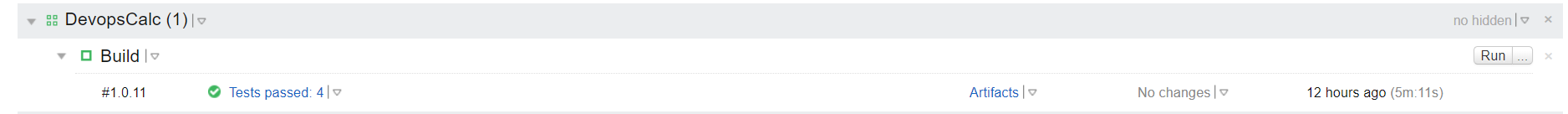
**Version Control System**



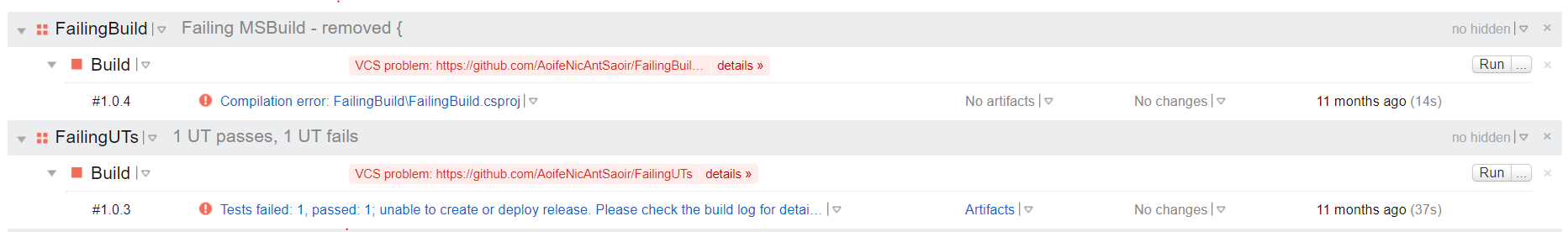
The VCS Version Control System settings are pointing towards a Git Server, GitHub to fetch the most recent commit for the build pipeline in Teamcity. A VCS trigger is setup to automatically kick off a build each time a new commit is checked into the Version Control System (VCS)

**Build**

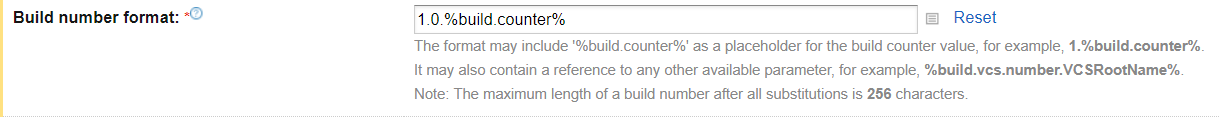
A Successful Build is shown as followed below with a green tick mark and process exit 0 in the build log



An Unsuccessful Build is denoted by a red ‘x’ and process exit 1 in the build log



The Build Number in the format of Major.Minor.Patch

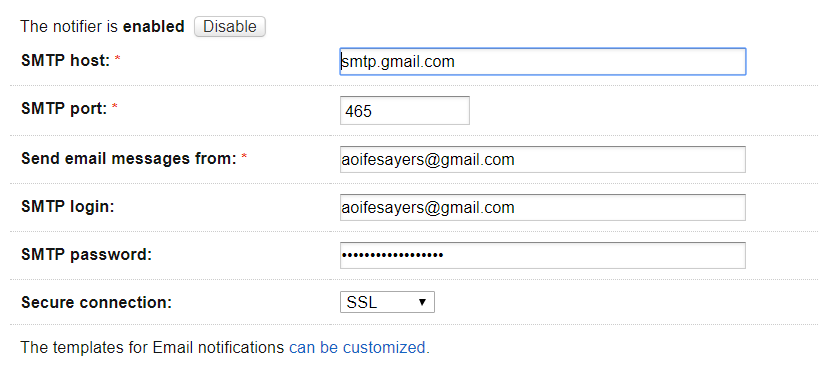




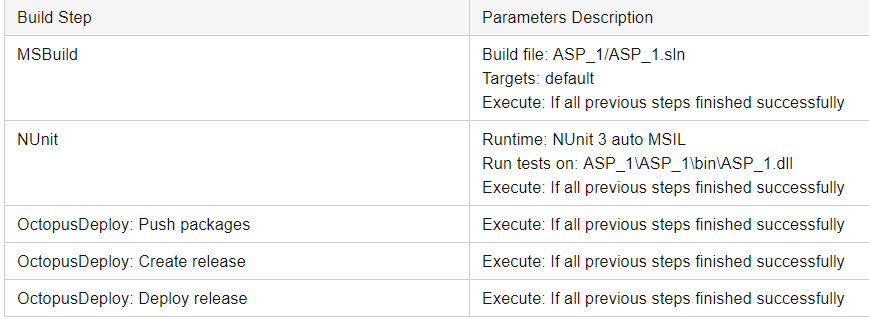
**Build Notifications**

Teamcity allows for Email notifiers on success of a build. I pointed the email notifier to the SMTP Gmail host. On Gmail I had to change settings for IMAP/POP3 accounts to enable IMAP to be in receipt of messages from Teamcity server.

See tutorial <https://www.wpsitecare.com/gmail-smtp-settings/>





**Build Steps** 

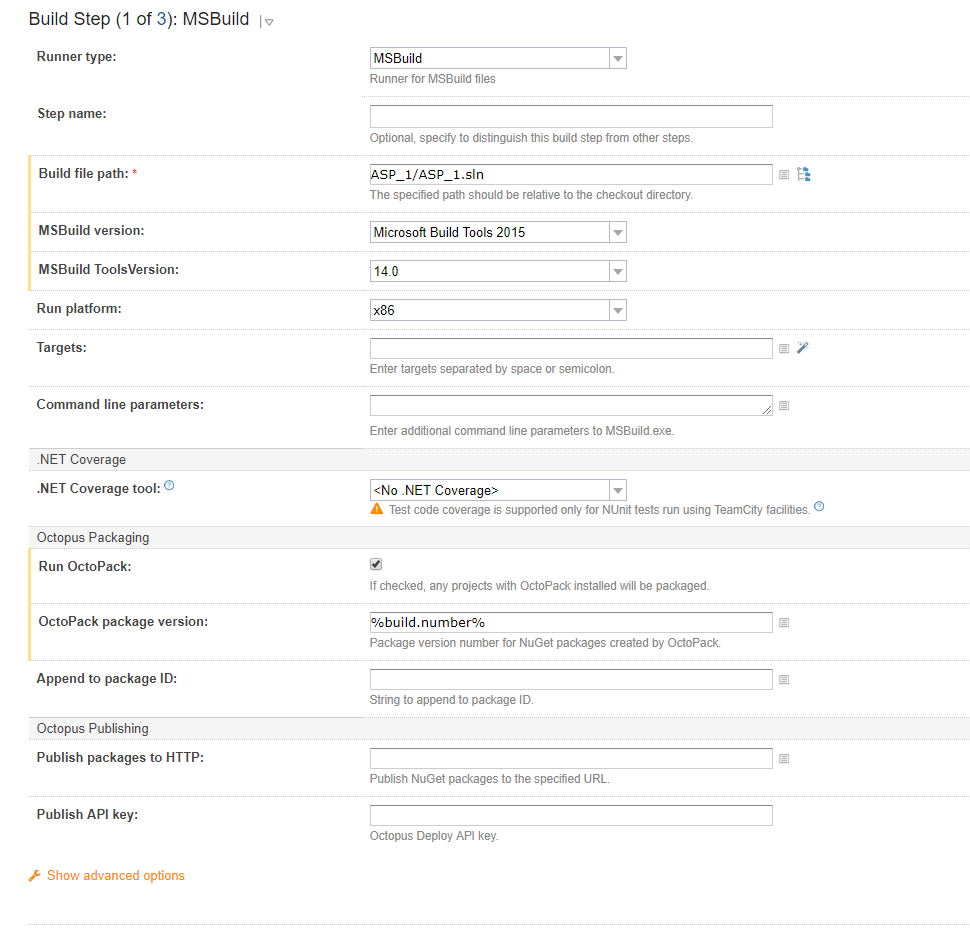
The various build steps in Teamcity were listed above. Each subsequent step executes only if the previous steps finish successfully.

Autodetected Build Step for Visual Studio would not build – due to Visual Studio being in Debug/Release Mode so MsBuild was used as a result.

## MSBuild

For the MSBuild step, point the build file path to the .sln file specifying the MSBuild version and platform. There was an issue where Teamcity MSBuild.exe was located on the system. To fix this, I configured the MSBuild.exe path on the BuildAgents folder in C:\TCBuildAgent1\conf\buildAgent.properties file and added the following line:

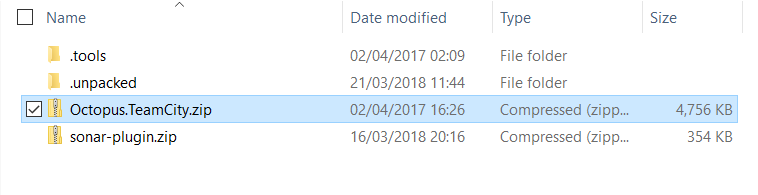


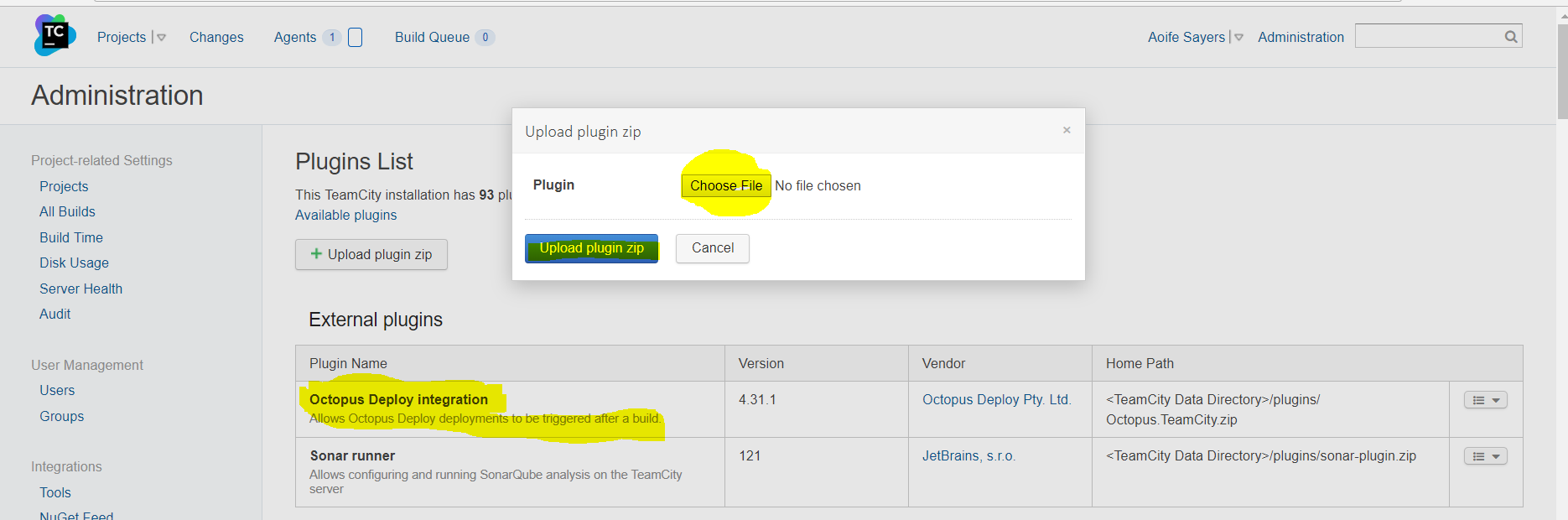


## Installing Plugins for Teamcity:

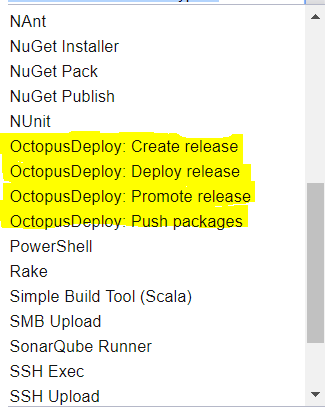
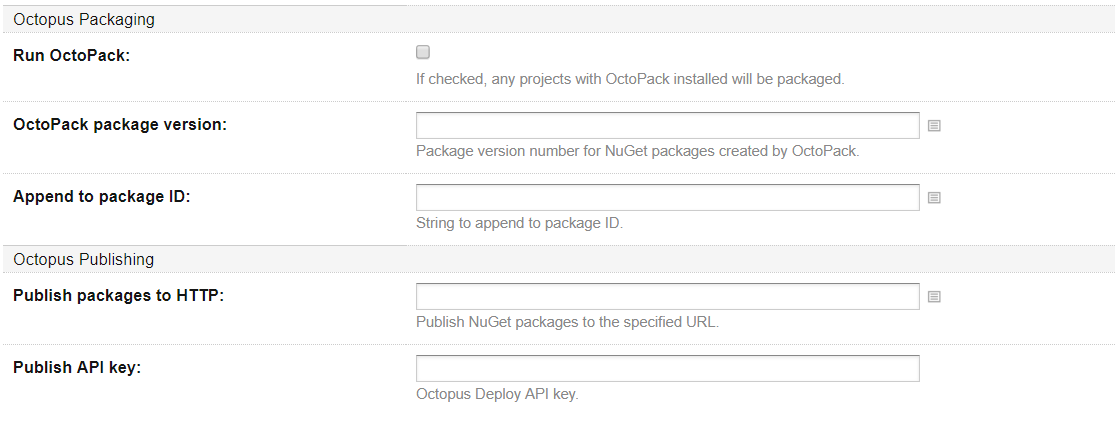
### Octopus Deploy

I installed an Octopus Deploy plugin zip file for Teamcity by shutting down the Teamcity server service. Copy the Zip file to the plugins folder in the Teamcity directory on the C:/. Then in Teamcity go to Administration > Plugins List > Upload Plugin Zip and change directory to where the Octopus Deploy zip folder is.





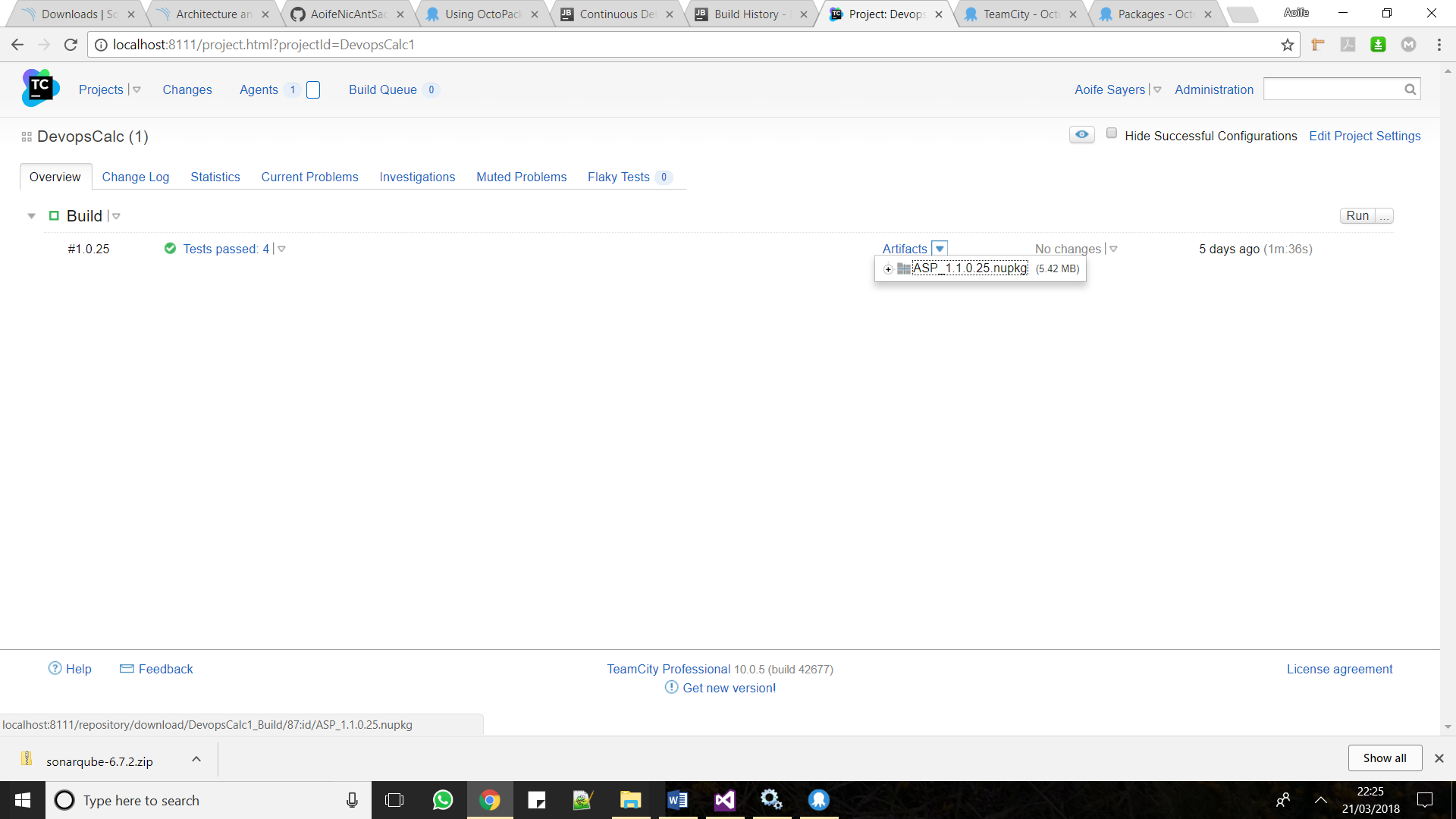
Restart the Teamcity Server (Deploy, 2018). The Teamcity plugin provides functionality for creating and pushing Octopus Packaging (in MSBuild) to the Octopus Server, Creating a release on the Octopus Deploy server, deploying a release and promoting releases on Octopus Deploy.



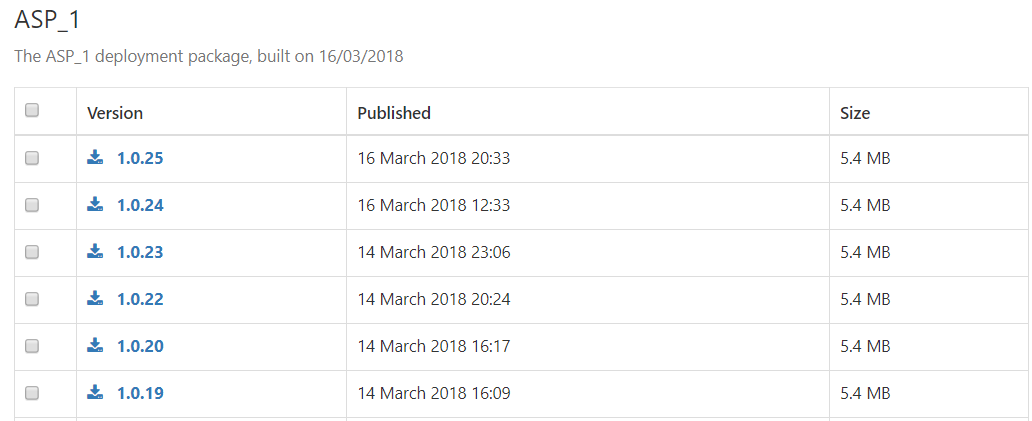
\*Need to enable the Nuget feed under Administration > Integrations > Nuget Feed. Teamcity has public guest feed & Teamcity private nuget feeds <http://localhost/guestAuth/app/nuget/v1/FeedService.svc/>

OctoPack: creates nupkg package of the project as an artifact with files required for deployment. Octopack is installed as a package to a .NET project. Octopack discards .sln, .csproj, web.release.config, web.debug.config, obj and Properties folders. Octopack keeps relevant web.config/app.config and bin folder with .dll. Octopack occurs within the MSBuild step in Teamcity and outputs nupkg files.

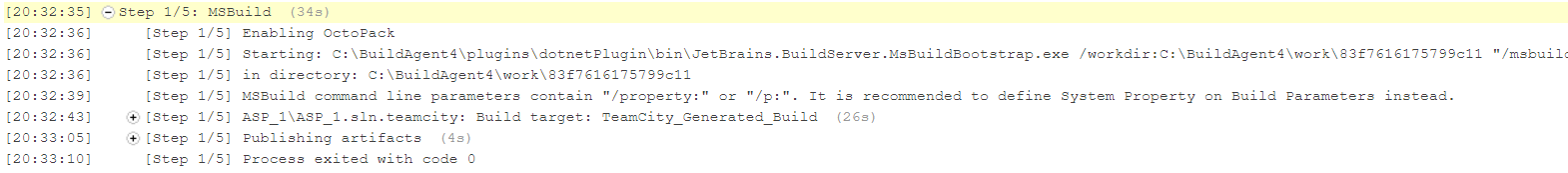
The resulting artifact of MSBuild build number 1.0.25 in Teamcity.



Artifacts are sent to Octopus Deploys Server nuget feed with an API key. List of all the nugpkgs deployed to the Octopus Deploy Server’s Library from Teamcity are shown below.

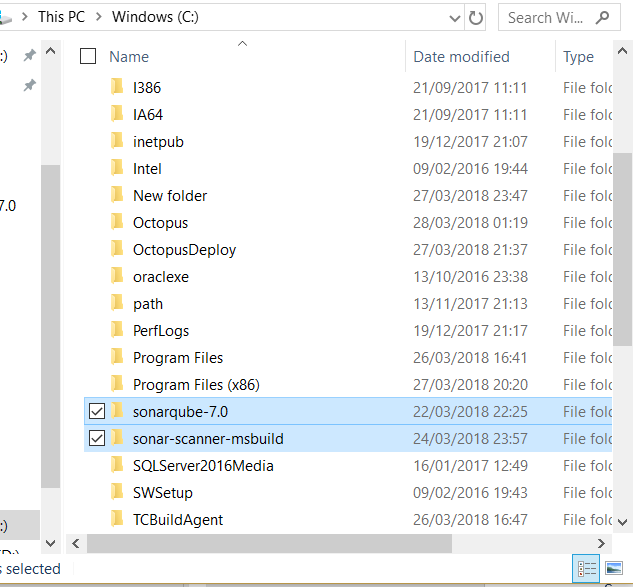


MSBuild log in Teamcity shows a successful MSBuild



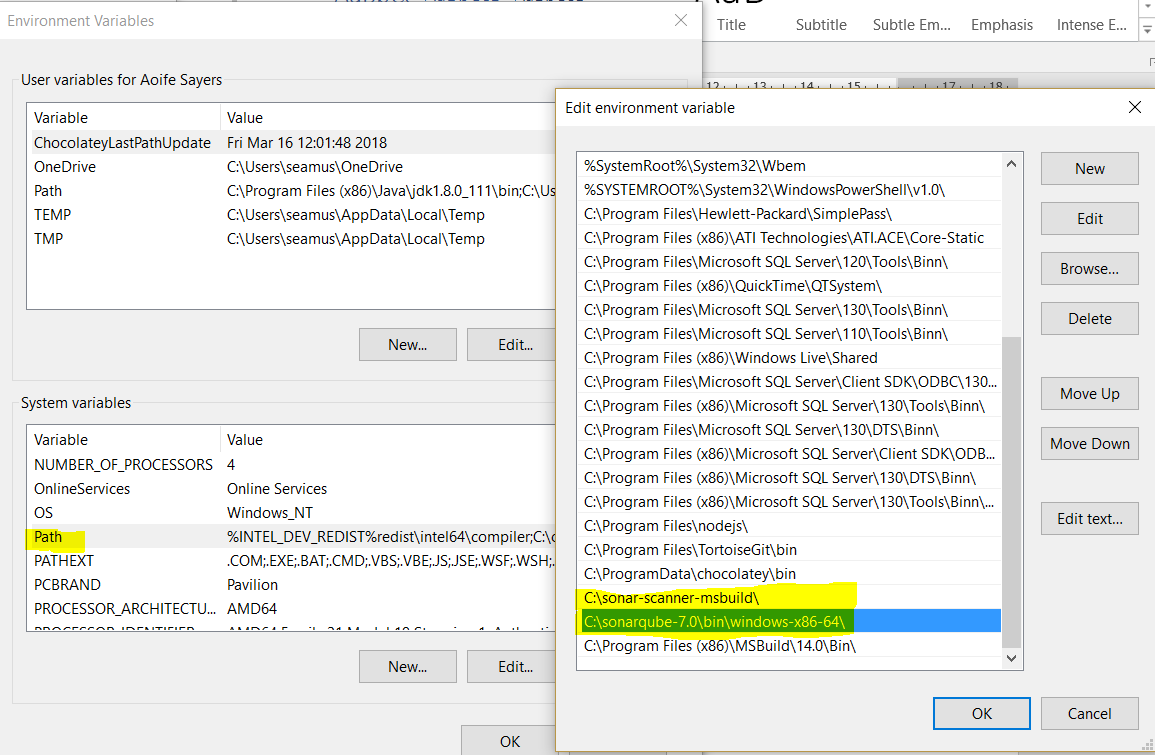
## SonarQube

I installed the SonarQube Server locally on C:\ - C:\sonarqube-7.0. I also downloaded the sonar-scanner-msbuild for analysing C# projects

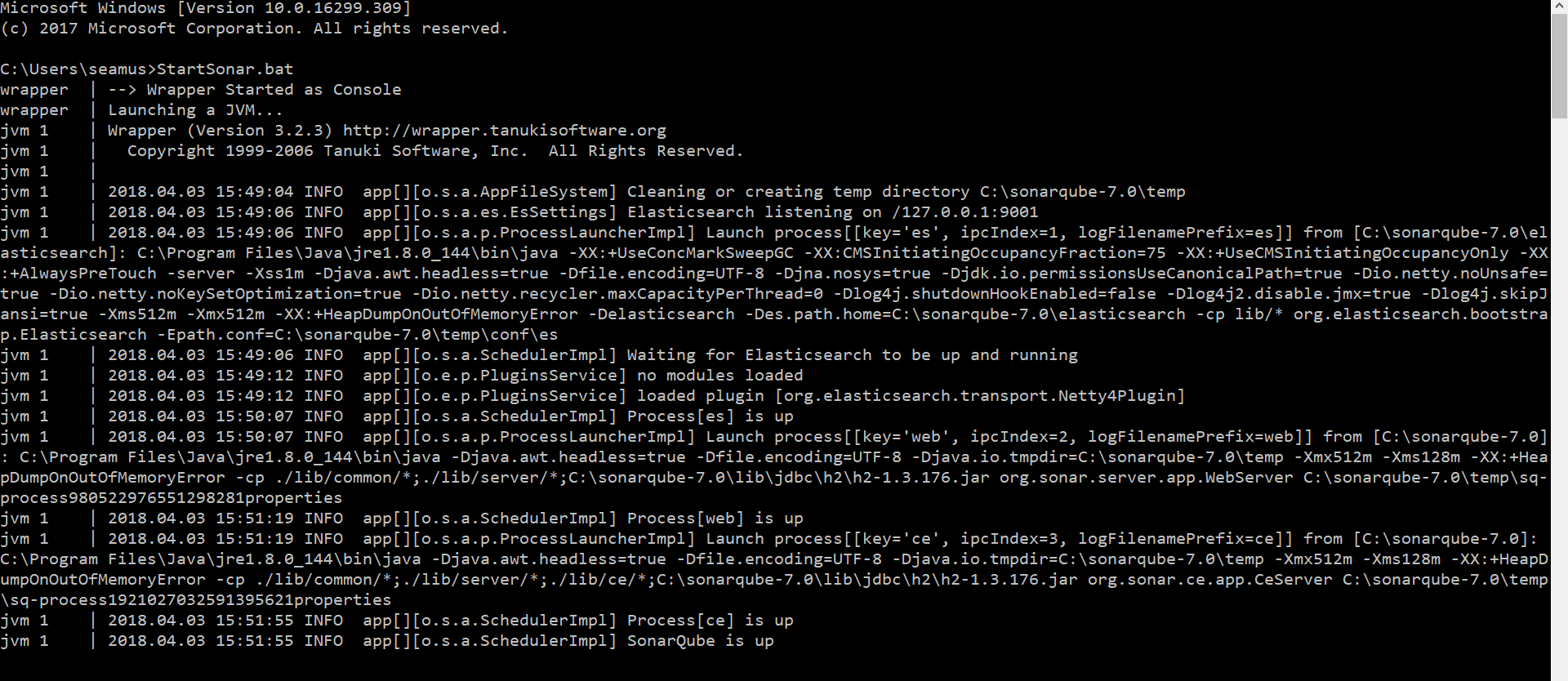


Added System Environment variables for SonarQube Server and SonarQube Scanner for MSBuild and MSBuild.exe

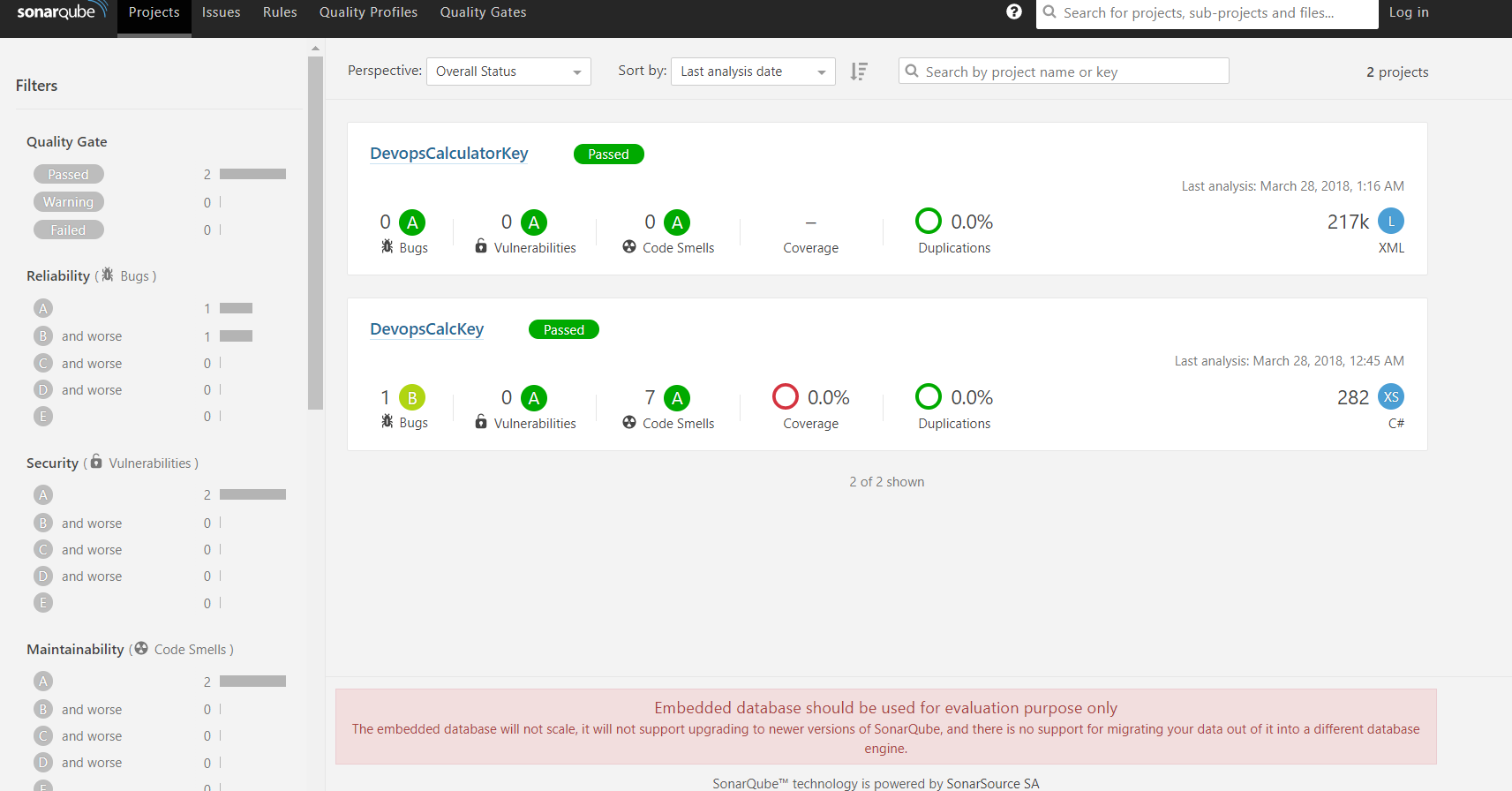
I added the Environment into the PATH variable for the Sever and for the MSBUILD



To start the SonarQube Server open a command prompt cmd and type StartSonar.bat



To logon to SonarQube proceed to <http://localhost:9000> and logon with the credentials admin for the username and admin for the password



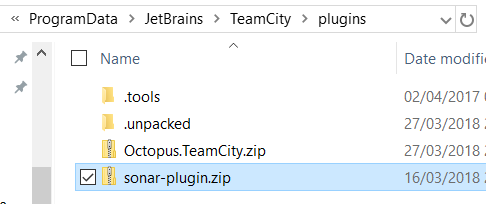
To run SonarQube scanner msbuild manually I used the following command

SonarQube.Scanner.MSBuild.exe begin /k:"DevopsCalculatorKey" /d:sonar.host.url="http://localhost:9000" /d:sonar.login="7c0e0c6be0b5c419d5e37ce1aba445daf25010bd"

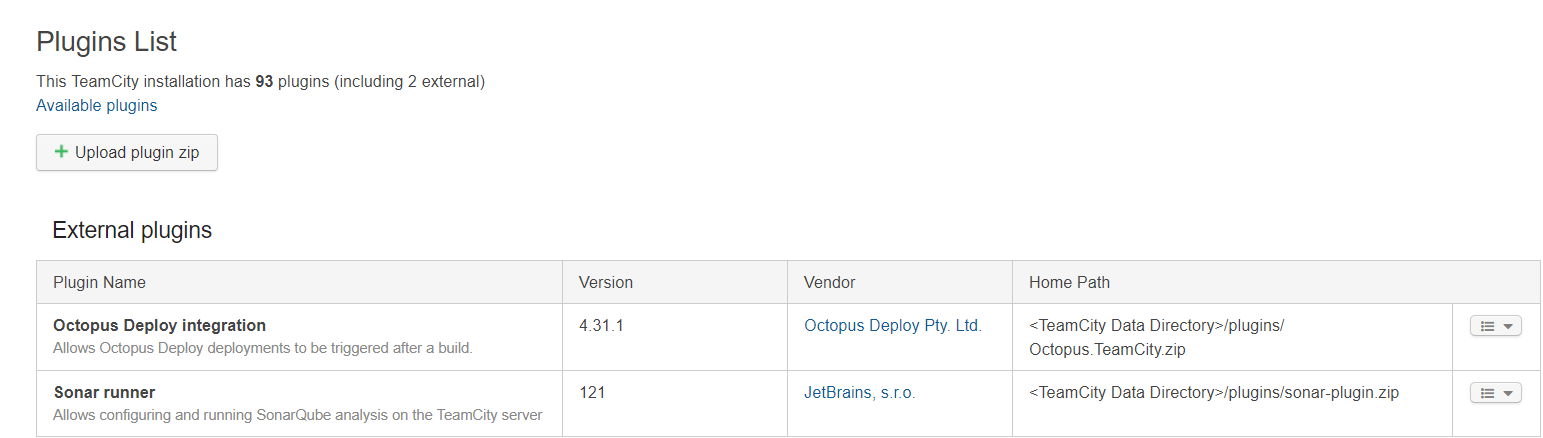
MsBuild.exe /t:Rebuild

SonarQube.Scanner.MSBuild.exe end /d:sonar.login="7c0e0c6be0b5c419d5e37ce1aba445daf25010bd"

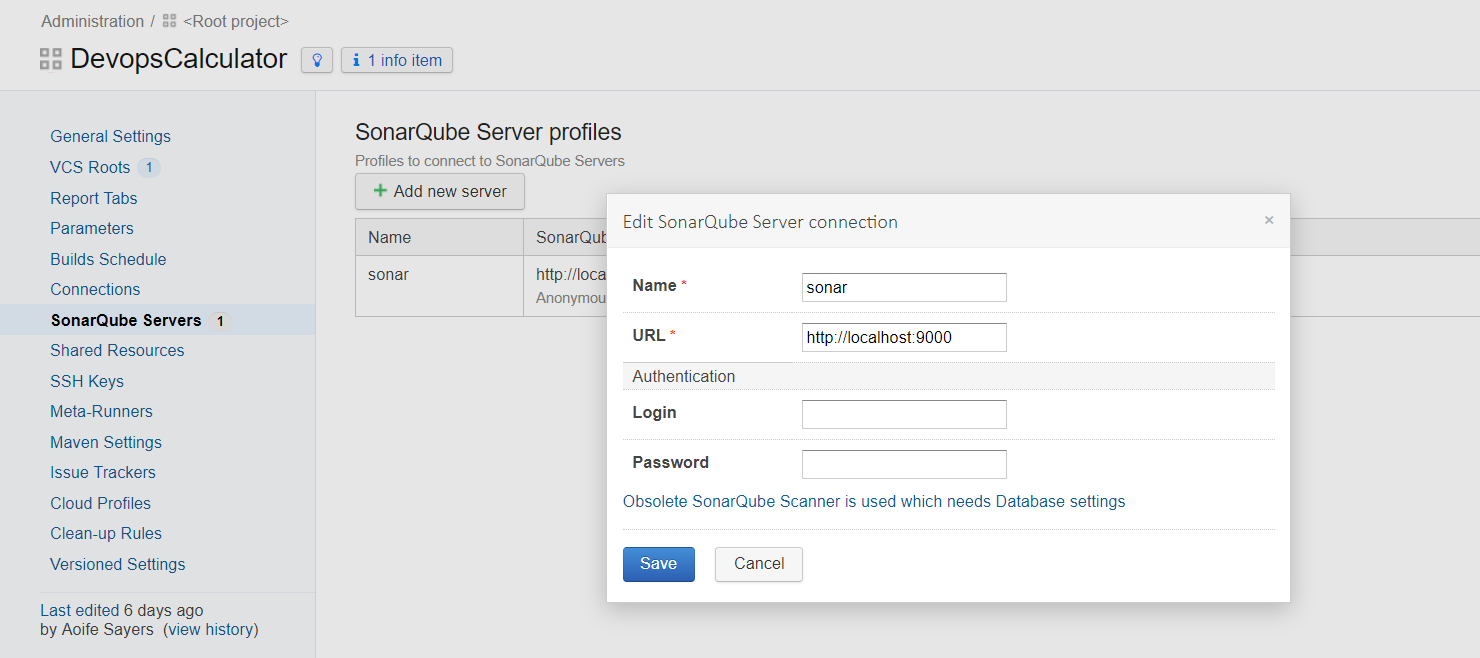
For my deployment pipeline I automated this step via Teamcity by installing the sonar plugin zip for Teamcity and placing it within the plugin directory in the Teamcity folder



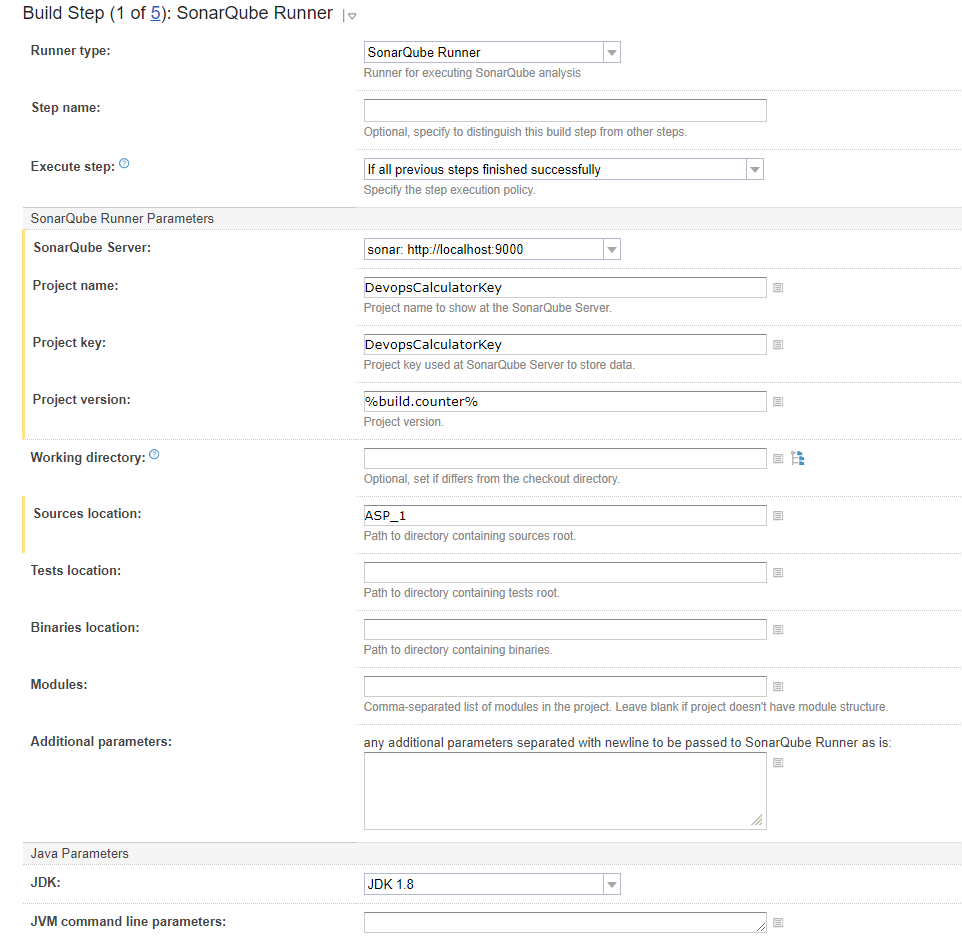
Teamcity requires to point to the plugins and will then restart



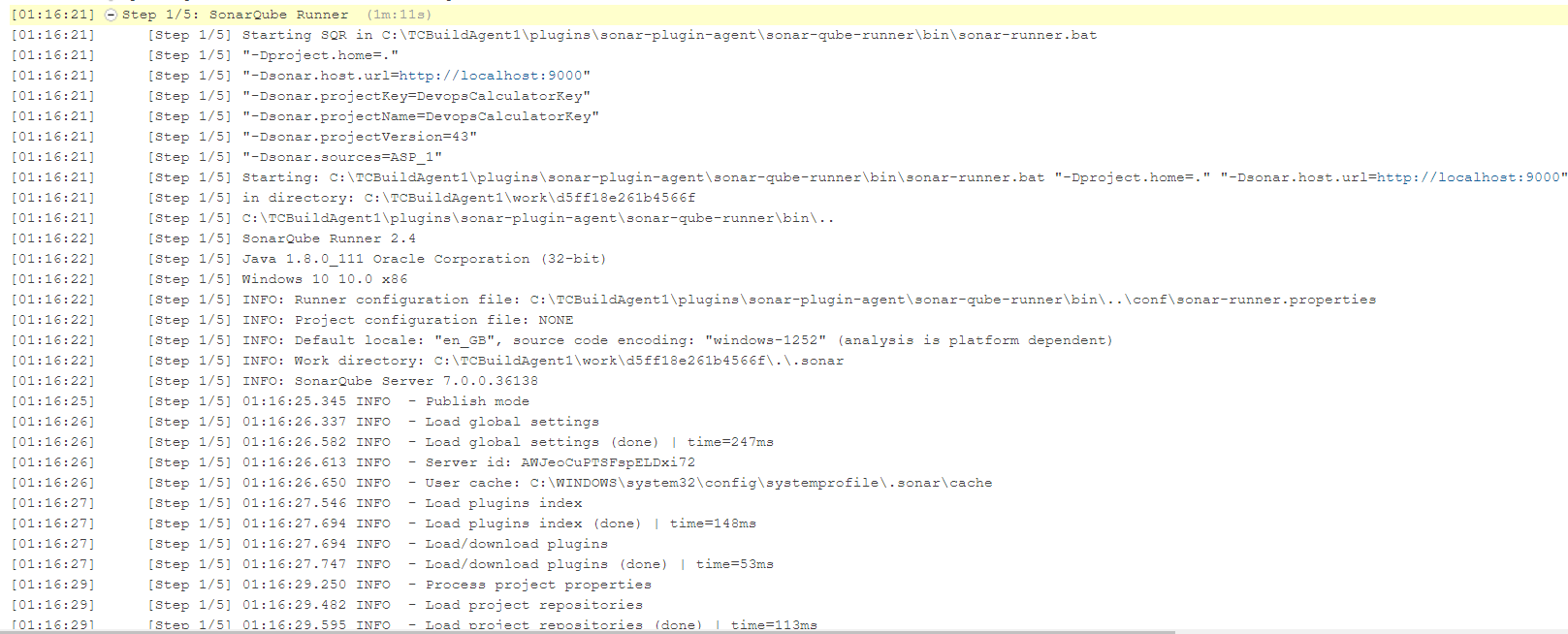
Configure the SonarQube Server as follows



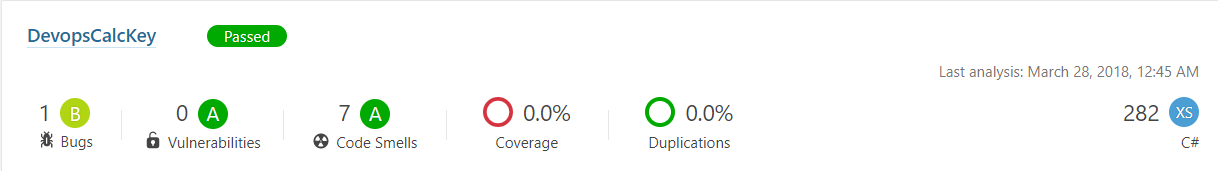
Add a SonarQube Runner build step



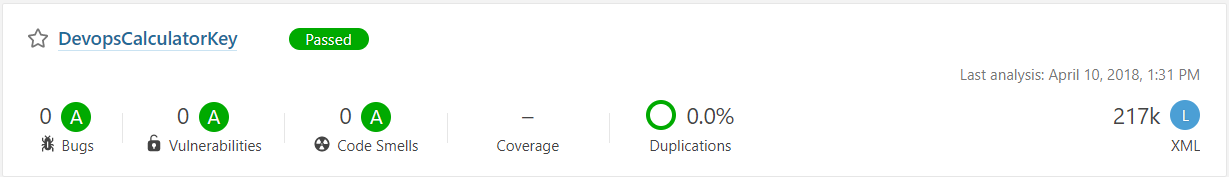
A snapshot of SonarQube Runner build step in Teamcity’s build log



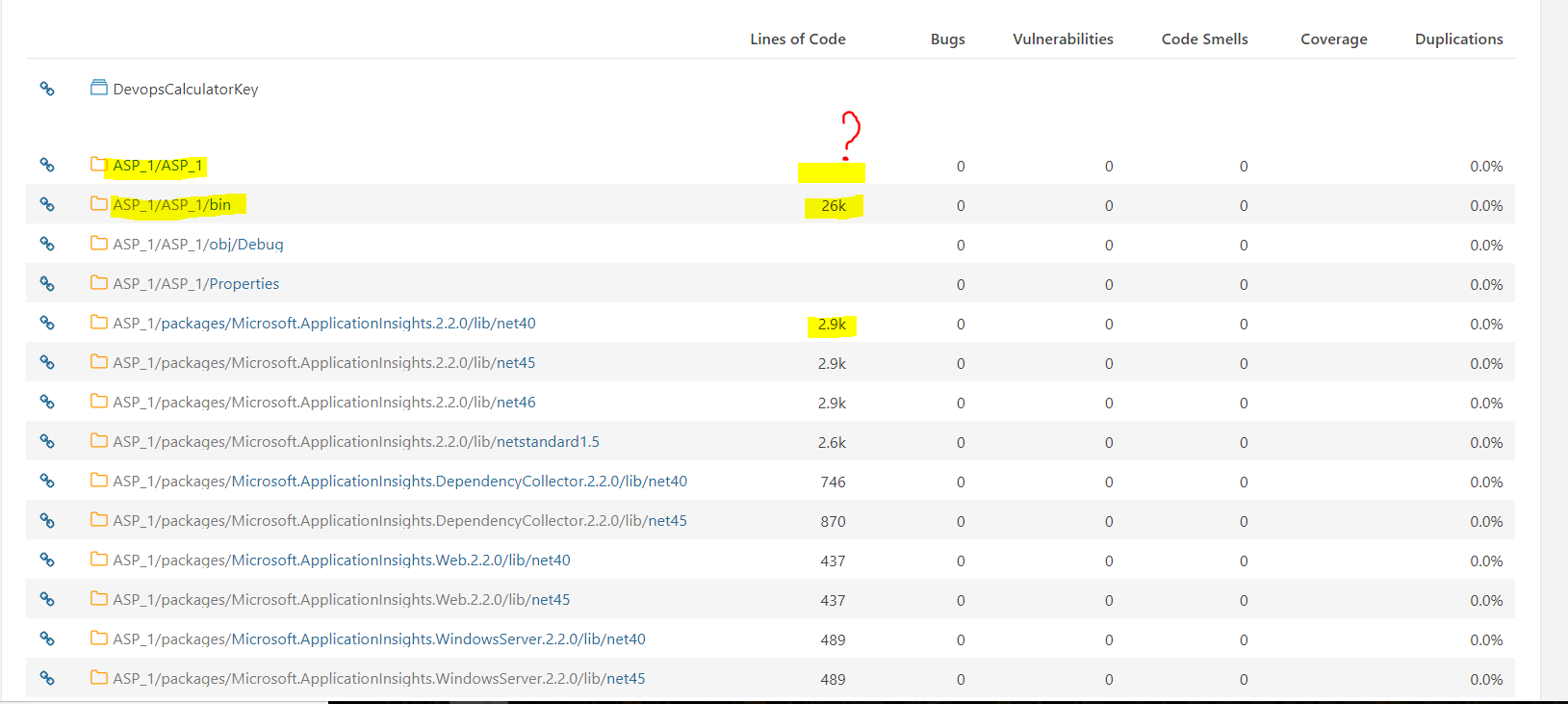
Analysis of DevopsCalculator project in Sonar using the manual command brings up results on the SonarQube server on localhost:9000



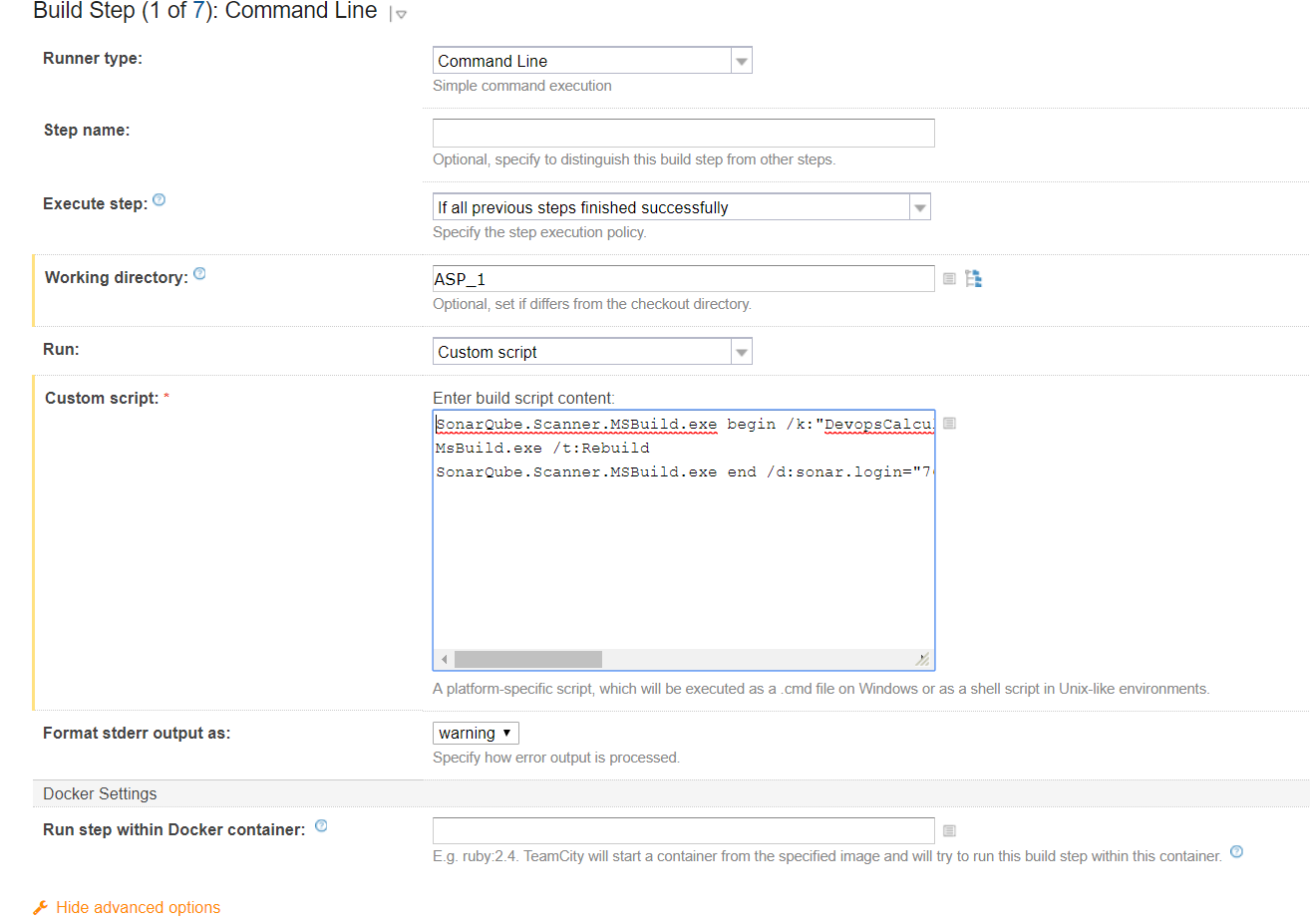
However, the automated build step of executing the SonarQube Scanner does not work on Teamcity and shows up as follows



It picks up on the number of Lines of code in the bin folder and packages file but no the .cs files with tests and code etc

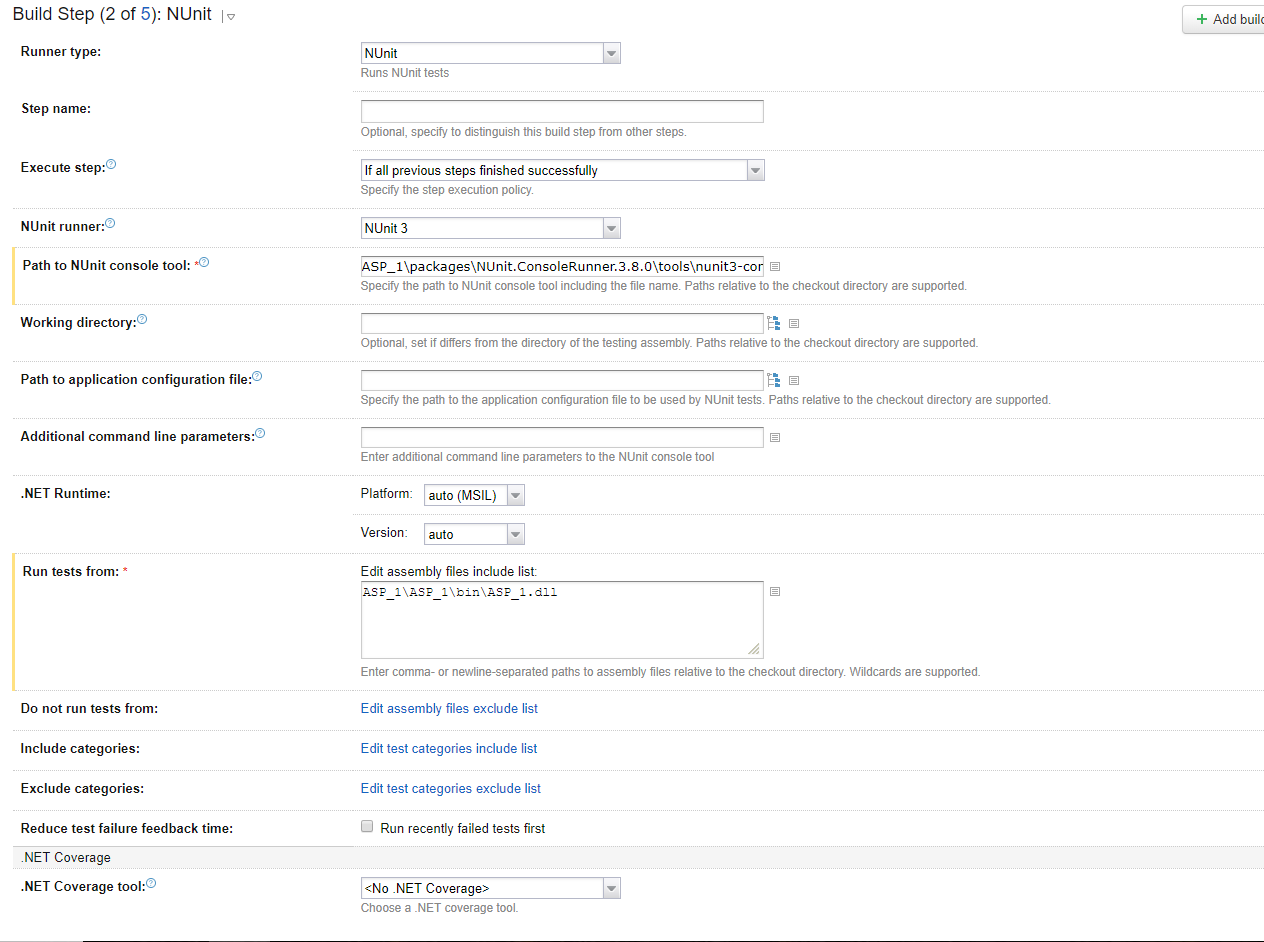


As a workaround I tried to execute a command line step using the command that worked previously on Teamcity, however as my version control system is on Github I think I am restricted

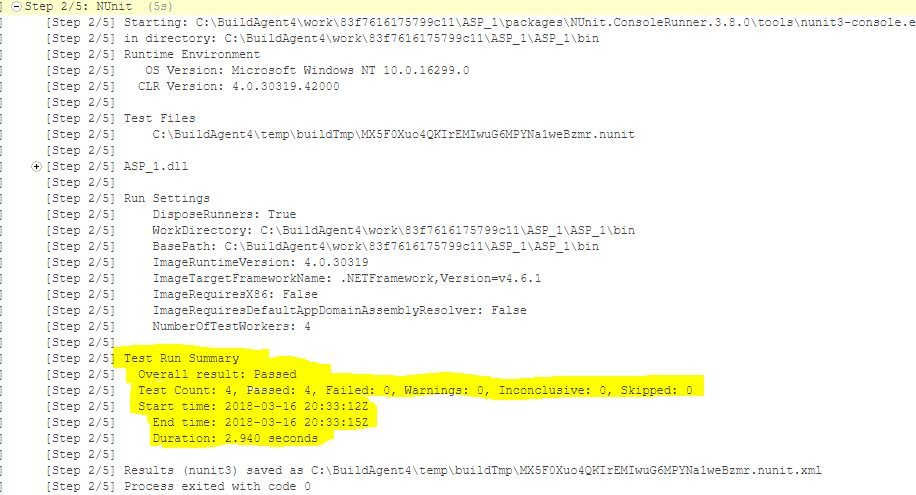


An error of no .sln file for the MsBuild showed up here.

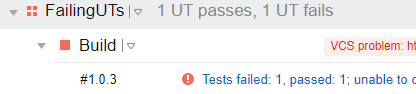
## Nunit

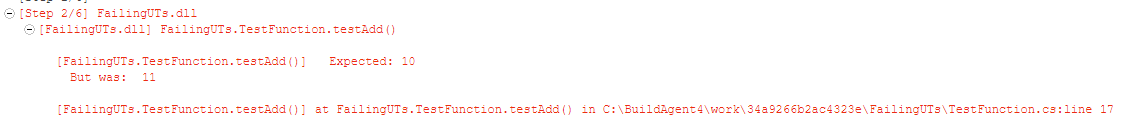
To run Nunit on Teamcity, NUnit, NUnit Console Runner and NUnit Console are required to be installed as packages in the .NET project. The NUnit Build step in Teamcity requires you to provide path to the NUnit console tool and specify the path to the NUnit projects.

Successful Passing of test results on Teamcity

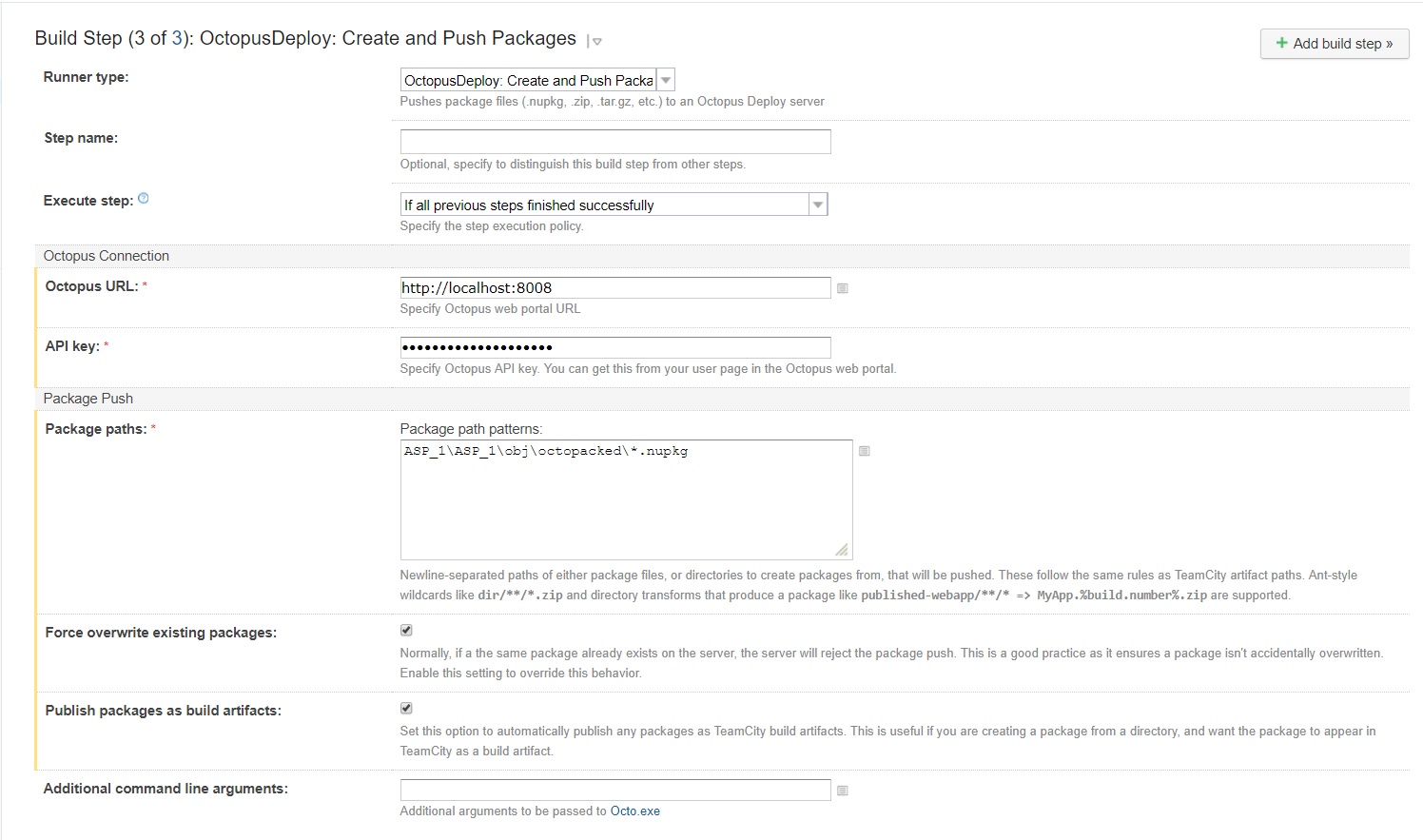
 

Failing of tests on Teamcity will fail the build and the build will not progress onto the subsequent build steps.

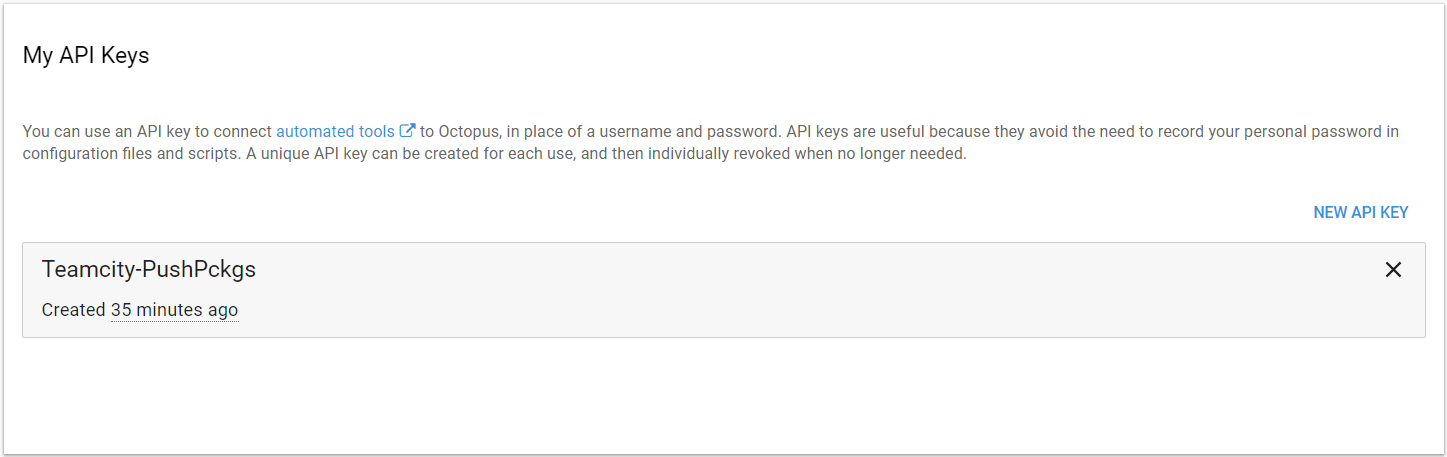




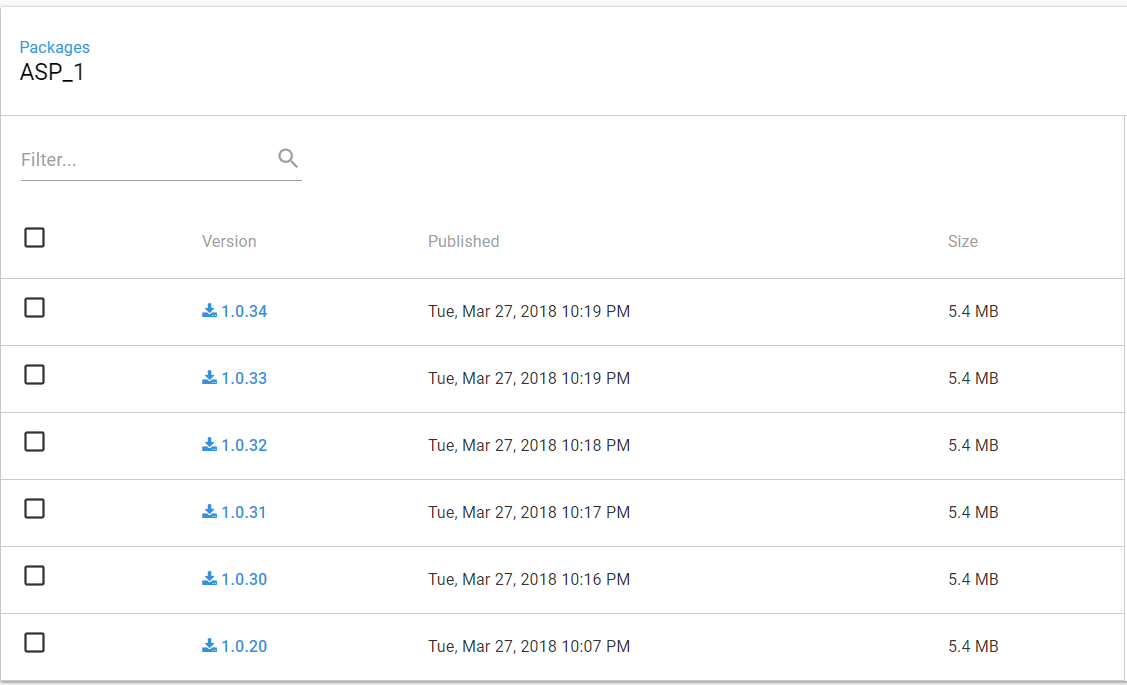
## Octopus Deploy: Create & Push packages



To create an API key on Octopus > <username> > profile > My API keys > New API key



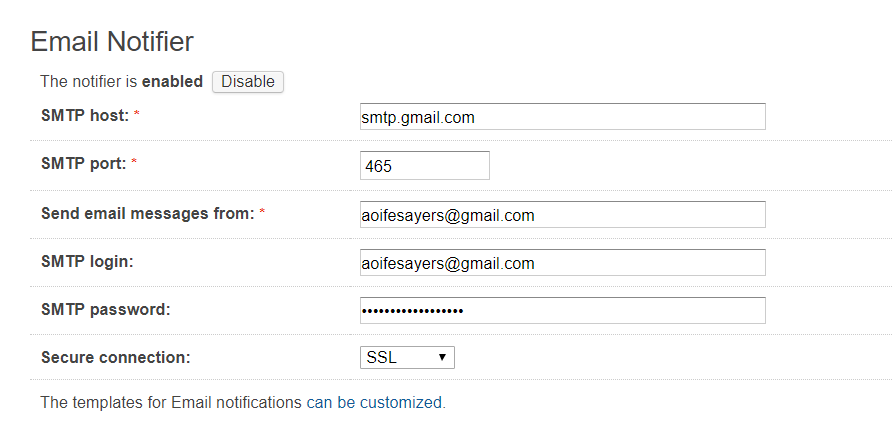
The Nupkg Artifact created by Octopack should end up on Octopus Deploys in build repository in the ‘Library’ with the version number of the build number on Teamcity. All the various packages from builds pushed to the Octopus Deploy library are shown as follows. Octopus Deploy deploys the package with the highest build number e.g. 1.0.34 in the case here.



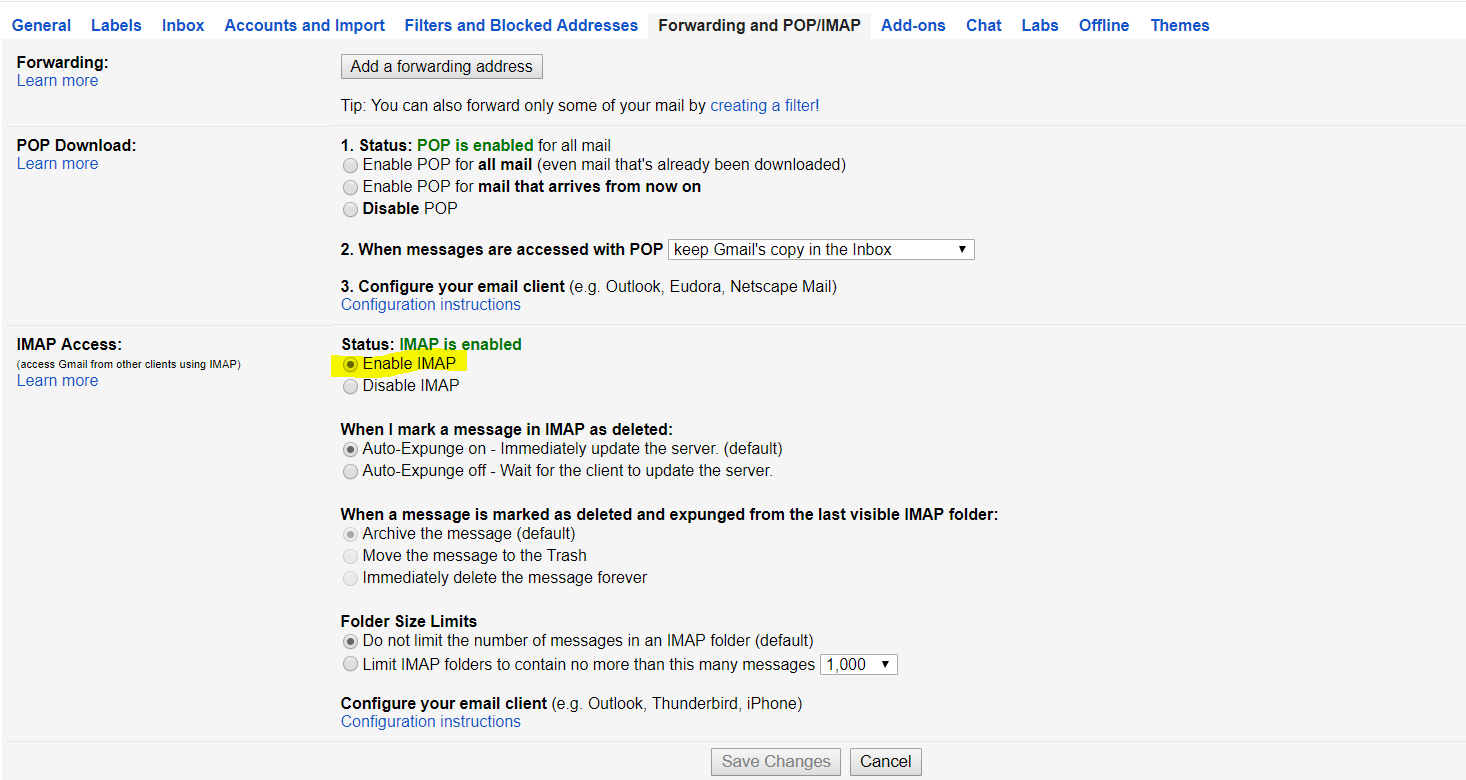
## Teamcity Notifications

### Email Notifier

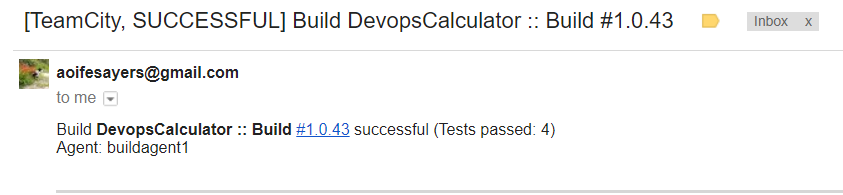
I configured an Email Notifier in Teamcity under Administration > Server Administration > Email Notifier and configured the following settings for a Gmail account

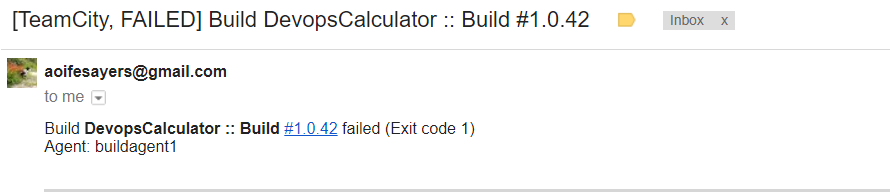


On Gmail I had to change settings to Enable IMAP access



Subsequently, Teamcity will send emails to the address specified every time a build completes successfully or unsuccessfully





# Octopus Deploy

Octopus Deploy requires SQL Server database. Octopus Deploy Server is running on localhost:8008

### Octopus Deploy

* Distributing applications to all the remote machines, securely
* Environment-specific configuration, like connection strings
* Configuring IIS sites and installing Windows Services
* Doing all of the above across many machines in parallel
* Popular .NET deployment tool with integration for CI servers Teamcity (Jetbrains), Team Foundation Server (Microsoft)

45 day free trial

Community Version – Free discontinued from February 15 – allow up to 5 projects

Professional -  20 users, 20 projects and 20 target machines

Team - 60 users, 60 projects and 60 target machines.

Enterprise Unlimited users, projects and target machines.

High Availability - Unlimited users, projects, target machines and multiple nodes

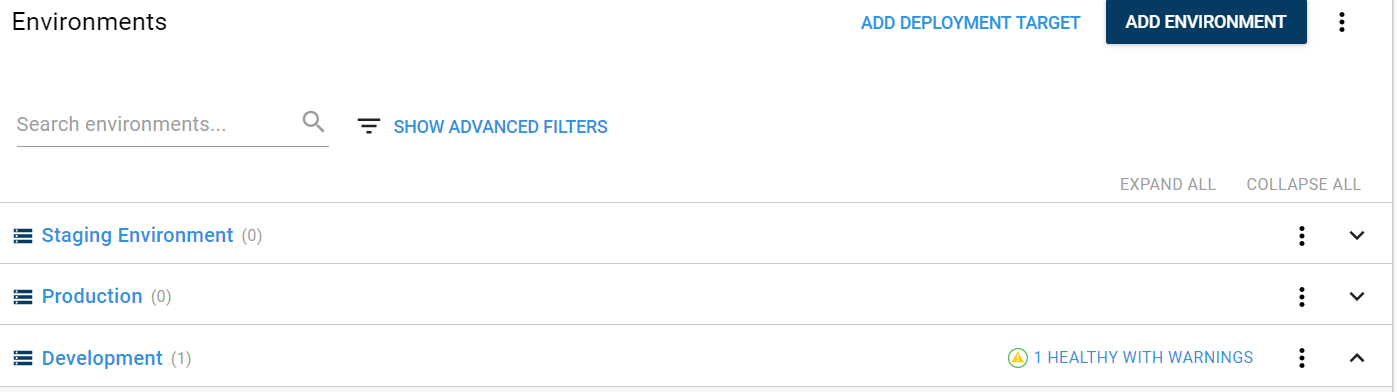
Command line Interface - octo.exe

### Machines Roles

* Each of these servers, whether physical or virtual machines, would be a machine within Octopus.

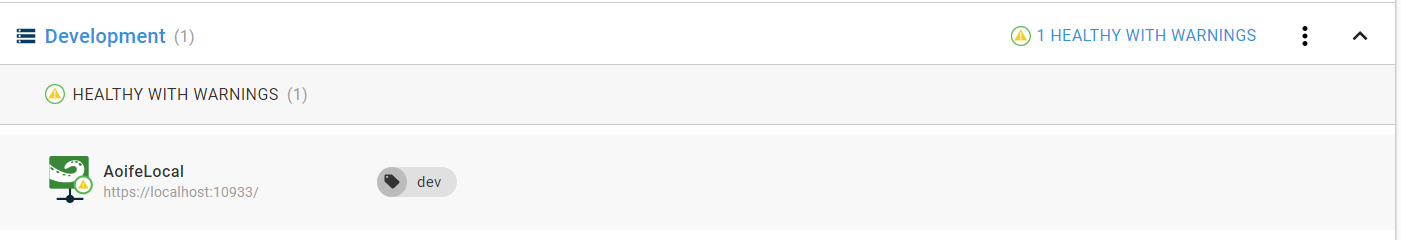
### Environments

* An environment is made up of multiple machines, and each machine is tagged with a set of roles
* A group of these machines that are deployed to at the same time is called an environment.
* Each environment contains one or more Deployment Targets
* To add an Environment go to Infrastructure > Environments > Add Environment

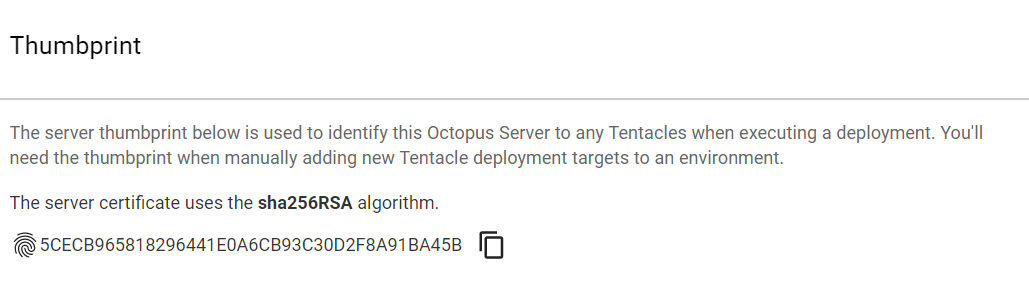


### Deployment Targets

Within your Environments, you have deployment Targets – instances you deploy to

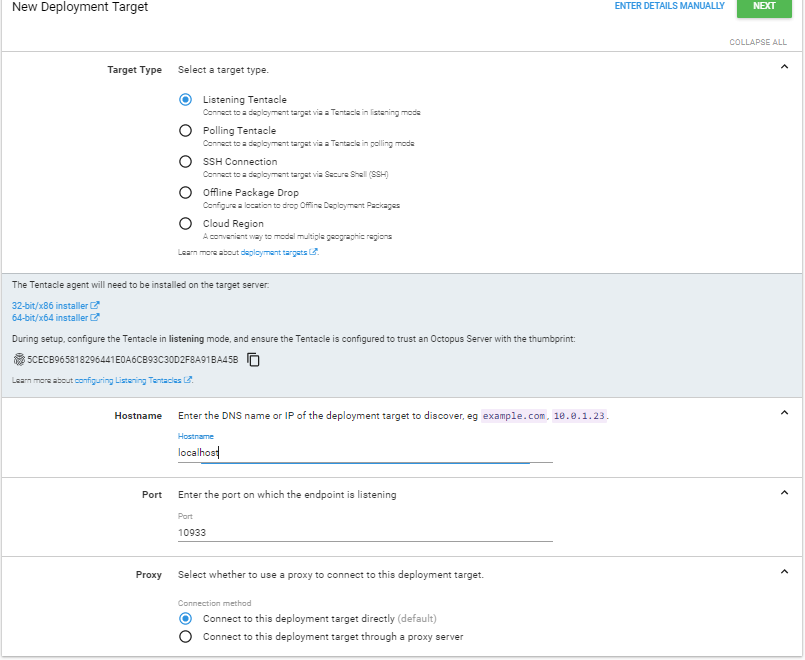


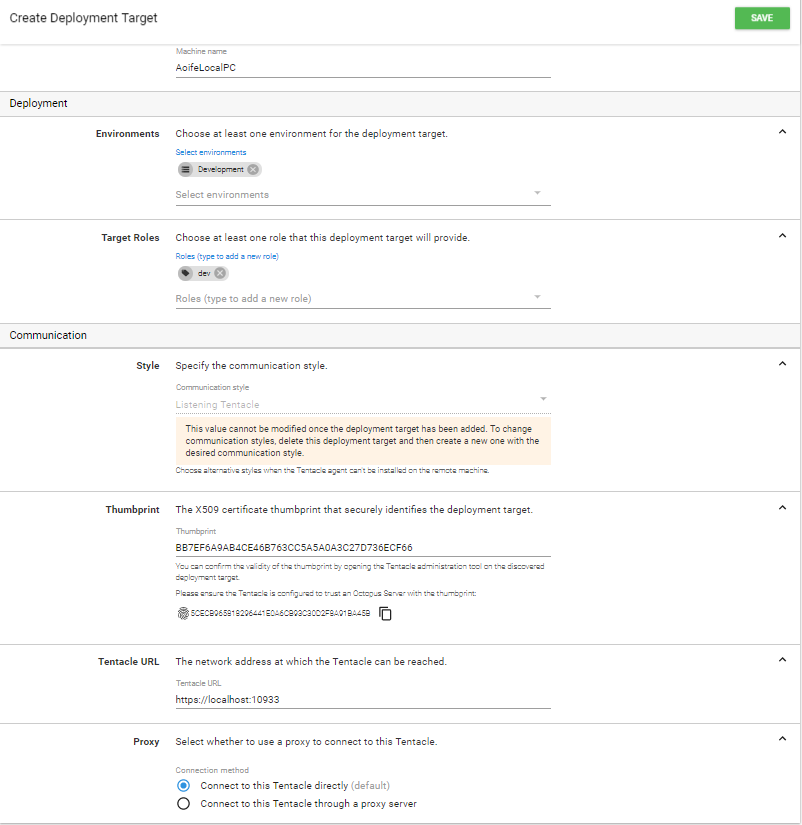
To add deployment targets, you need to install tentacle manager on each deployment target/machine with the thumbprint of the Octopus Server. To get the thumbprint on the Octopus Server go Configuration > Thumbprint



Once you have the thumbprint, go to the deployment target and install Tentacle Manager on the

deployment target. On the Octopus Server Web UI go Infrastructure > Deployment Targets > Add Deployment Target. Set the Target Type to Listening Tentacle. Enter the hostname and port number (Listening Tentacles are always on port 10933). Select the environment and roles.. Copy the thumbprint from the Octopus server and paste.





### Tentacle

* Run as Windows services - Tentacle manager installed on deployment targets
* Lightweight agent service/job runner
* installed onto target deployment machines.
* Connection secured by public & private key  - no passwords

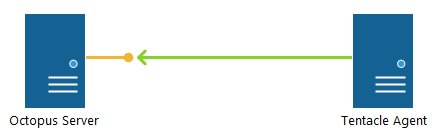
Listening Tentacles



Tentacle plays the role of server and Octopus as the client:

* Octopus establishes the HTTPS connection with the Tentacle
* The Tentacle presents its certificate as the server certificate allowing Octopus to verify the identity of the Tentacle
* Octopus presents its certificate as a client certificate so the Tentacle can verify the identity of Octopus
* Once the identity of the Octopus and Tentacle have been established the connection is held open and Octopus will start issuing commands to the Tentacle
* Listening Tentacles by default on port 10933

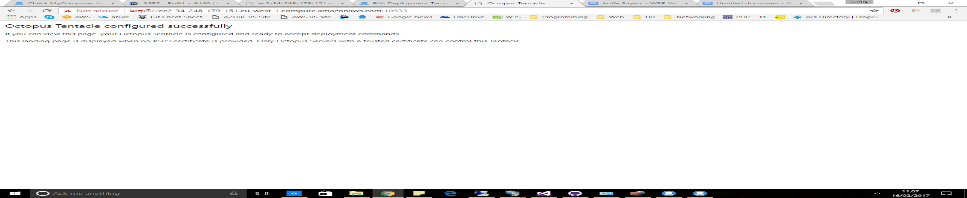
Polling Tentacles



Octopus plays the role of server and Tentacle as the client:

* The Tentacle establishes the HTTPS connection with Octopus
* Octopus presents its certificate as the server certificate allowing the Tentacle to verify the identity of Octopus
* The Tentacle presents its certificate as a client certificate so Octopus can verify the identity of the Tentacle
* Once the identity of the Octopus and Tentacle have been established the connection is held open and Octopus will start issuing commands to the Tentacle

Verify the deploy target is configured successfully http://localhost:10933

****

### Lifecycle

List of different environments involved in the deployment process

### Channels

Experimental branch of code, perhaps to introduce a new feature, or an entirely new version of your software.

### Variables

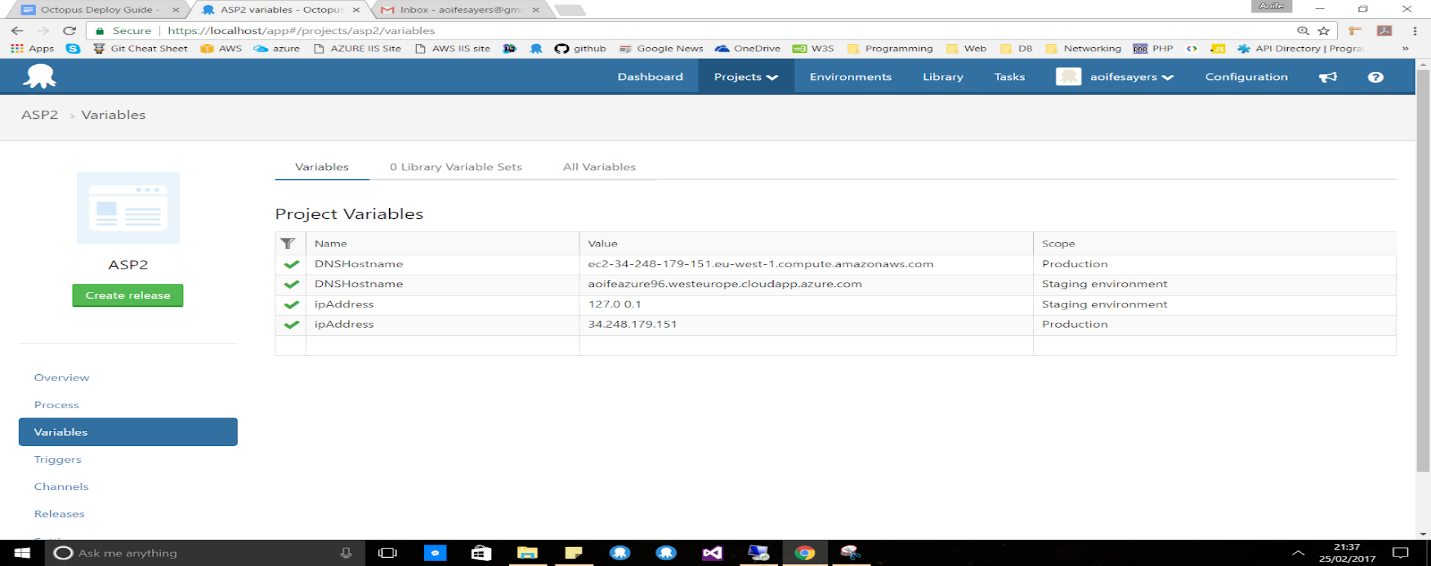
[Use variables to scope different settings to different environments](https://octopus.com/docs/deploying-applications/variables)

Octostache format #{myVar}

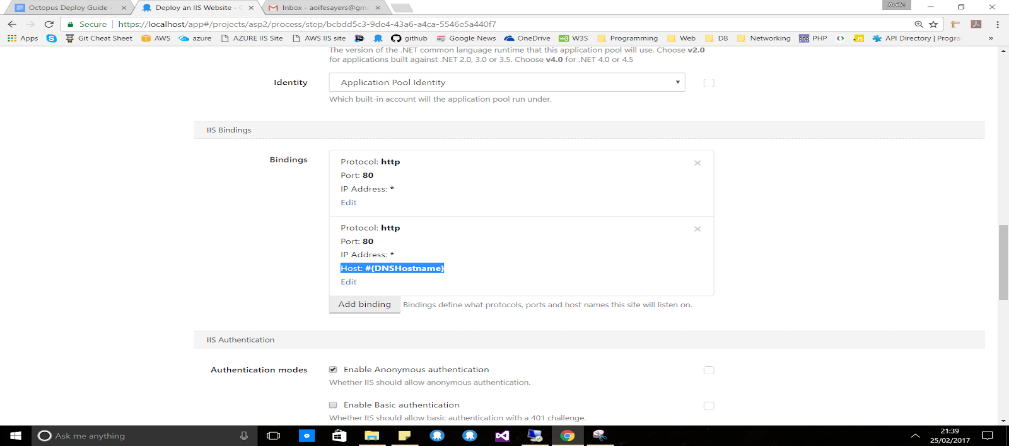
Octopus automatically takes care of:

* Replacing application settings
* Swapping connection strings
* Running environment-specific configuration transforms
* Configuring IIS application pools and web sites
* Installing and updating Windows Services

For both the Staging Environment (Azure) and Production Environment (AWS) I used the same variable for the IIS Website bindings so I could route on my local machine to check the website on both environments.



Using variables in a step

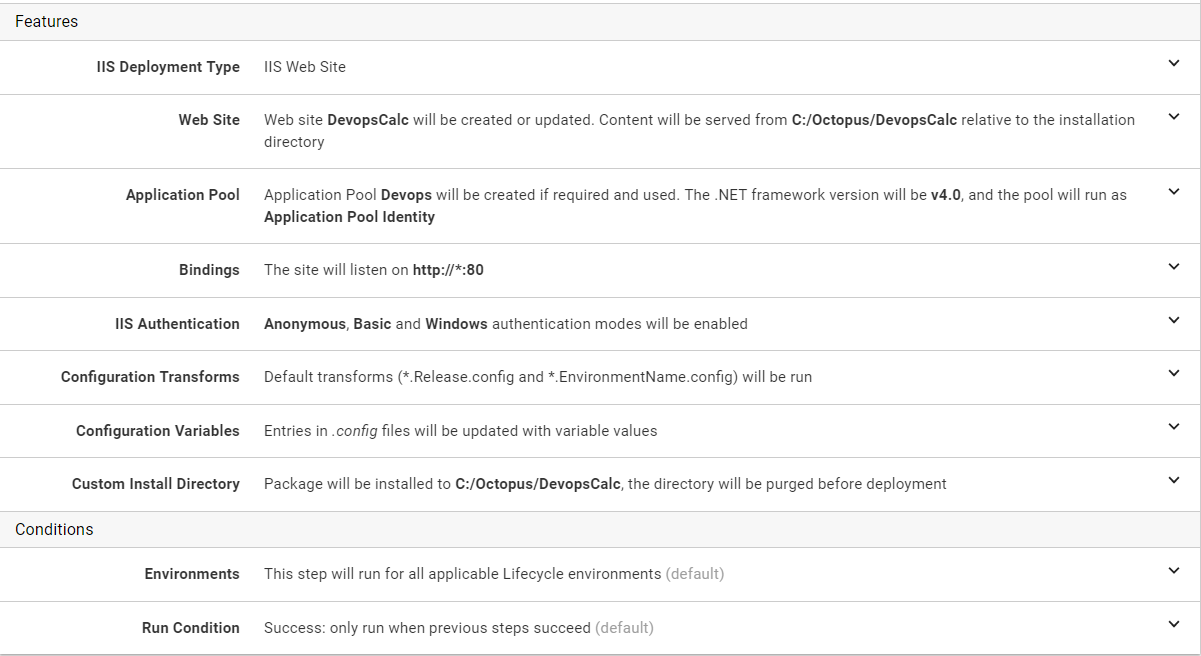


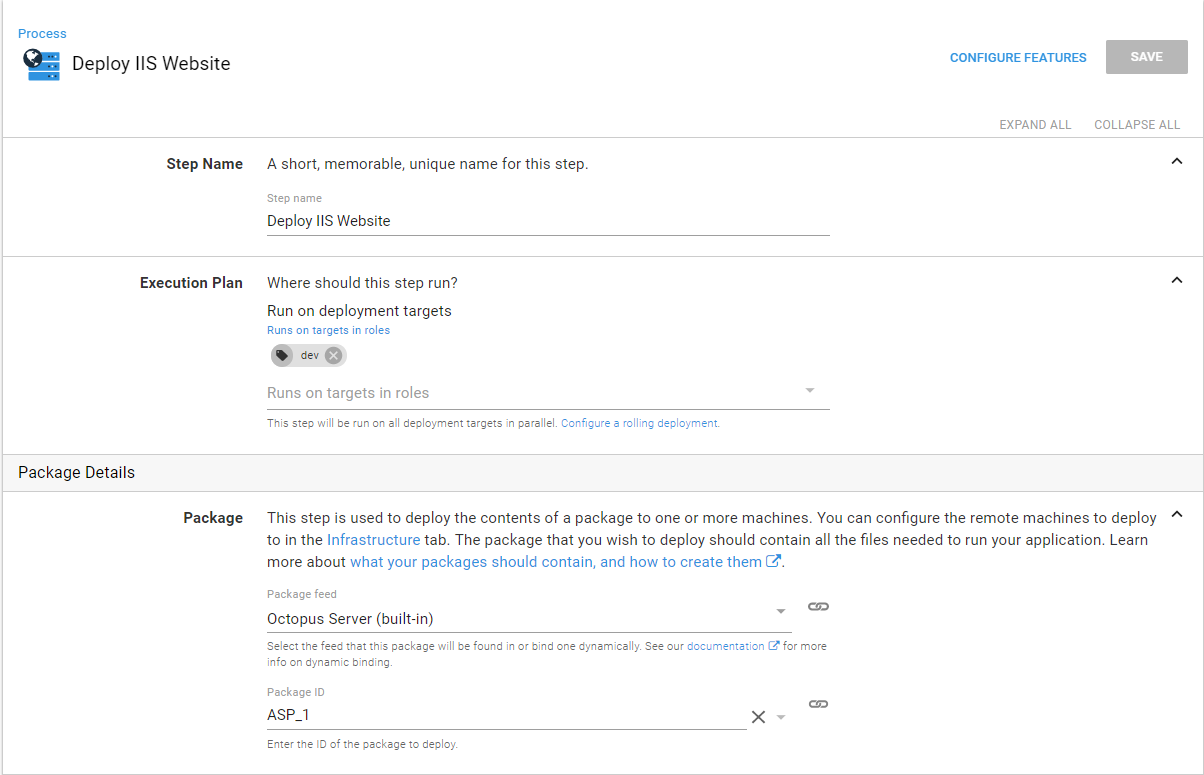
Also Octopus Deploy is able to configure <connectionStrings> with variables & transform the files during the deployment process

### Deployment Process/Step

* Specifies the steps that need to happen in a given order during the deployment
* Manual Intervention / Guided Failure Mode
  + Pause before failing
  + Ignore, stop, retry - add comments
* Installed Step Templates
  + IIS Website
  + Windows Service
  + Package
  + Run Powershell, C#, F# or bash scripts
  + Deploy Azure
    - Web App
    - Services
    - Powershell scripts
    - Resource Group
  + Send email - SMTP
* Community Step Templates (plus many many more).
  + Run NUnit Tests from assemblies
  + AWS
  + Git
  + SQL Server
* Install own step template

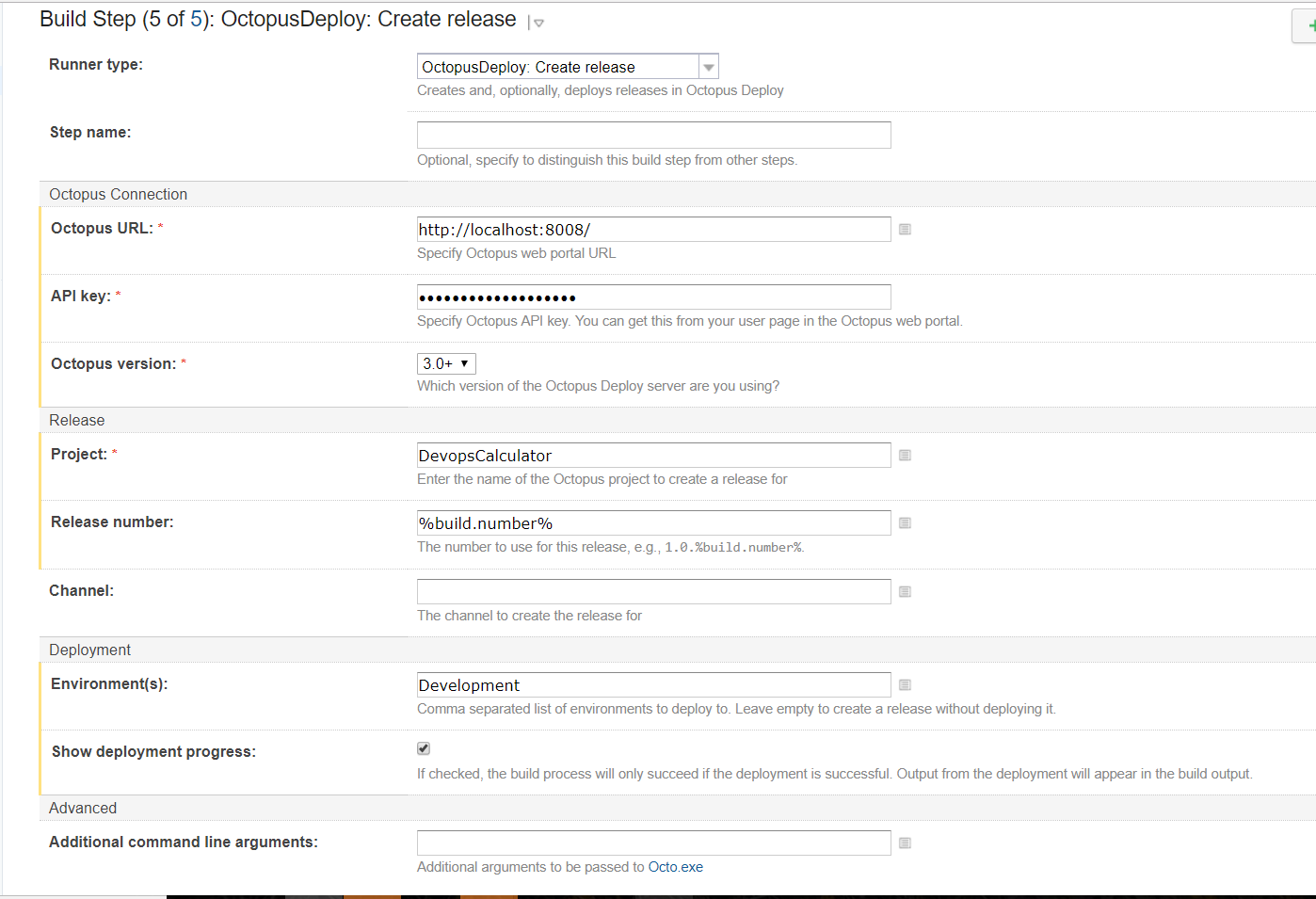
## Deploy an IIS website process



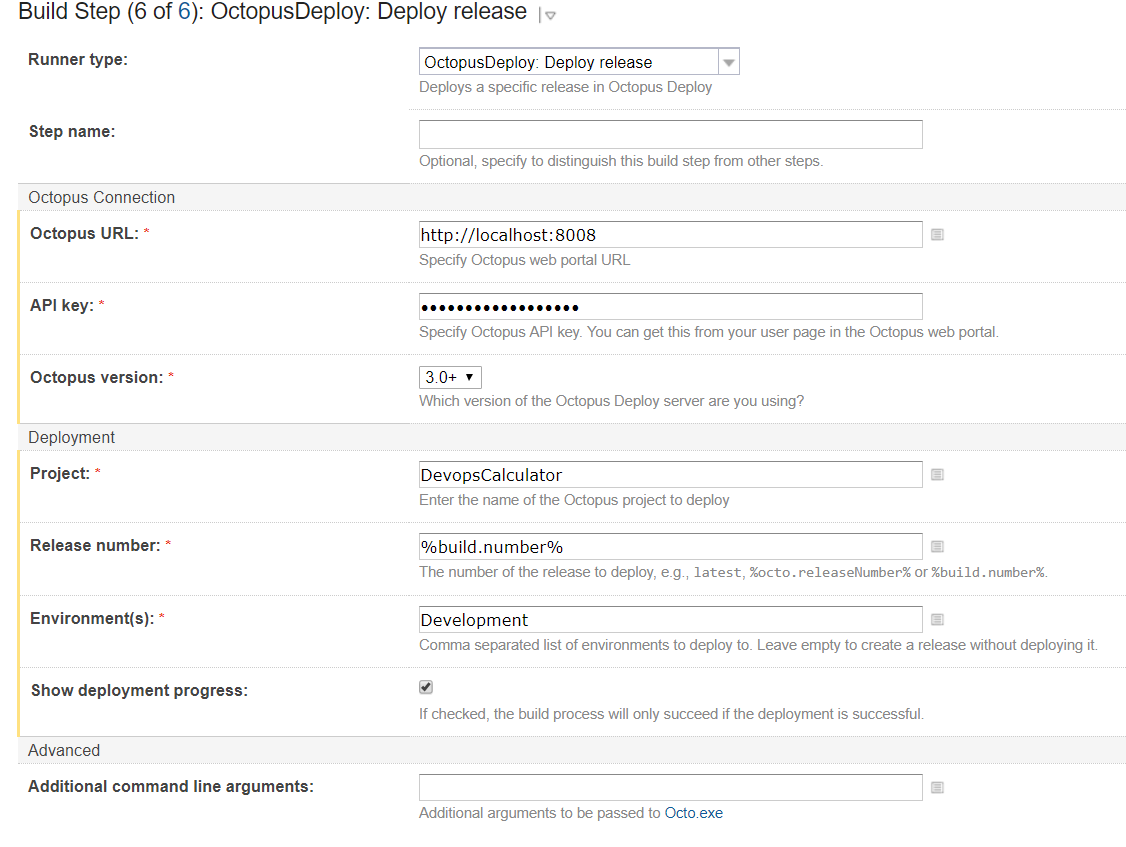


## Octopus Deploy: Create release

This build step in Teamcity automates creating a release in the Octopus Deploy Server. The build step in Teamcity is shown as follows



## Octopus Deploy: Deploy release



## Octopus Deploy: Promote release

# Virtual Machines

Windows Server 2016 VM are used for this using Amazon’s Web Services and Azure’s Web Services

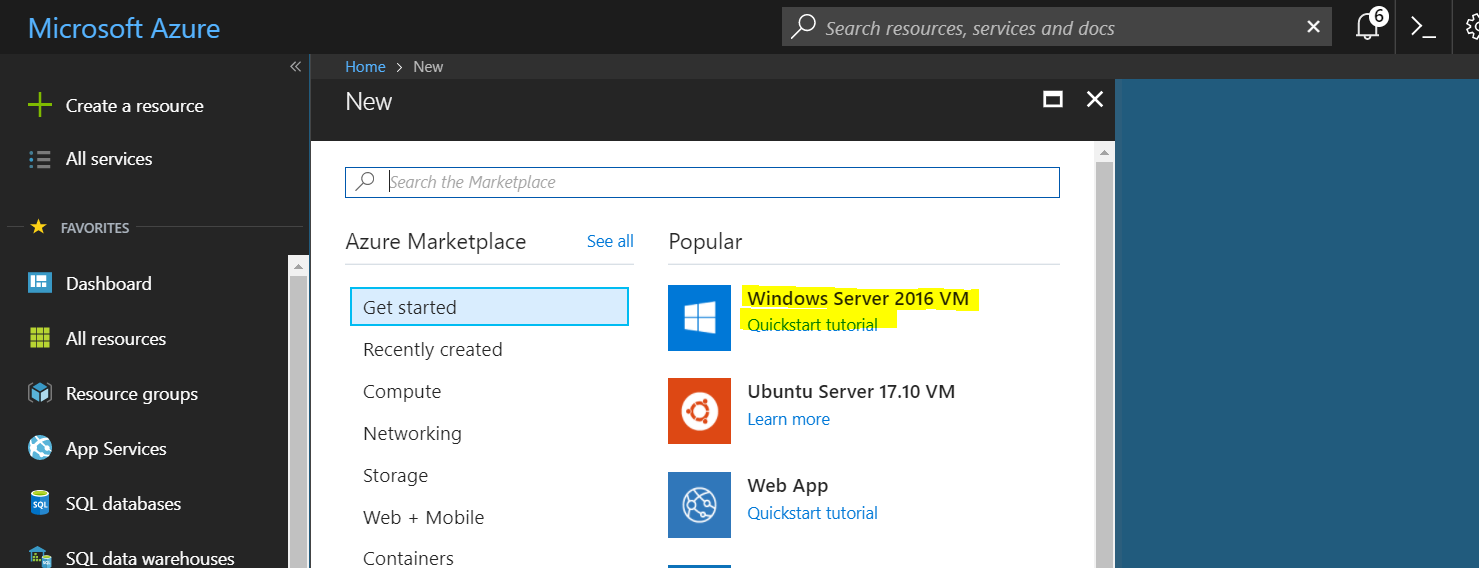
**Firewall Rules**

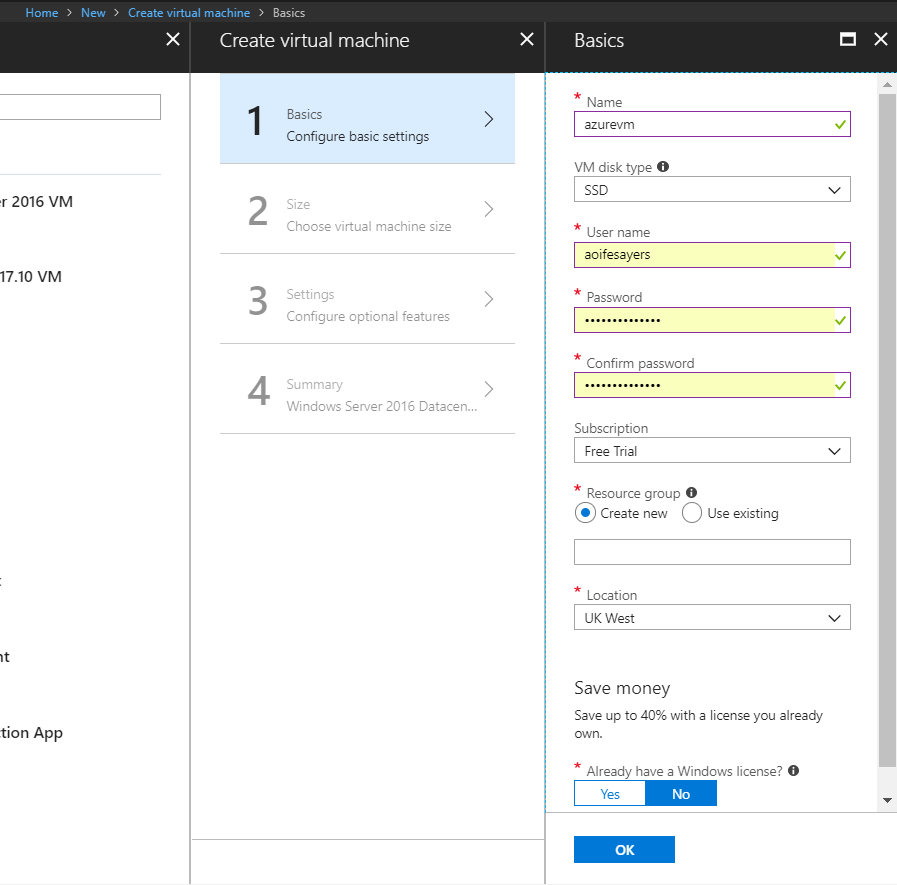
The following firewall rules/ports must be defined to allow Octopus Deploy communicate with the server

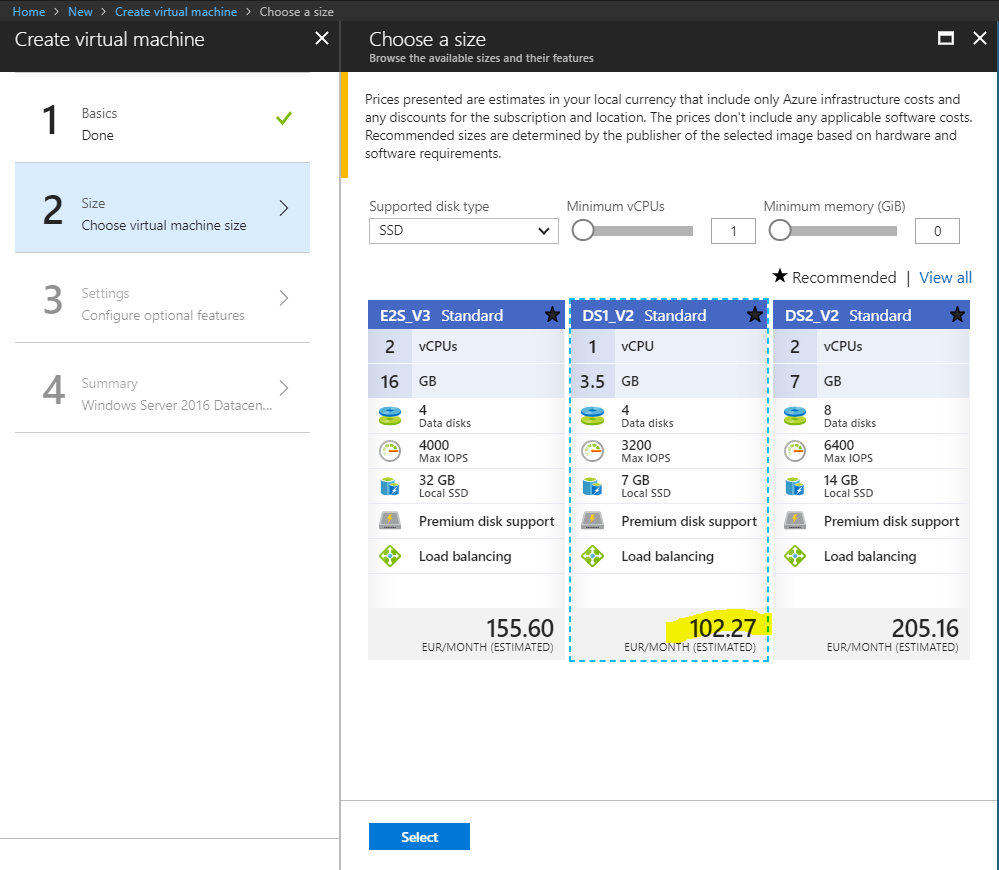
* RDP - 3389
* HTTP - 80
* HTTPS - 443
* Listening Tentacle - 10933
* Polling Tentacle – 10943

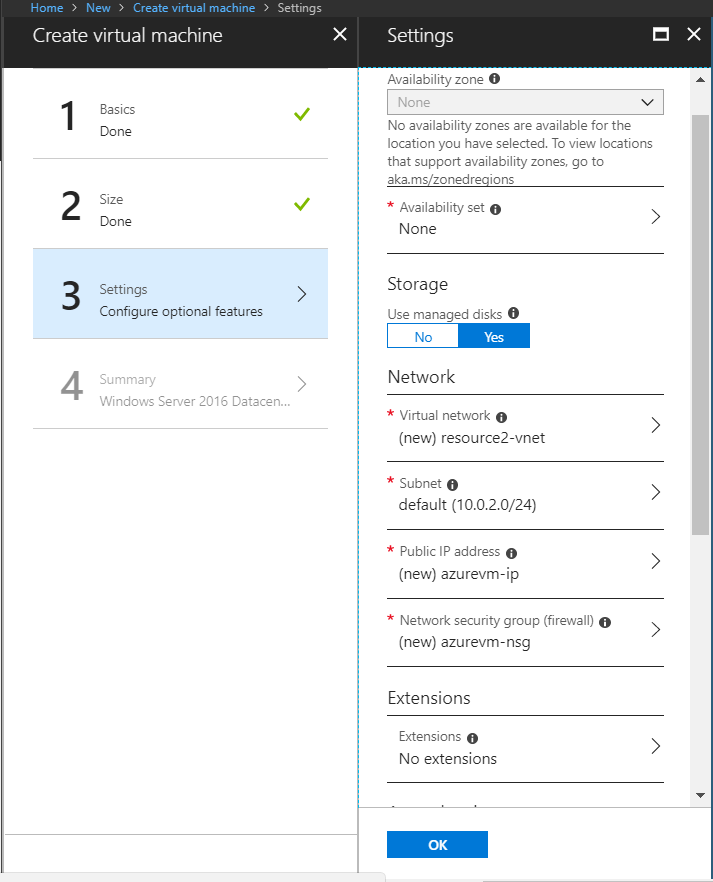
## Azure Windows Server VM

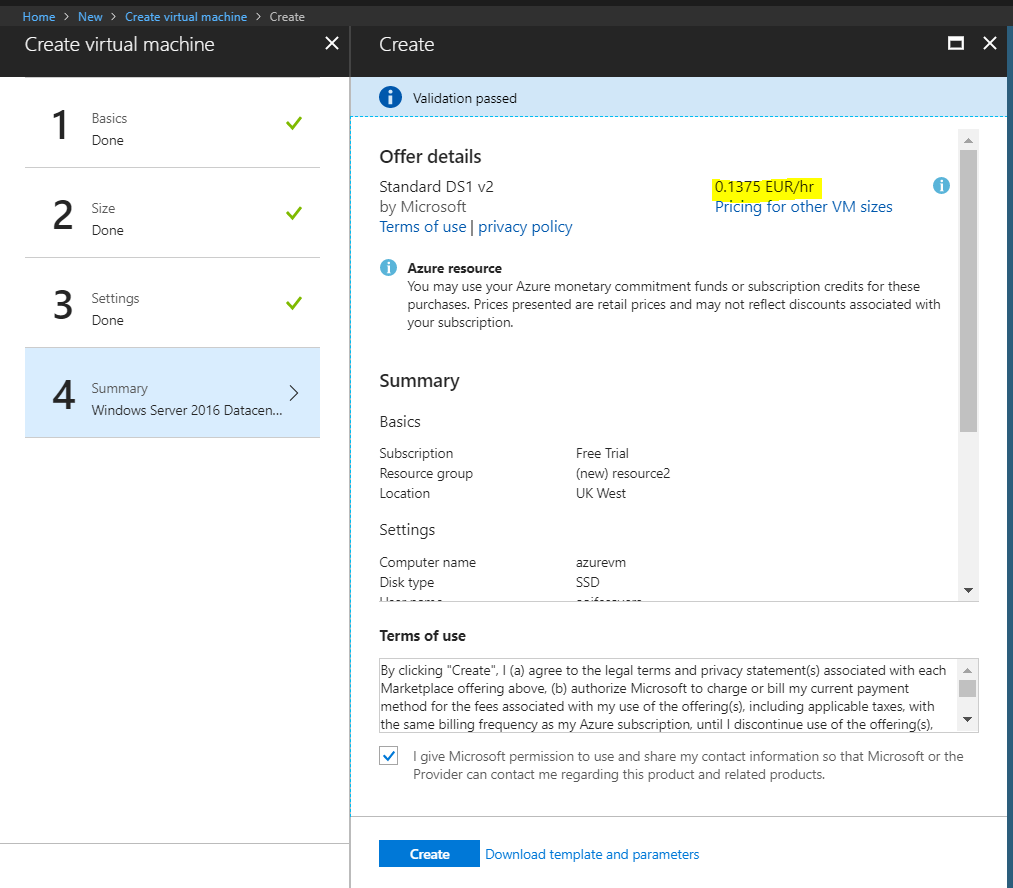
Azure provides a free trial with €170 complimentary credit. To set up an Azure Windows Server VM, select create a resource and search for Windows Server 2016. Select Windows Server.



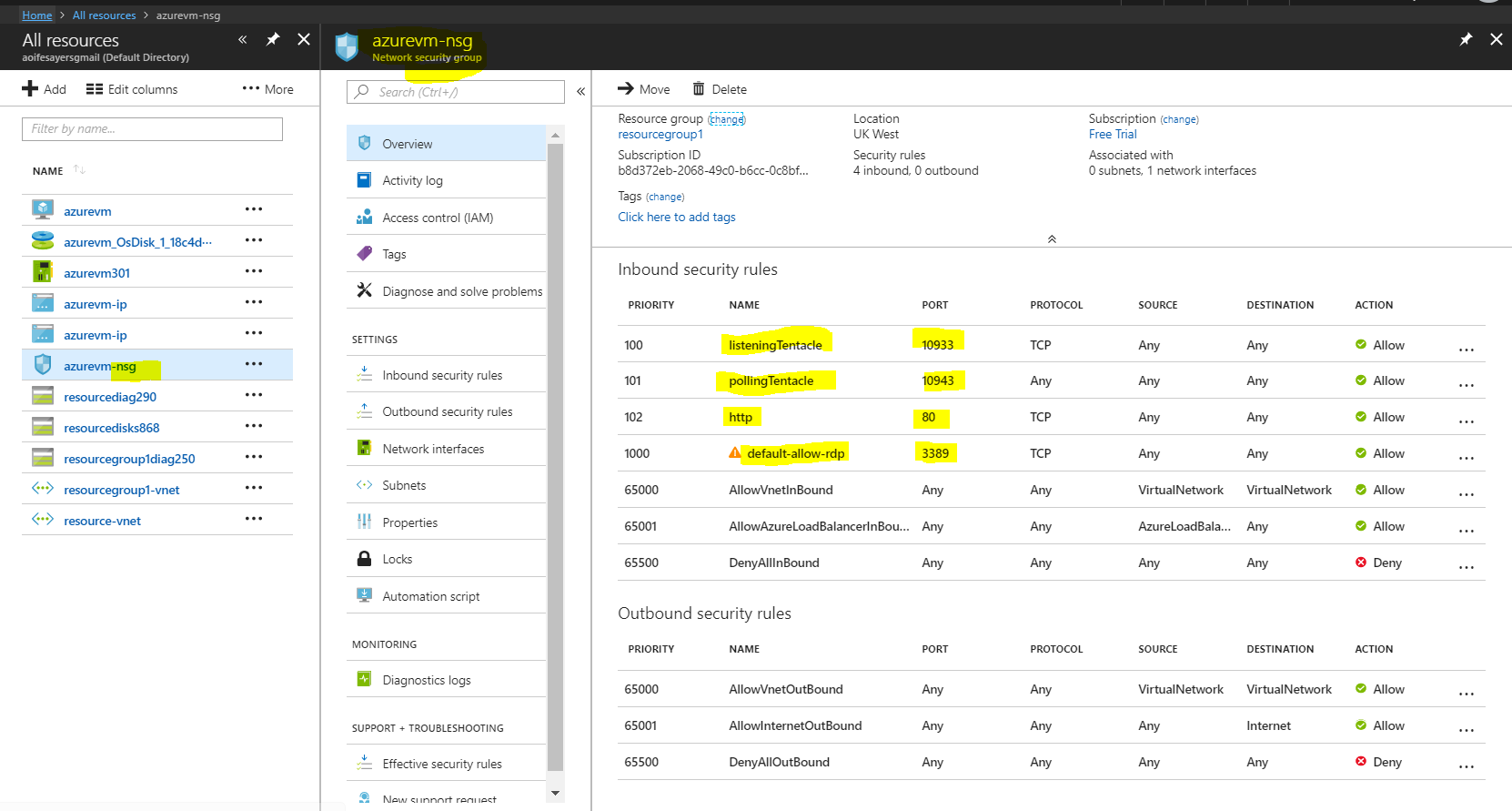




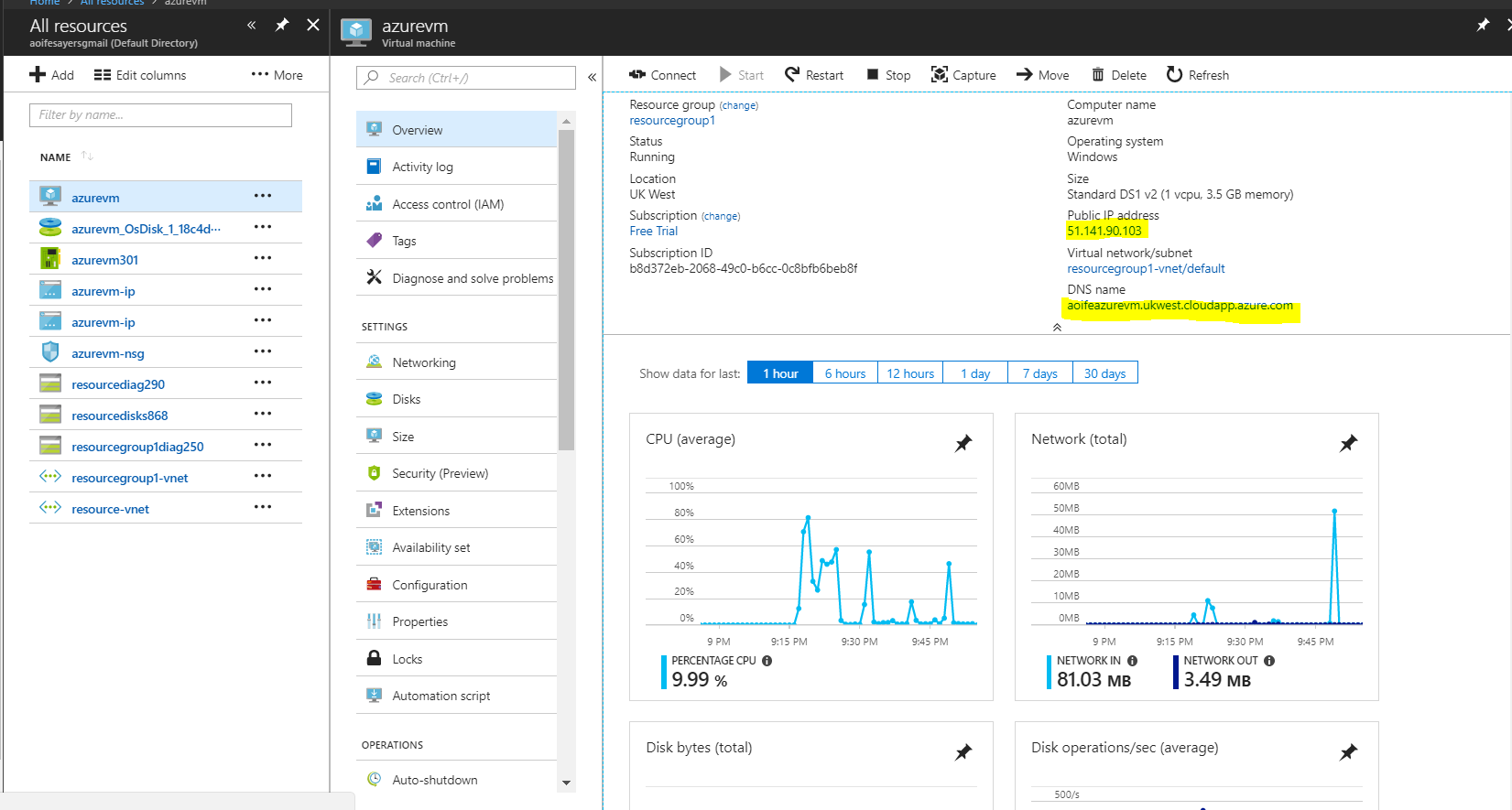




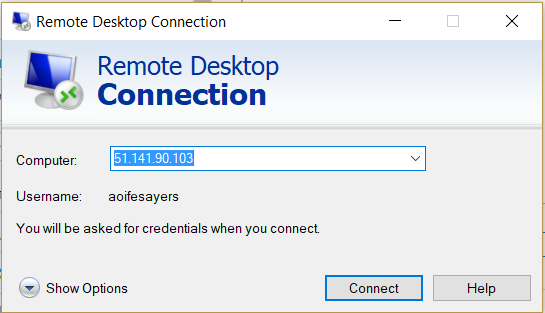
To configure the NSG (network security group), go to all resources in the menu. Select the nsg and add the firewall rules for the listening tentacle port 10933, polling tentacle port 10943, http port 80 and RDP (Remote Desktop Protocol) port 3389.



The Azure VM will launch and provide a public IP address and DNS name.

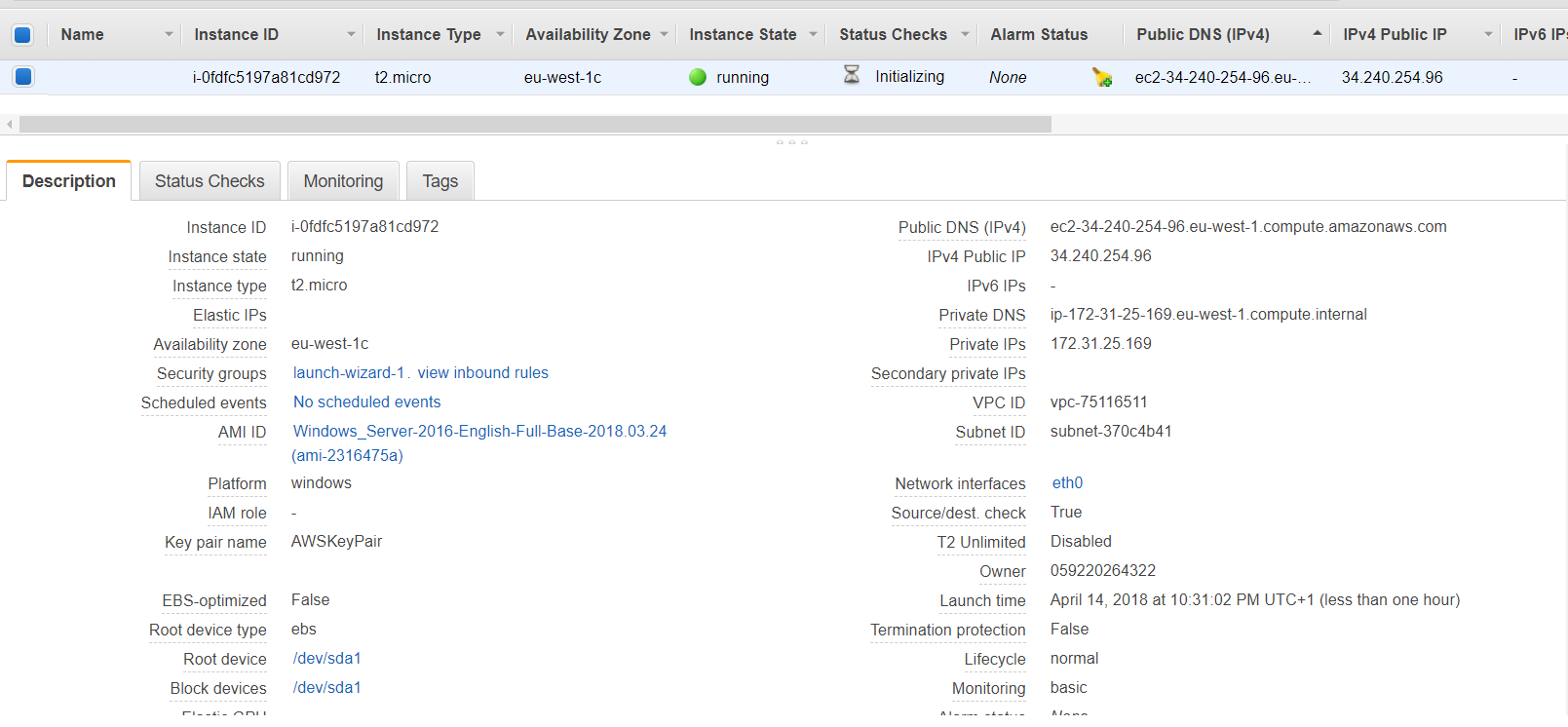


Use Remote Desktop Connection on Windows with the public IP address/ DNS name and the username and password configured when setting up the VM

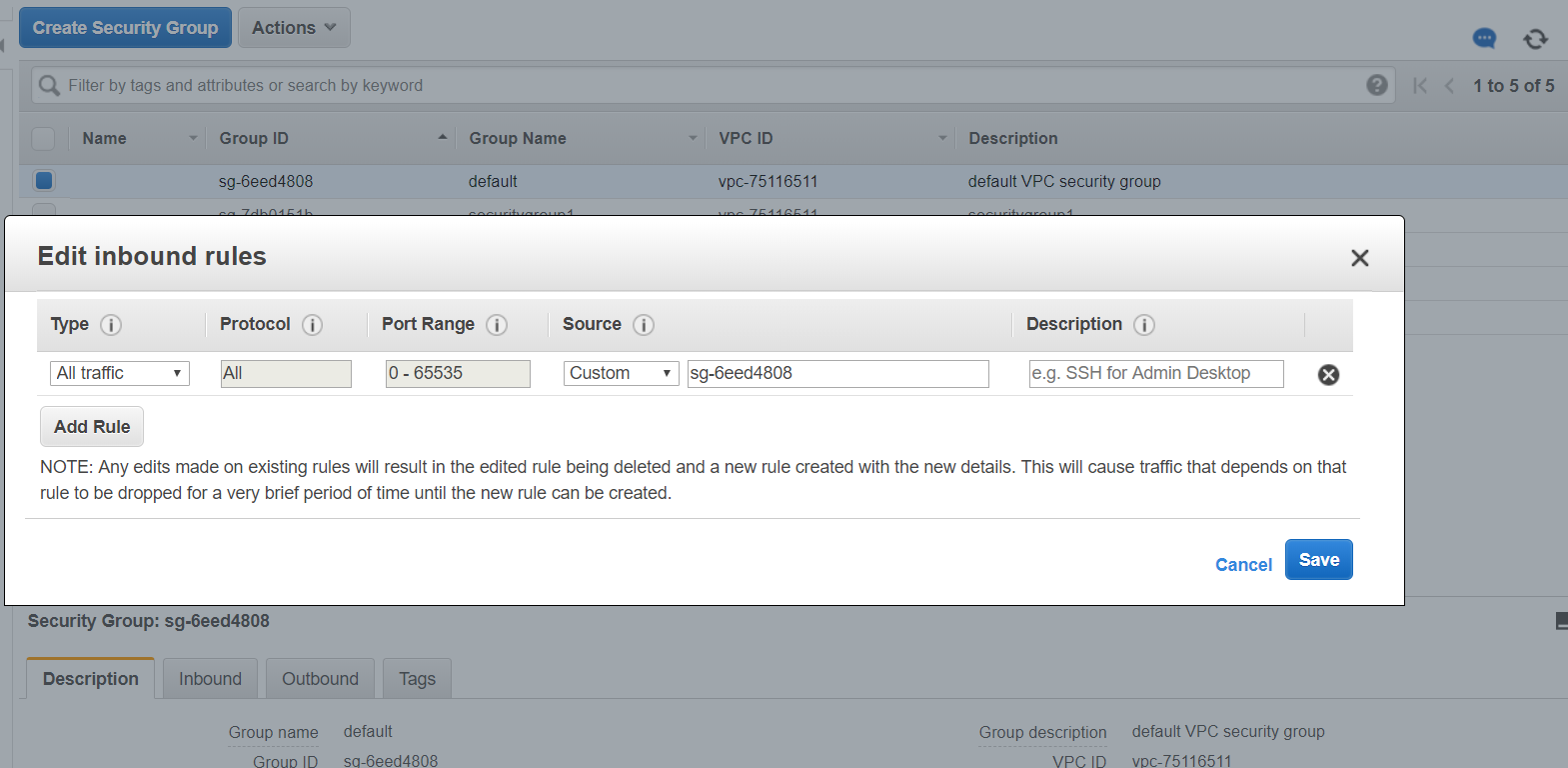


## AWS Windows Server VM

To set up a Windows Server VM select a Windows Server 2016 VM and select the t2.micro instance in the eu-west-tc availability zone. The instance should launch and provide a public DNS and IP address which you can use to connect to.



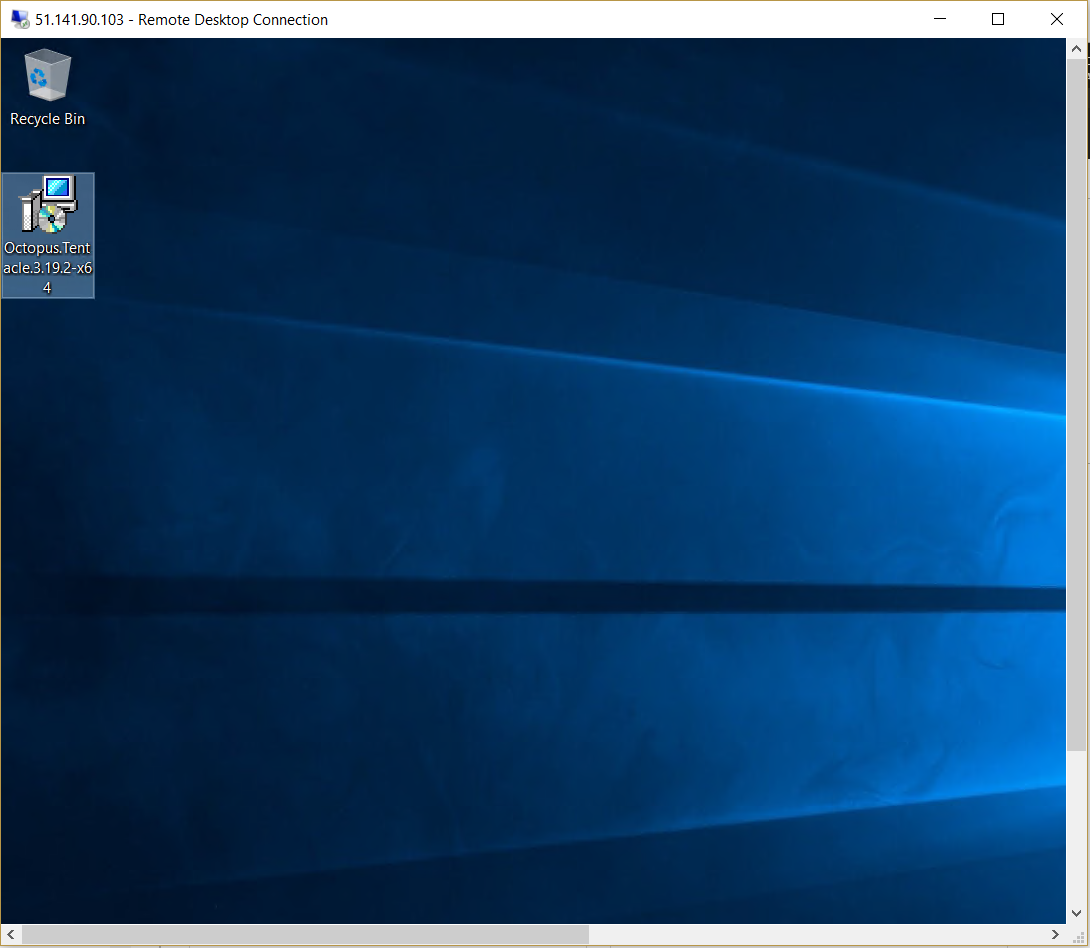
To edit the firewall, select the Security Groups inbound rules to configure the firewall. The inbound rules should be configured as follows

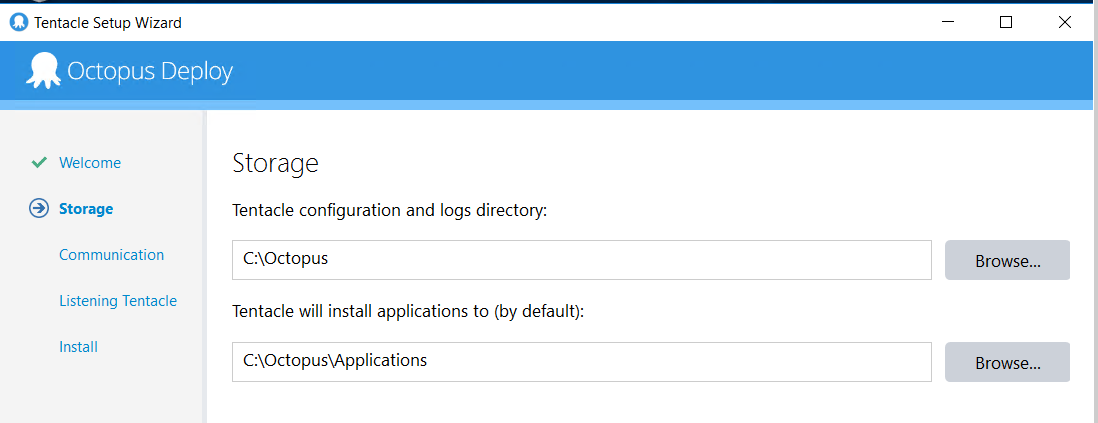


To connect via RDP on Remote Desktop connection, you must get details for the connection by pressing connect on the VM instance. A .pem file will download with the password and username for connecting.

## Setting up Deployment Target on VM

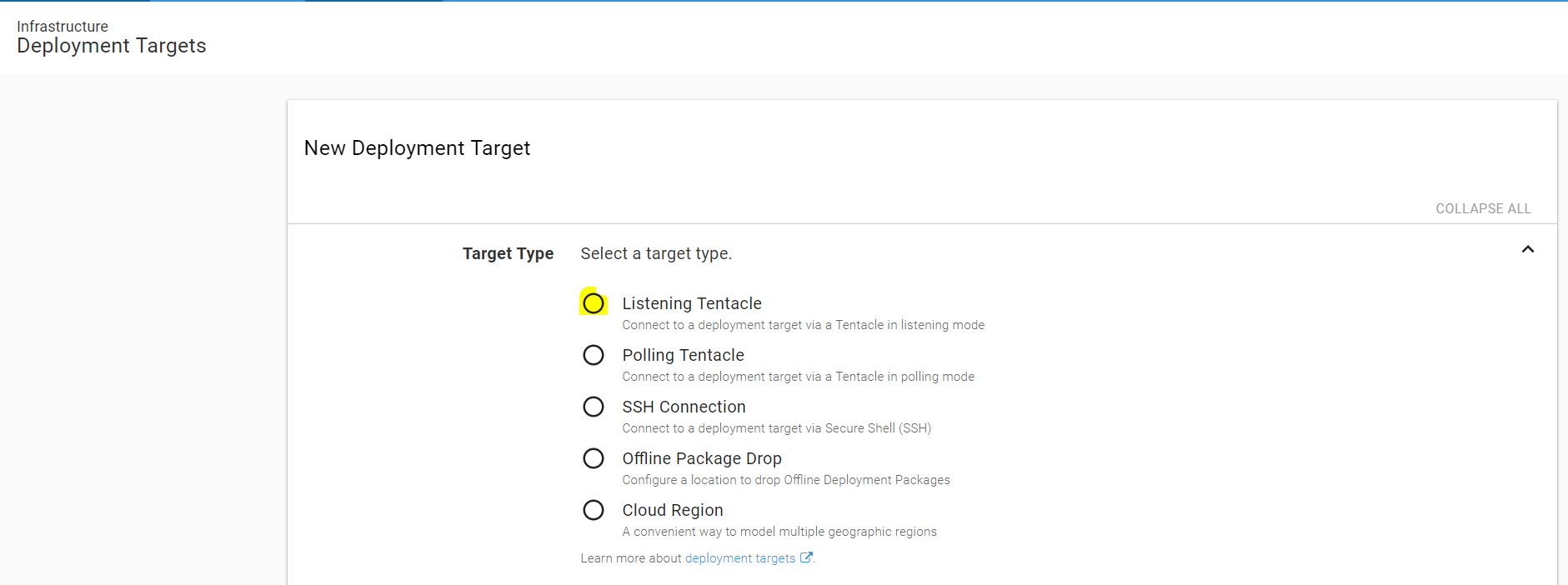
Open the VM, download <https://octopus.com/downloads> Tentacle Agent and install the msi file and install as demonstrated in the screenshots below

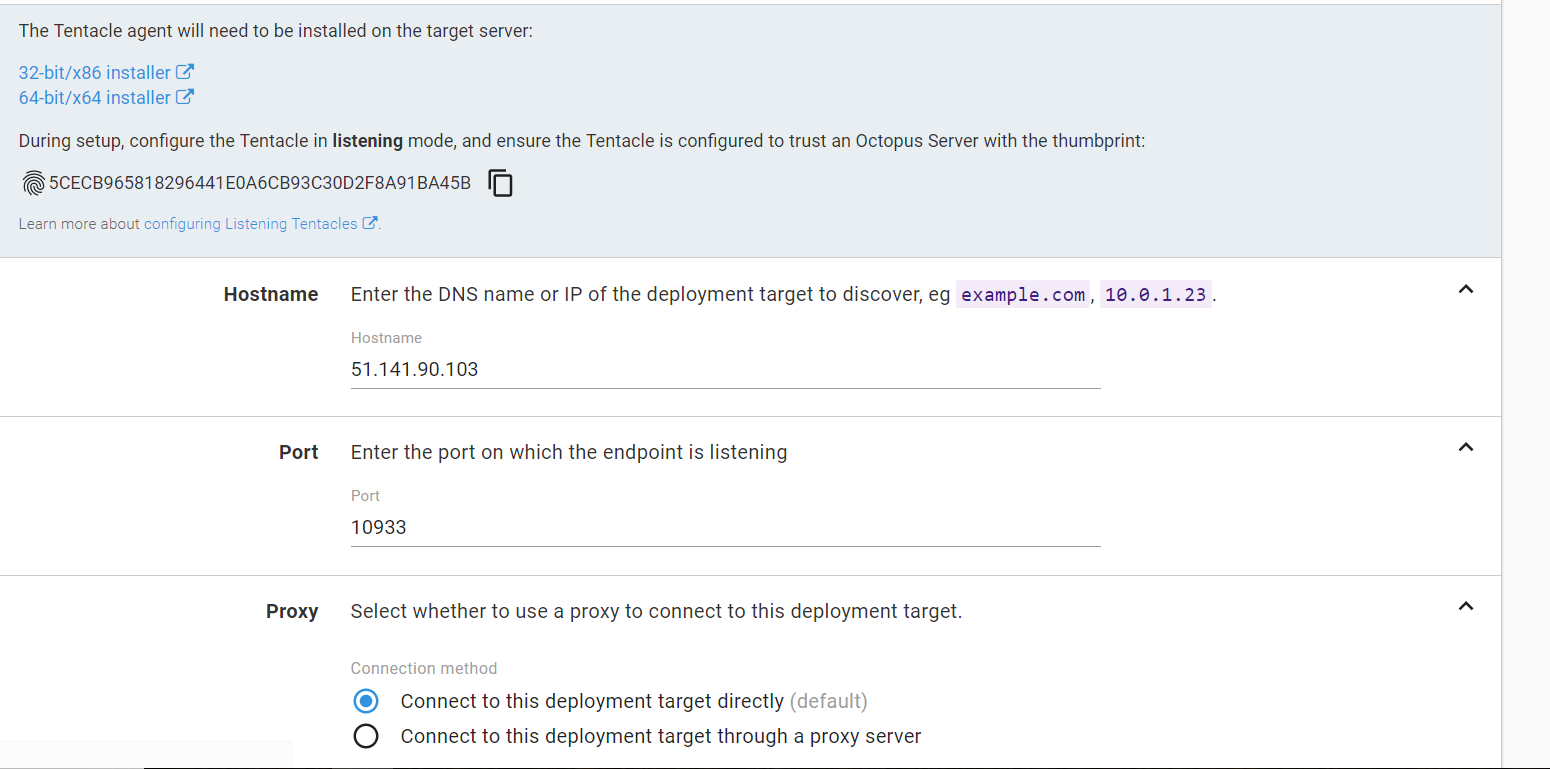




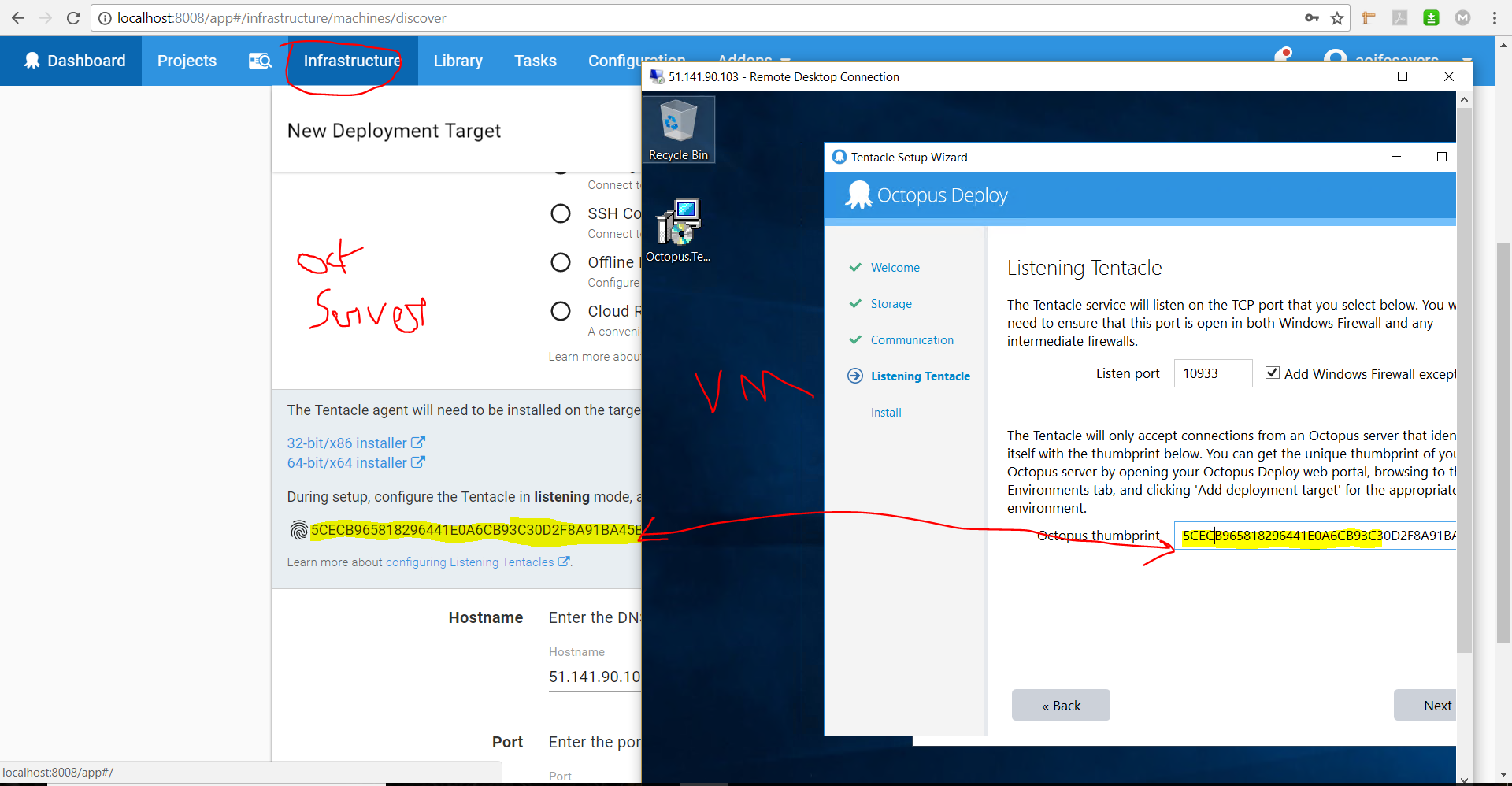


On the Octopus Server on your machine proceed to Infrastructure > New Deployment Target. Copy the thumbprint from the server and point it towards the VM’s IP/DNS





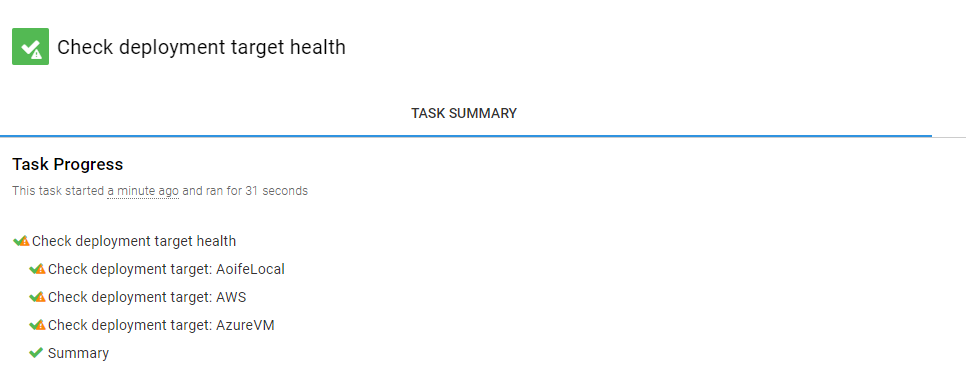
Paste the thumbprint from the Octopus Server into the Tentacle manager on the VM



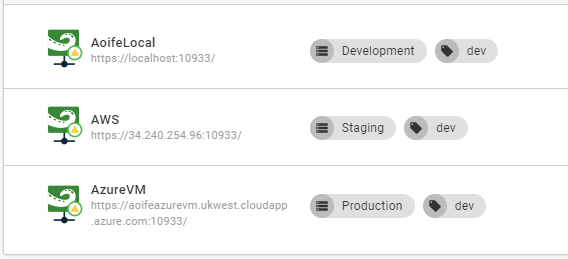
On the Octopus Server, check the health of the VM when the Tentacle is installed, check the health to ensure it’s installed correctly



After the 3 VMs are installed check the health of them all.



The three environments I have configured: Development, Staging and Production



# Tutorials Used

Teamcity Documentation: <https://confluence.jetbrains.com/display/TCD10/TeamCity+Documentation>

To fix team city build agent disconnected issue <https://stackoverflow.com/questions/12279081/teamcity-build-agent-disconnected> - go through log files

MSBuild command <https://msdn.microsoft.com/en-us/library/ms164311.aspx>

MsBuild missing parameters on Teamcity http://blog.chudinov.net/how-to-build-and-deploy-web-deployment-package-using-msbuild/

NUnit <https://www.codeproject.com/Articles/178635/Unit-Testing-Using-NUnit>

Git <https://rubygarage.org/blog/most-basic-git-commands-with-examples>

Octopus Deploy

<https://octopus.com/docs>

# Issues

Clashing ports – Teamcity Build agent & Octopus Deploy Server running on the same ports

Disconnected build agent – read through log files

Visual Studio & MsBuild

MsBuild missing parameters on Teamcity http://blog.chudinov.net/how-to-build-and-deploy-web-deployment-package-using-msbuild/

MSBuild path pointing to the wrong directory – had to edit the config file

Sonar Qube Scanner build step in Teamcity was not showing any results whereas the manual command was.

# References

A., S., 2018. *Most Basic Git Commands with Examples – a Welcome Introduction to Git.* [Online]   
Available at: https://rubygarage.org/blog/most-basic-git-commands-with-examples  
[Accessed 14 March 2018].

Git, 2018. *Git.* [Online]   
Available at: https://git-scm.com/  
[Accessed 14 March 2018].