# Enterprise Grade DevOps with OutSystems

## Intro

In the last recent years, I’ve been seeing **OutSystems** helping large enterprises developing their mobile and web solutions. In several vertical segments, helping their customers from building their public websites to internal mobile apps. For those who have never heard of it, let me briefly try to introduce it:

OutSystems provides an enterprise **Rapid Application Development (RAD)** platform. This platform allows you to create, deploy, change, and manage custom mobile and web applications delivered seamlessly across all devices. Nowadays it’s available as a cloud or on-premises solution with an open architecture.

Now, as a developer, I’ve had several healthy discussions about RAD tools, and I still prefer the broad options choosing technologies, languages, patterns, IDEs, but most importantly this days, the DevOps cycle and respective toolchain. Yes, there are multiple scenarios where RAD tools are the best choice, especially if they deliver as promised.

Large enterprises usually have complex information systems. From legacy technology still running in critical workflows, heterogeneous environments and teams. In the other hand, there are also the innovative teams, with modern development environments and a cool stack running in production, full monitoring and telemetry feeding the modern DevOps cycle put in practice.

So far, I’ve witnessed this large enterprises not being able to have a common DevOps cycle between Outsystems Apps and Non-Outsystems Apps. What happens when you have a large project where in a single release you must sync legacy Line of Business apps (running in your UNIX and windows servers), rest APIs, cloud backends…. AND Outsystems Apps (Mobile and Web)?

Some invest some time and resources, but not all.

I’ve been working with Visual Studio platform for years, and the recent strategies have brought a very interesting value proposition. For me top highlights:

* Cross platform IDE and code editors (love VS Code by the way!)
* A full modern DevOps cycle toolchain – plan work, build, test, release and monitor
* Completely open to Open Source tools (ex: Jenkins or Kubernetes). If you like your tools you can integrate with Visual Studio, and continue to use it.
* An extensible platform, bringing the community together with an awesome marketplace.
* A cloud based version (Visual Studio Team Services) with updates every 3 weeks! – yes, just read about Microsoft DevOps journey.
* An on-promises version (Team Foundation Server), that can have agents both in Linux or Windows.

I decided to contribute creating a Visual Studio extension to integrate with Outsystems platform.

You can read more details in the following sections, or skip everything to the walkthrough.

## Outsystems meets Visual Studio platform

Not being an expert in Outsystems Platform or development, I had to do some research around their current versions. They use model-driven approach to configure the app layers — UIs, data model, web services and/or APIs. Developers can still incorporate their own custom Java or C# code or libraries, and compose them as part of the model, as well as custom JavaScript (and CSS) for front-ends.

One of the Outsystems platform’s module is Lifetime. Lifetime is responsible for managing versioning, deployments and environments, hence managing the releases. And the best news is that, recently a Lifetime API (Swagger defined) was exposed allowing anyone to consume this API.

The Outsystems Integration is now available, and it starts by allowing any enterprise to create, maintain their DevOps cycle, while integrating with Outsystems platform. This extension focuses on:

* A Service endpoint in VSTS/TFS configuring the Outsystems platform;
* An Outsystems Release Task, allowing tag and/or deploy Outsystems Apps;
* Getting the best from VSTS/TFS and Outsystems Lifetime, for release management purposes;
  + Automatically versioning apps;
  + Complete trackability for every release;
  + Release definition cloning for multiple environments;
  + Schedule automatic releases and/or request on demand
  + An approval system for release executions

The extension was completely built using Typescript in Visual Studio Code, and using some ES6 improvements. If you like Promises and Async/Await, this is a good example.

I’ve decided to make it open source and available on Github, so everyone can collaborate, send feedback, or register bugs!

## Conclusion

Hope to see enterprises, small development teams using this extension, so they can have a single common DevOps cycle between Outsystems Apps and non-Outsystems Apps.

It will be easier to build and manage an Enterprise-grade flow with higher developer productivity targeting Multi-channel and heterogeneous developments, while reducing operational costs.

This is one of the biggest advantages of current Visual Studio platform – to easily integrate other tools and make operations/developer life easier.

In the future, we expect that Outsystems platform’s surface of integration will increase, allowing trackability from repositories, work planning (agile or scrum), builds or monitoring. For now release was the most relevant use-case, and it can be done in the cloud (hosted agents) or on-promises agents (linux or windows). It should cover most of the scenarios.

## Walkthrough

Here I will deep dive into Outsystems Extension available for TFS/VSTS, showing how to install, configure and use. I’ll try to list other possible scenarios that can be explored with this extension.

For this walkthrough, I’ll be using a VSTS subscription and my Outsystems platform is running in my Azure subscription.

Installing any Visual studio extension is extremely easy and fast. Just go to the Visual Studio marketplace and search for **Outsystems Integration**, or try this direct [link](https://marketplace.visualstudio.com/items?itemName=joalmeid.OutsystemsExtBeta).

Install the extension on you vsts subscription.

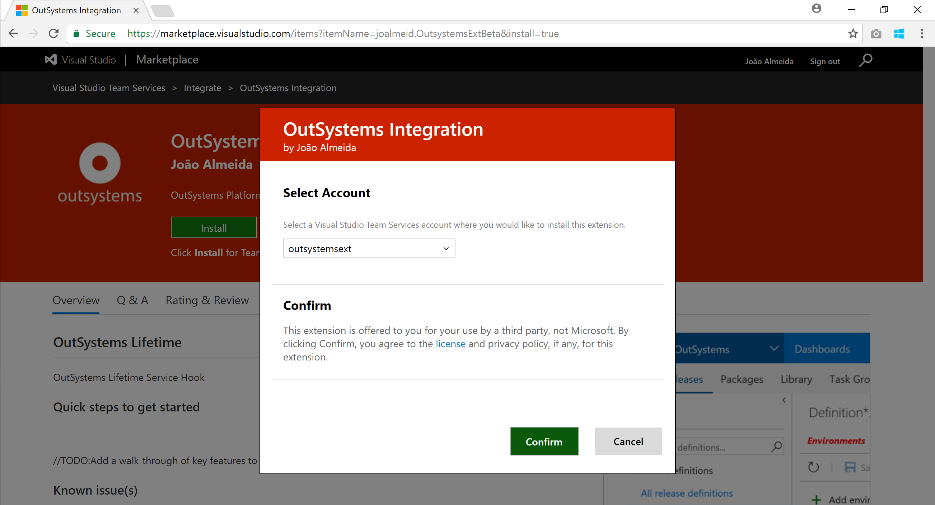


Figure 1 - 'Outsystems Integration' extension install

Now let’s configure the Outsystems Platform in your VSTS subscription, so we can act on it. This process is usually done by Service End-points. Let’s create a Service End-point by inserting some information about our OutSystems platform.

**Name**: A name for service endpoint (ex: myOutsystems).

**URL**: Your Outsystems LifeTime Deployment API endpoint

**StrictSSL**: Restricts communication between Visual Studio Team Services and your Outsystems platform to always use encrupted channels (SSL). Recommend.

**ApiToken**: Your Outsystems service account Your API Token. For additional information, check [Outsystems REST API Authentication documentation](https://success.outsystems.com/Documentation/10/Reference/OutSystems_APIs/REST_API_Authentication).

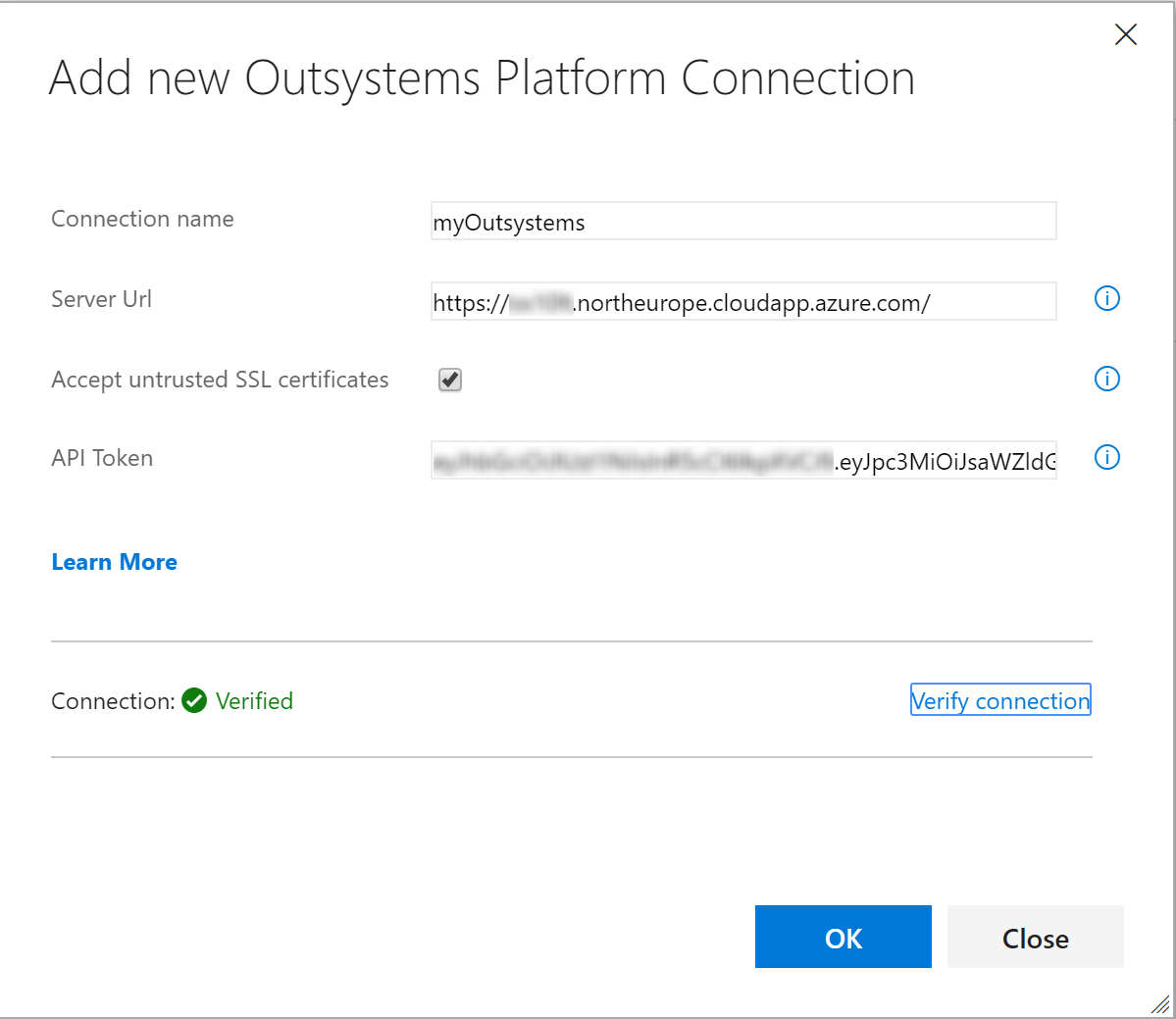


Figure 2 - Registring a VSTS Service endpoint for Outsystems Platform

You will be able to verify your connection, directly from this dialog, by clicking the ‘Verify connection’ link.

We’re ready to start building our Release definition plans, including Outsystems Apps. In your Releases, create a new Release Definition Plan. Here, you can start by either using a template available that may suite one of your apps, or change with an **empty process**.

|  |
| --- |
| Note: In my Outsystems Platform I had three environments configured: **Development**, **Test** and **Production**.  I used [Outsystems Service Studio](https://www.outsystems.com/platform/diagrams/service-studio/) focusing in two applications – X01 Darts(web) and X0M Darts (Mobile).    Figure 3 - Outsystems Service Studio |

Name your first environment, and also the Release Definition.

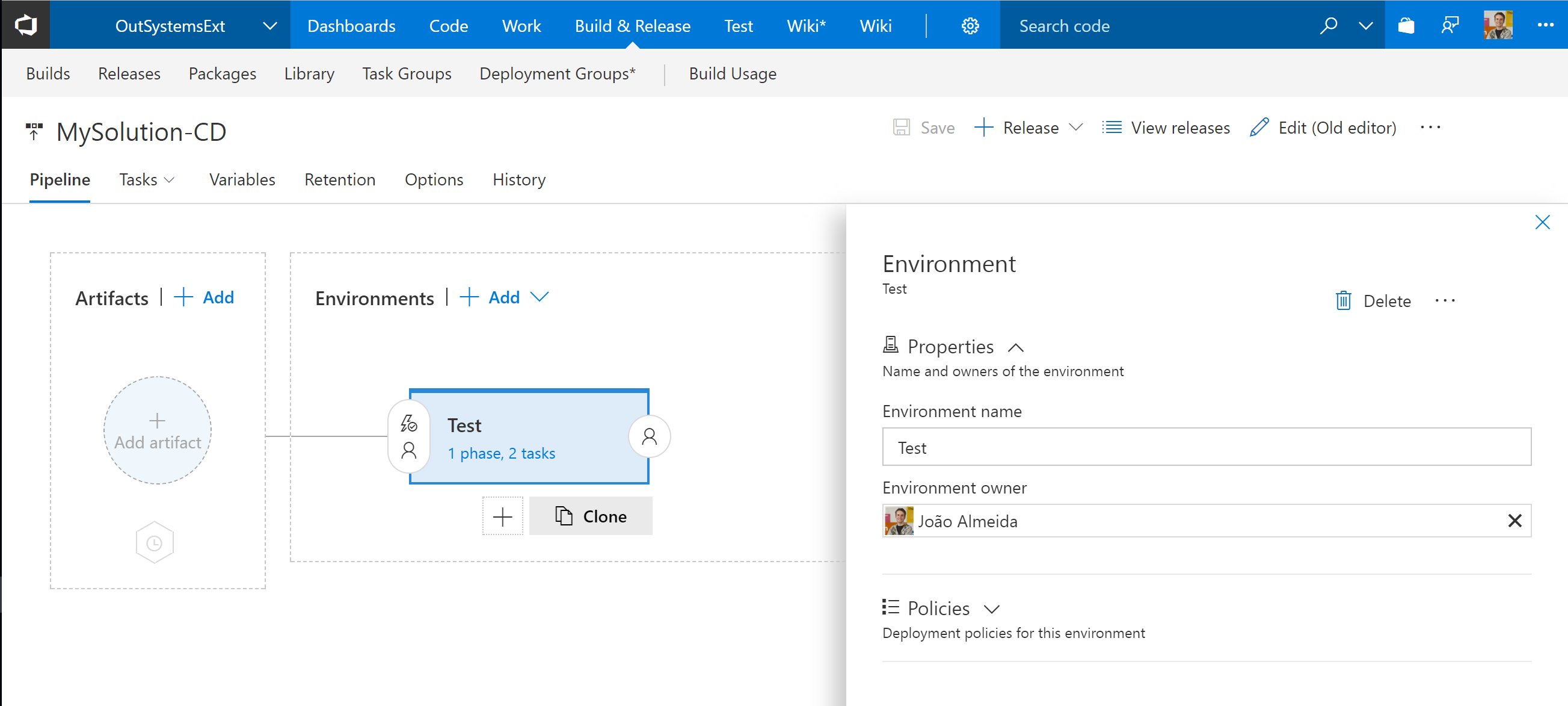


Figure 4 - Naming Release Definition environments

Next, we’ll dive into the first environment and define tasks the engine will execute. Add a Outsystems Release Task. Press OK, and let’s configure the task itself.

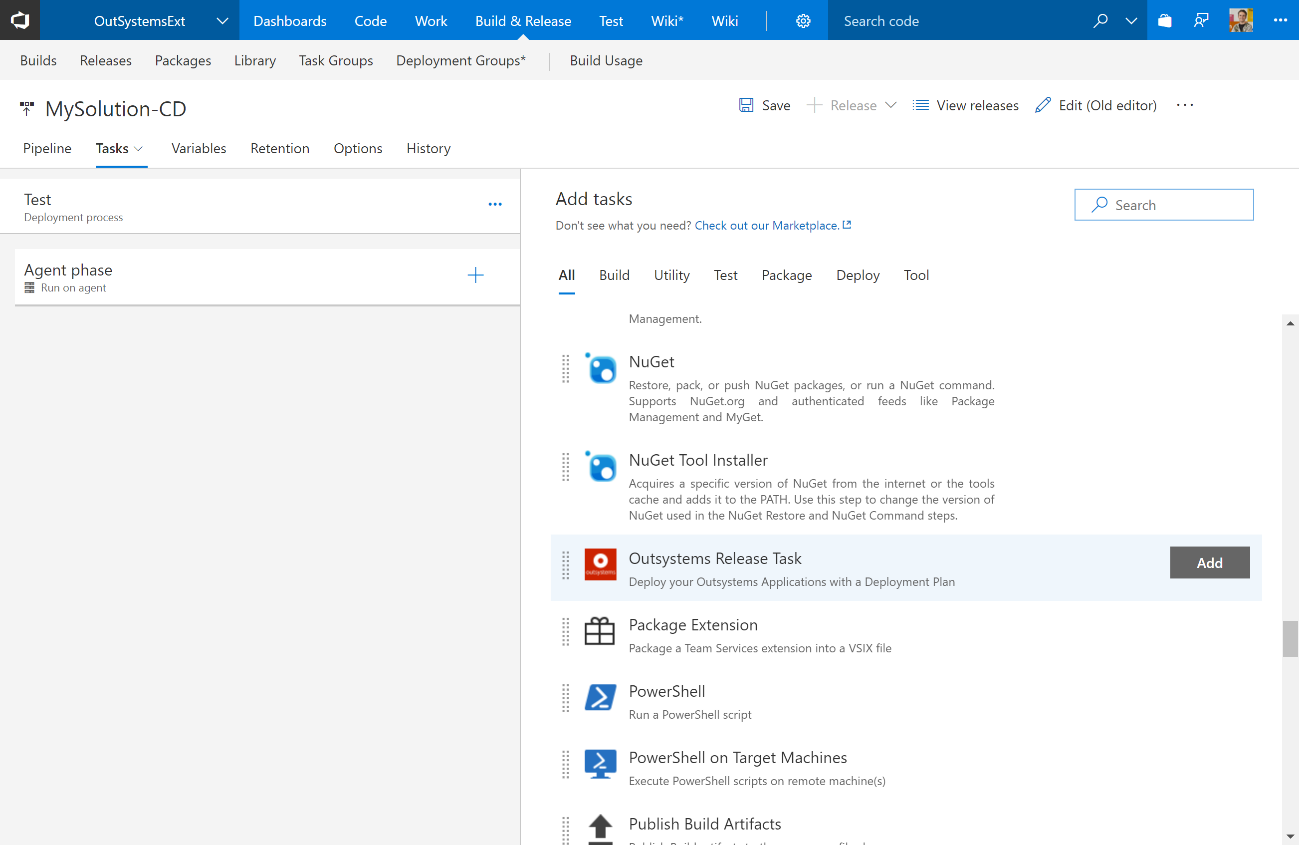


Figure 5 - Adding the Outsystems Release task

Now it’s time to configure our release task. Here we’ll specify information about the release process we want Outsystems Lifetime to execute.

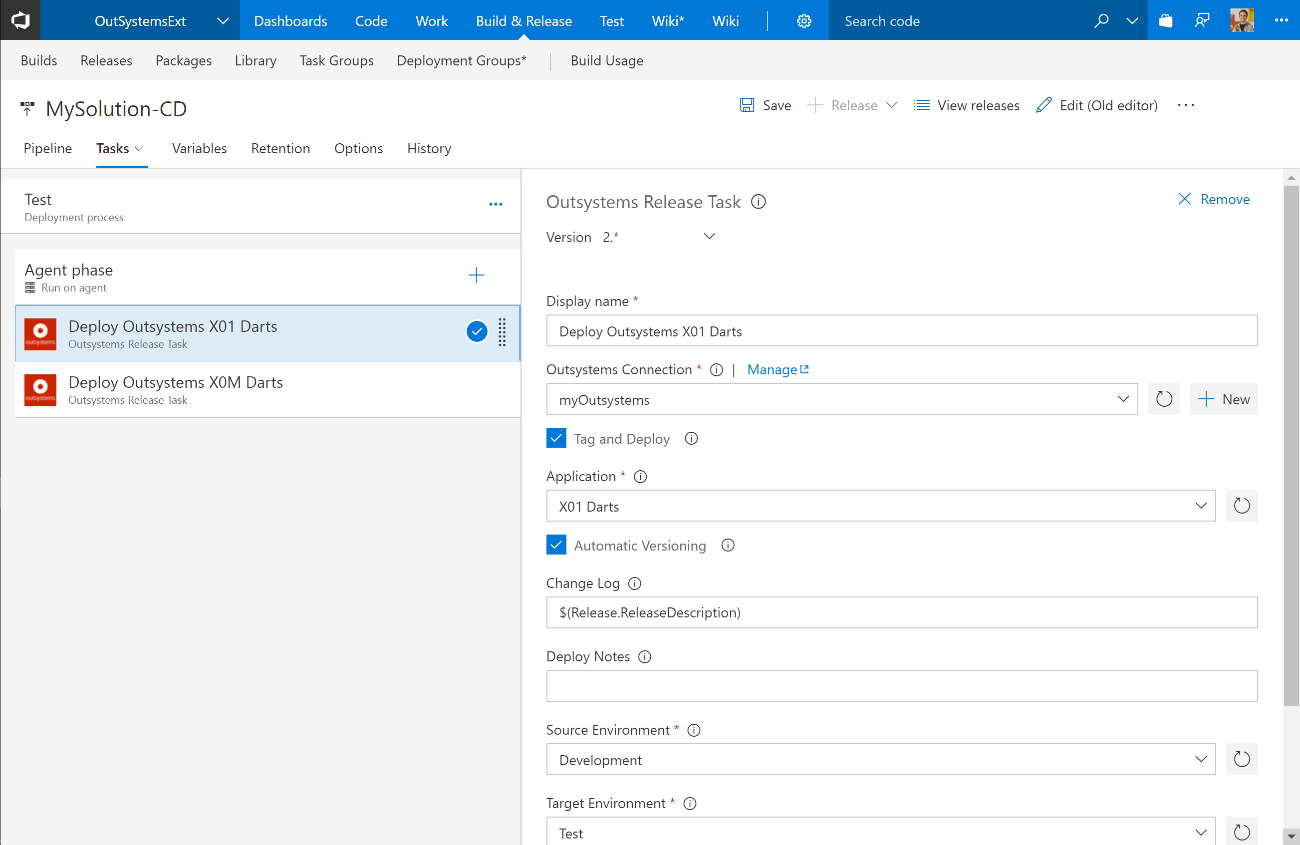


Figure 6 - Outsystems Release Task configuration form

In this example, I’ve already specified the Outsystems Connection, the Application, ChangeLog, Source and Target Environment:

|  |  |
| --- | --- |
| **Field** | **Description** |
| Outsystems Connection | List of your configured Outsystems Service Endpoints |
| Tag and Deploy | If we desire the application to be Tagged and Deployed |
| Application | List of the applications available in the Outsystems Platform. I’m deploying X01 Darts app. |
| Automatic Versioning | The extension will automatically version your release. Version formats is supported as [Major].[Minor].[Patch] |
| Change Log | Every release, can be associated with a Change Log. In this example, I’m stating the VSTS Release Description to be the release change log. |
| Deploy Notes | You can also add additional deployment notes |
| Source Environment | Your source Outsystems environment. Here I’m picking the ‘Development’ Outsystems environment |
| Target Environment | Your target Outsystems environment. Here I’m picking the ‘Test’ Outsystems environment |

Now, you can either add new Outsystems tasks to deploy additional Outsystems apps, or you can configure deployment of non-Outsystems apps with all the release tasks necessary to meet your goals. Remember: you can clone tasks and this will make your life much easier.

Our goal is to create the ‘Production’ environment that will deploy apps between the ‘Test’ Outsystems environment and the ‘Production’ Environment. Because it’s also possible to clone environments, we’ll do exactly that: Cloning our ‘Test’ environment and renaming it as ‘Production’ will do it.

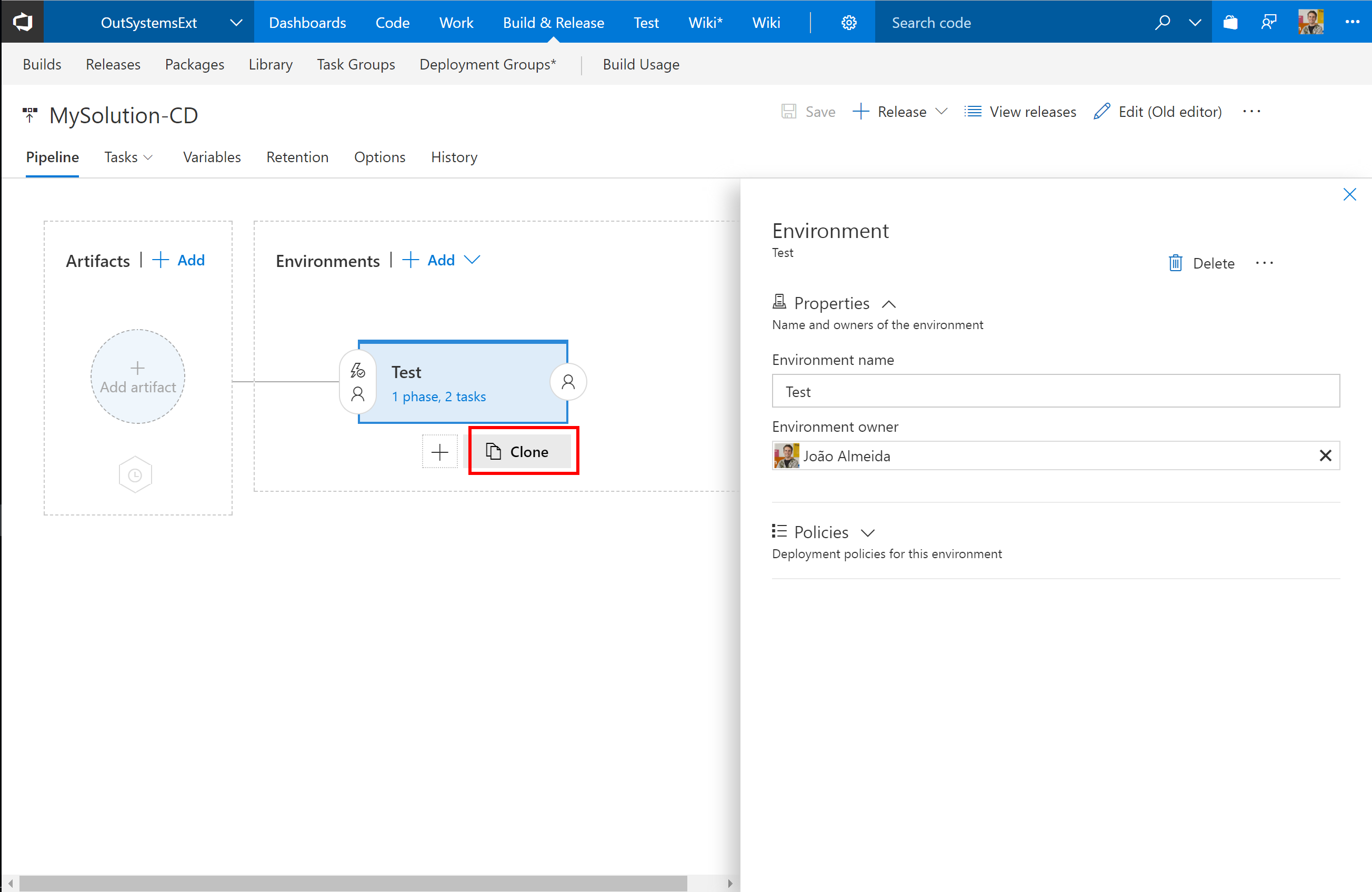


Figure 7 - Cloning the VSTS environments

Now we should go through all Outsystems Release Task and change the source and target environments:

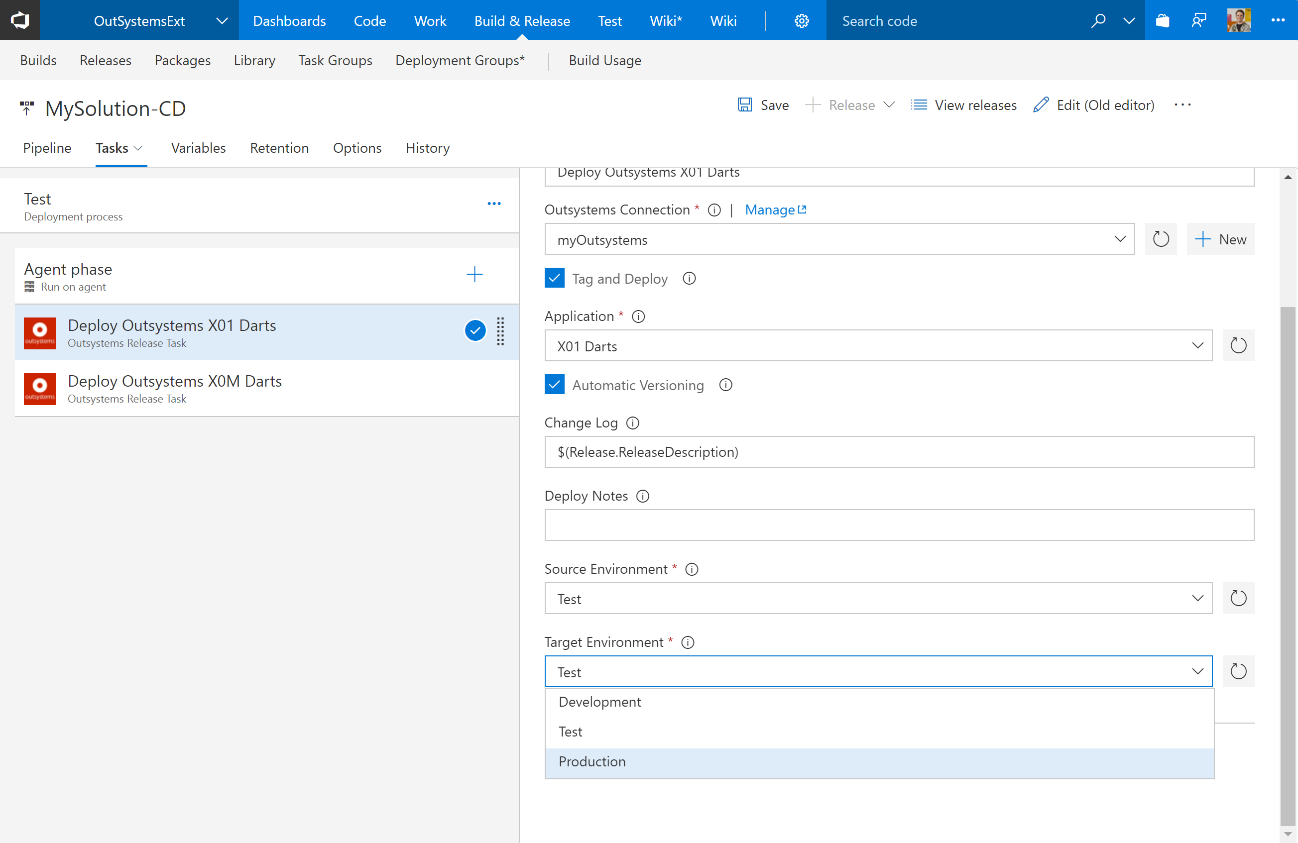


Figure 8 - Changing Outsystems environments in cloned tasks

Finally, one important concept to consider. When releasing apps from Outsystems ‘Test’ and ‘Production’ environments we’re not interested in tagging the applications. The desired behavior is to release any new version from ‘Development’ and ‘Test’ environments tagging it. However, when releasing from ‘Test’ to ‘Production’ we want the exact same version to be released. The way to configure this, is unchecking the ‘**Tag and Deploy**’ option. This configuration will always release the latest version from the source to the target environments.

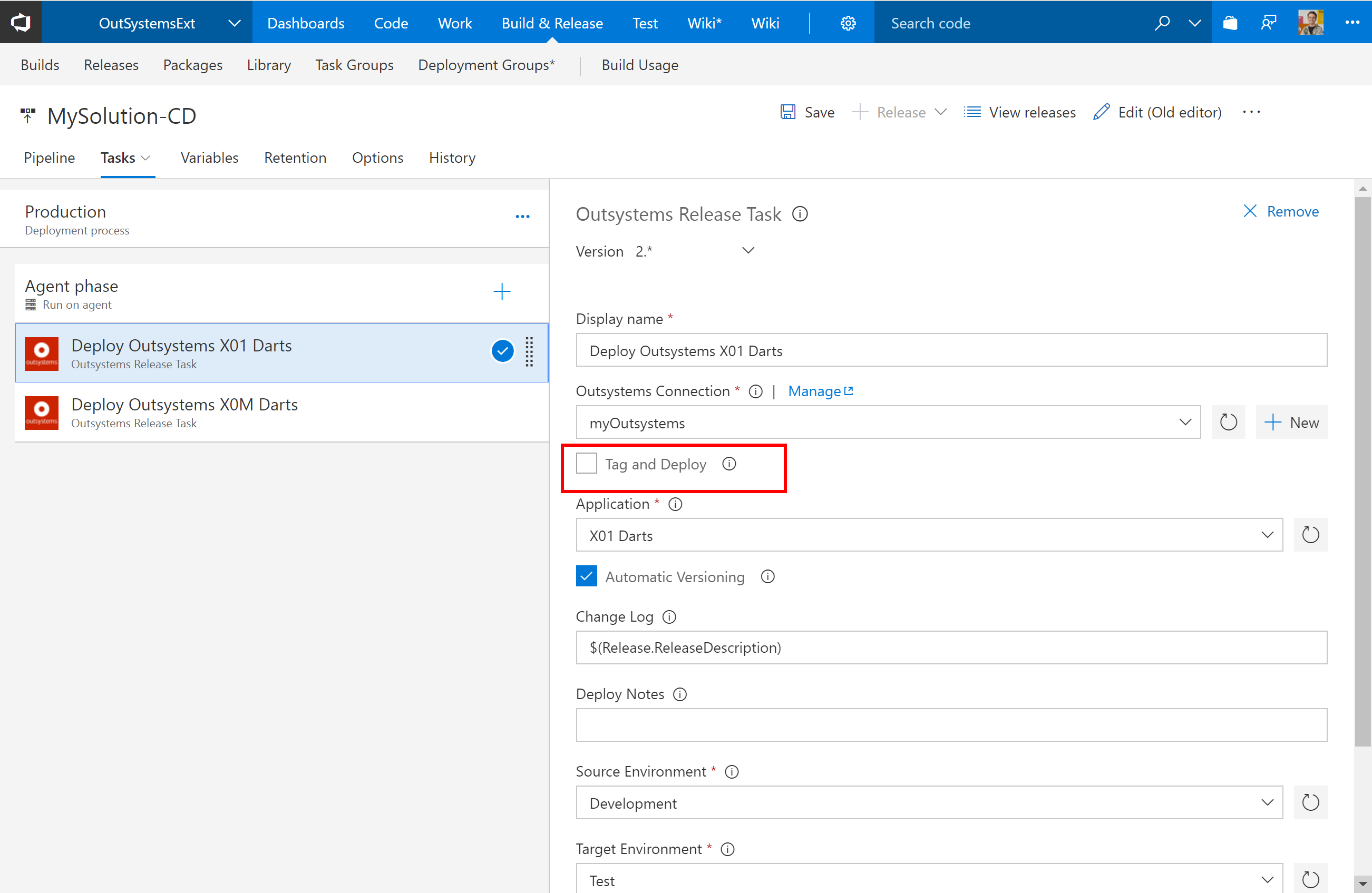


Figure 9 - Not using 'Tag & Deploy' to relase latest version available in source environment

We’re all set. Before creating a release and running it, let’s go through what we’ve accomplished:

* We’ve created a Release Definition to deploy two applications (X01 Darts and X0M Darts) through all the Outsystems Environments: ‘Development’ -> ‘Test’ -> ‘Production’;
* We configured two VSTS environments ‘Test’ and ‘Production’. It doesn’t make sense to have a ‘Development’ environment in VSTS, as the deployment is automatic from Outsystems Service Studio.
* In each VSTS environments, we used two Outsystems Release tasks, one for each application;
* The only difference in configuration was the ‘Tag and Deploy’ option in the ‘Production’ VSTS environment, as we don’t need to tag applications.
* We were able to use VSTS variables to provide information to Outsystems Platforms. We specified the ‘Change Log’ as the VSTS release description through the variable $(Release.ReleaseDescription).

Now let’s create a new Release and see execution. We can see complete delays of execution of all Outsystems Release tasks. Even the deployment log from Outsystems Lifetime, is retrieved and presented. This way, you can have full logging in VSTS and check recent or past outcomes without the need to jump into Outsystems Lifetime.

## C:\Users\joalmeid\AppData\Local\Microsoft\Windows\INetCache\Content.Word\demo.ostask.1stRelease.Running.png

Figure 10 - Outsystems Release Tasks execution

Now, imagine how you can manage your releases of Outsystems applications together with on-premises, cloud apps. If you need to deploy your existing Classic ASP websites, WCF services, Java REST APIs, Python websites or NodeJS REST APIs you can easily do it with VSTS. And this apps can now be direct dependencies from any Outsystems apps (mobile or web). You will not have to manage different release pipelines, toolchains.

And I close, with a final tip. Testing in VSTS can also be done upon builds or after releases. These can be specific tasks in the release definitions! Imagine having your tests running as part of your builds and releases, and target Outsystems Apps and you non-Outsystems applications. Sweet!