CI/CD Hello world

04/03/2017 • 10 minutes to read

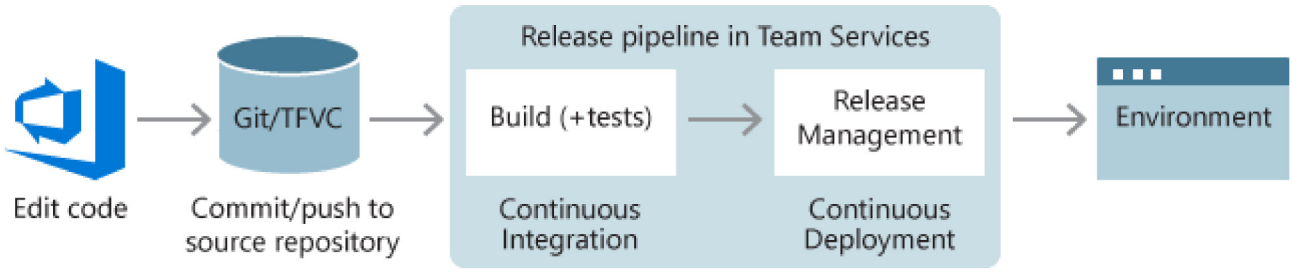
In this article

VSTS

What is continuous integration (CI)? What is continuous deployment (CD)? Why should I care? How do I get started using Team Build and Release Management?

Are any of these questions on your mind? If so, then you've come to the right place. We'll show you how to create a CI build that prints "Hello world" and then automatically creates a CD release that does the same. By the time you finish here, you'll see an end-to-end process run every time you push new code into your team project.

A quick introduction to CI/CD



CI means starting an automated build (and possibly running tests) whenever new code is committed to or checked into the team project's source control repository. This gives you immediate feedback that the code builds and can potentially be deployed.

CD means starting an automated deployment process whenever a new successful build is available.

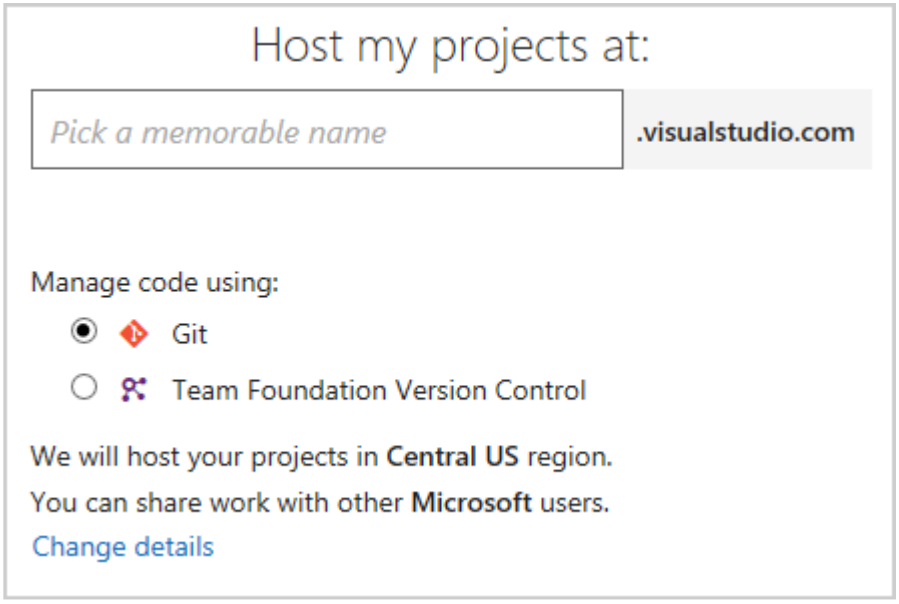
Together, CI and CD mean that any code changes you commit to your repository are quickly validated and deployed to a test server, a live web site, or wherever you need it.

Wanna try it?

Get set up with VSTS

> Do you already have access to a VSTS account and to a team project that has a Git repo? And do you already have permissions to create builds? If so, then you can skip to the next section. If you're not sure, it takes just a moment to create a new account and there's no charge.

1. Create a new account in VSTS.
2. If you're prompted, then sign in using your personal Microsoft account or your work or school account. (Need help signing up? See Sign up for VSTS.)
3. Create a Visual Studio Team Service account. Keep the option to use **Git** selected.

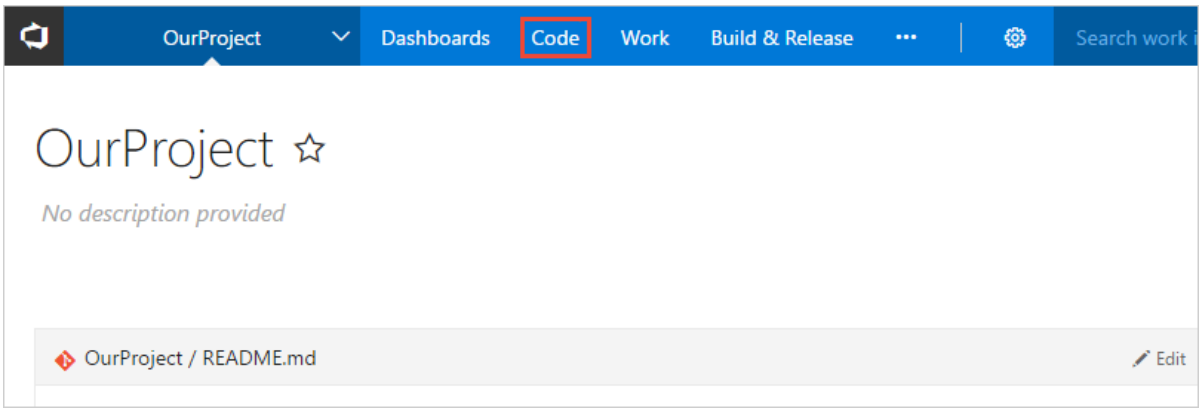


4. Select the option to **Initialize the repository with a README or gitignore**.
5. You see the home page for your first team project with a simple README.md file.

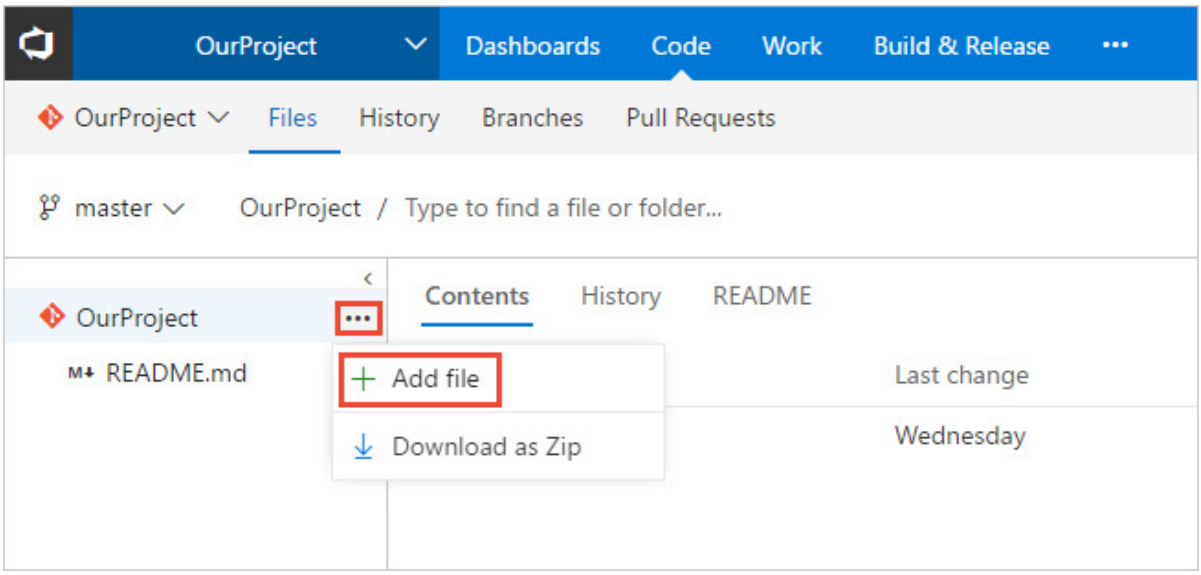
Add a script to your repository

Create a PowerShell script that prints `Hello world`.

1. Go to the **Code** hub.



2. Add a file.



3. In the dialog box name your new file.

```
HelloWorld.ps1
```

4. Copy and paste this script.

```ps
Write-Host "Hello world"
```

5. **Commit** (save) the file.

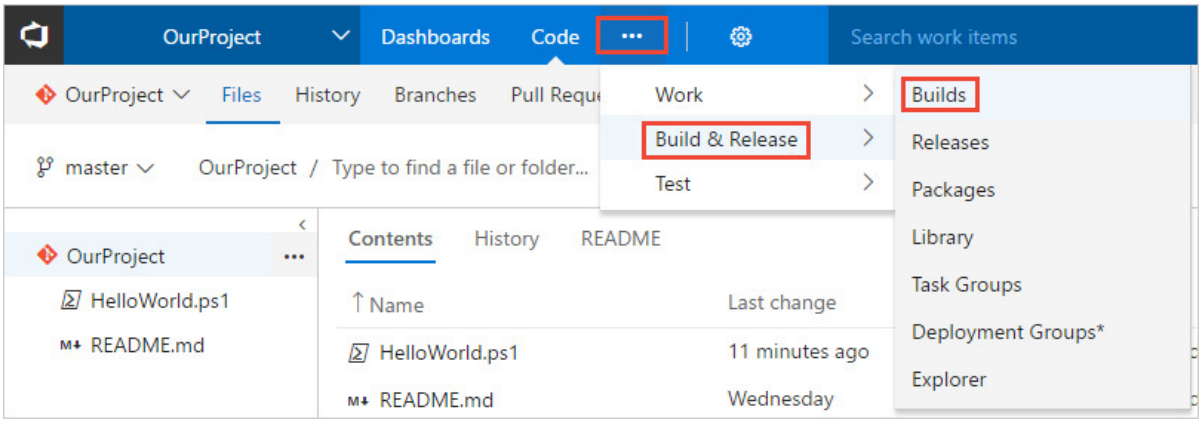
> In this tutorial, our focus is on CI/CD, so we're keeping the code part simple. We're working in a VSTS Git repository directly in your web browser.

> When you're ready to begin building and deploying a real app, you can use a wide range of version control clients and services with VSTS CI builds. Learn more.

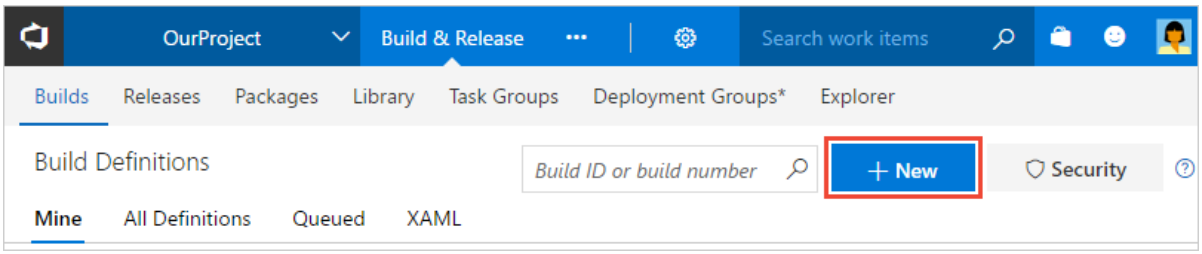
Create a build definition

Create a build definition that prints "Hello world."

1. Select the **Build & Release** hub in your VSTS project, and then the **Builds** tab.



2. Create a new definition.



3. Start with an **empty process**.

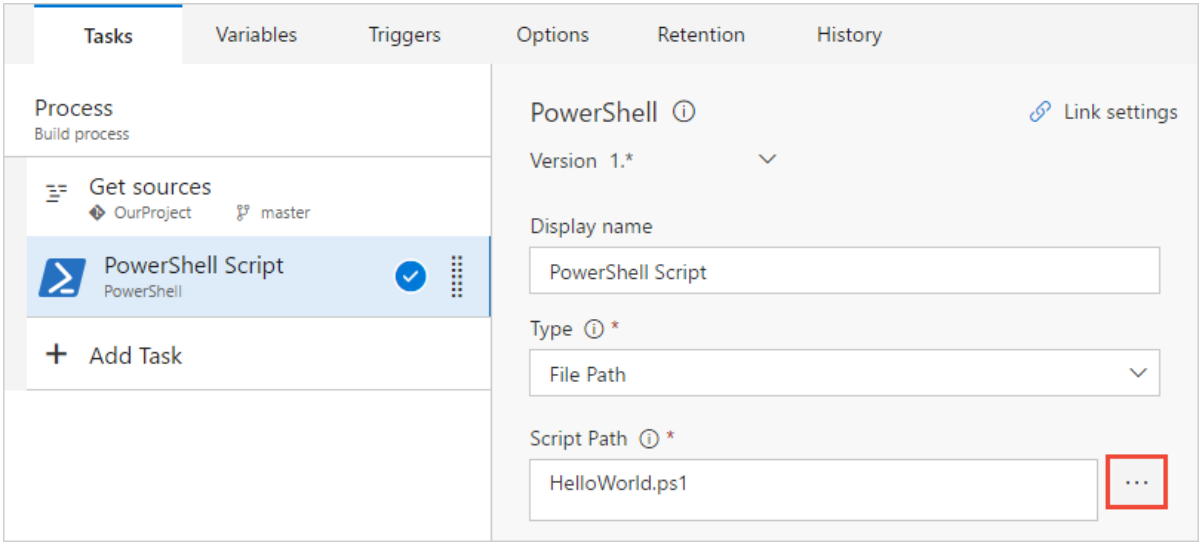
4. Click **Process** and specify whatever **Name** you want to use. For the **Default agent queue**, select **Hosted VS2017**.

5. Make sure that **Get sources** is set with the **Repository** and **Branch** in which you created the script.

6. On the left side click **Add Task**, and then on the right side click the **Utility** category, click the select the **PowerShell** task, and then click **Add**.

7. On the left side click your new **PowerShell** script task.

8. For the **Script Path** argument, click the ... button to browse your repository and select the script you created.



9. Click **Save & queue**, and then click **Save**.

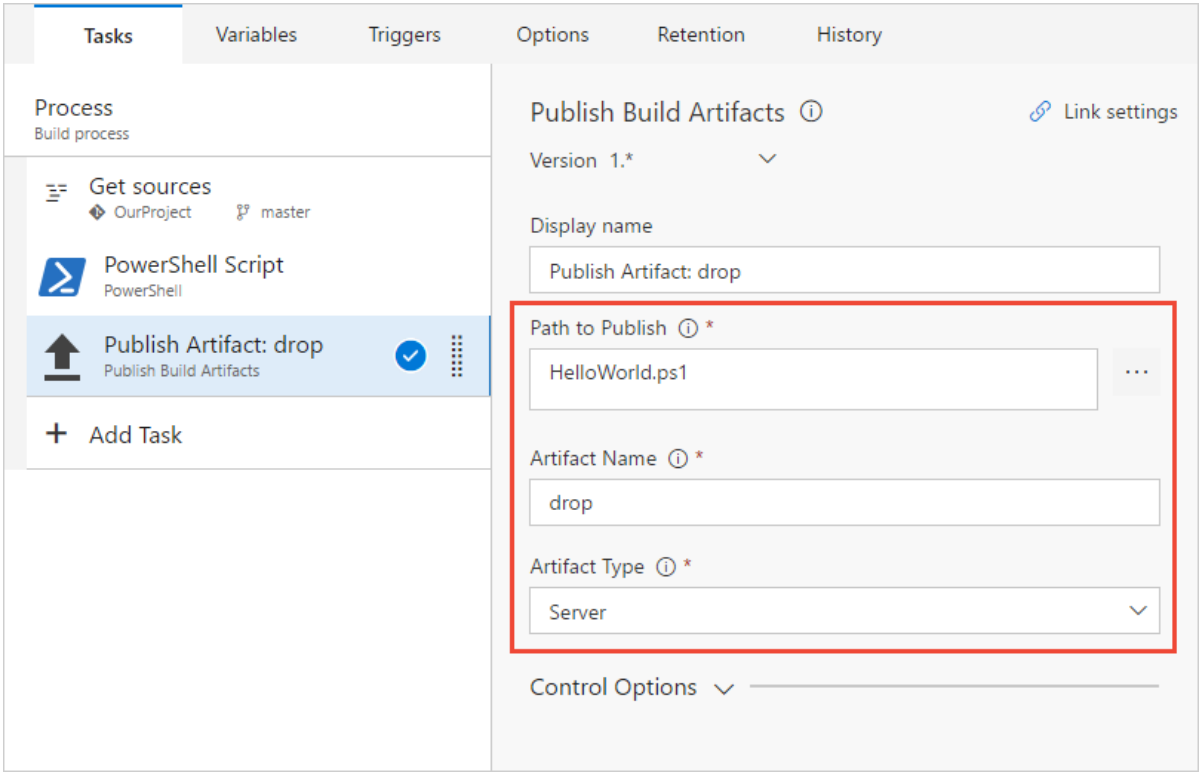
> A build definition is the entity through which you define your automated build process. In the build definition, you compose a set of tasks, each of which perform a step in your build. The task catalog provides a rich set of tasks for you to get started. You can also add PowerShell or shell scripts to your build definition.

Publish an artifact from your build

A typical build produces an artifact that can then be deployed to various environments in a release. Here to demonstrate the capability in a simple way, we'll simply publish the script as the artifact.

1. On the **Tasks** tab, click **Add Task**.

2. Click the **Utility** category, click the **Publish Build Artifacts** task, and then click **Add**.



Path to Publish: Click the **...** button to browse and select the script you created.

Artifact Name

```
drop
```

Artifact Type: **Server**.

> Artifacts are the files that you want your build to produce. Artifacts can be nearly anything your team needs to test or deploy your app. For example, you've got a .DLL and .EXE executable files and .PDB symbols file of a C# or C++ .NET Windows app.
>
> To enable you to produce artifacts, we provide tools such as copying with pattern matching, and a staging directory in which you can gather your artifacts before publishing them. See Artifacts in Team Build.

Enable continuous integration (CI)

1. Click the **Triggers** tab.
2. Enable **Continuous integration**.

> A continuous integration trigger on a build definition indicates that the system should automatically queue a new build whenever a code change is committed. You can make

> the trigger more general or more specific, and also schedule your build (for example, on a nightly basis). See Build triggers.

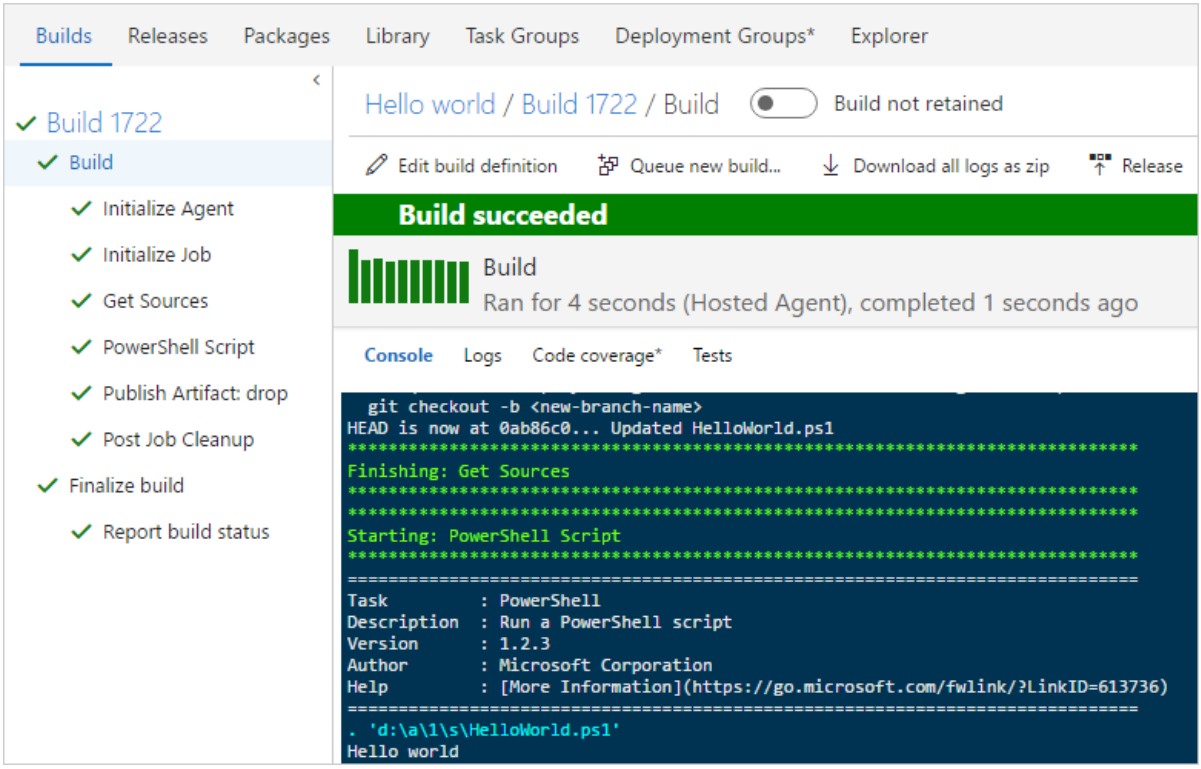
Save and queue the build

Save and queue a build manually and test your build definition.

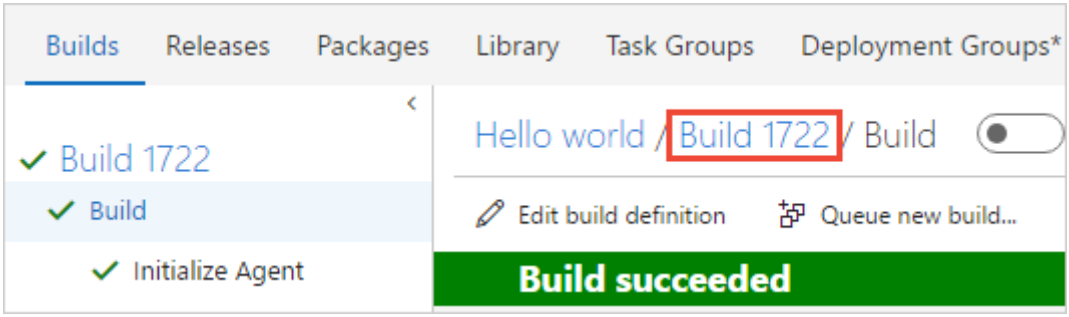
1. Click **Save & queue**, and then click **Save & queue**.

2. On the dialog box click the **Queue** button.

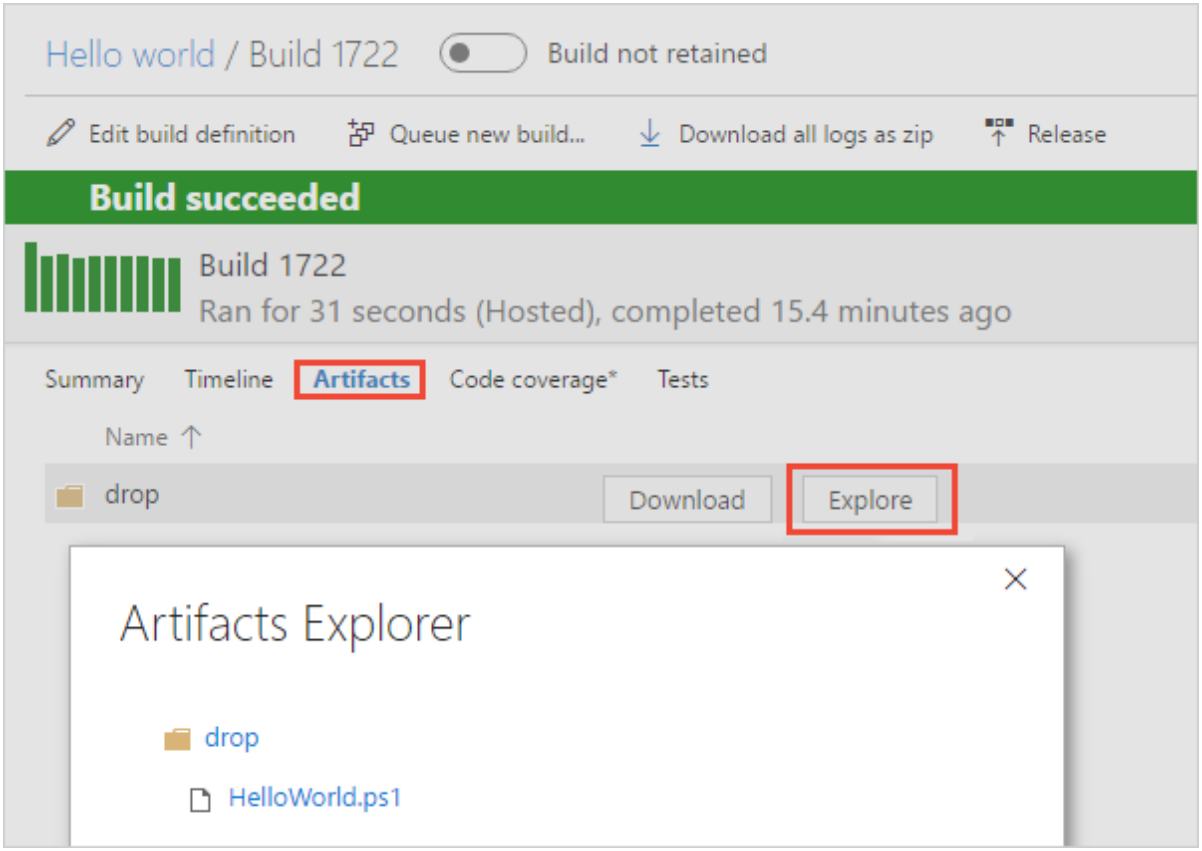
   This queues a new build on the hosted agent. Once the agent is allocated, you'll start seeing the live logs of the build. Notice that the PowerShell script is run as part of the build, and that "Hello world" is printed to the console.



3. Go to the build summary.



4. On the **Artifacts** tab of the build notice that the script is published as an artifact.

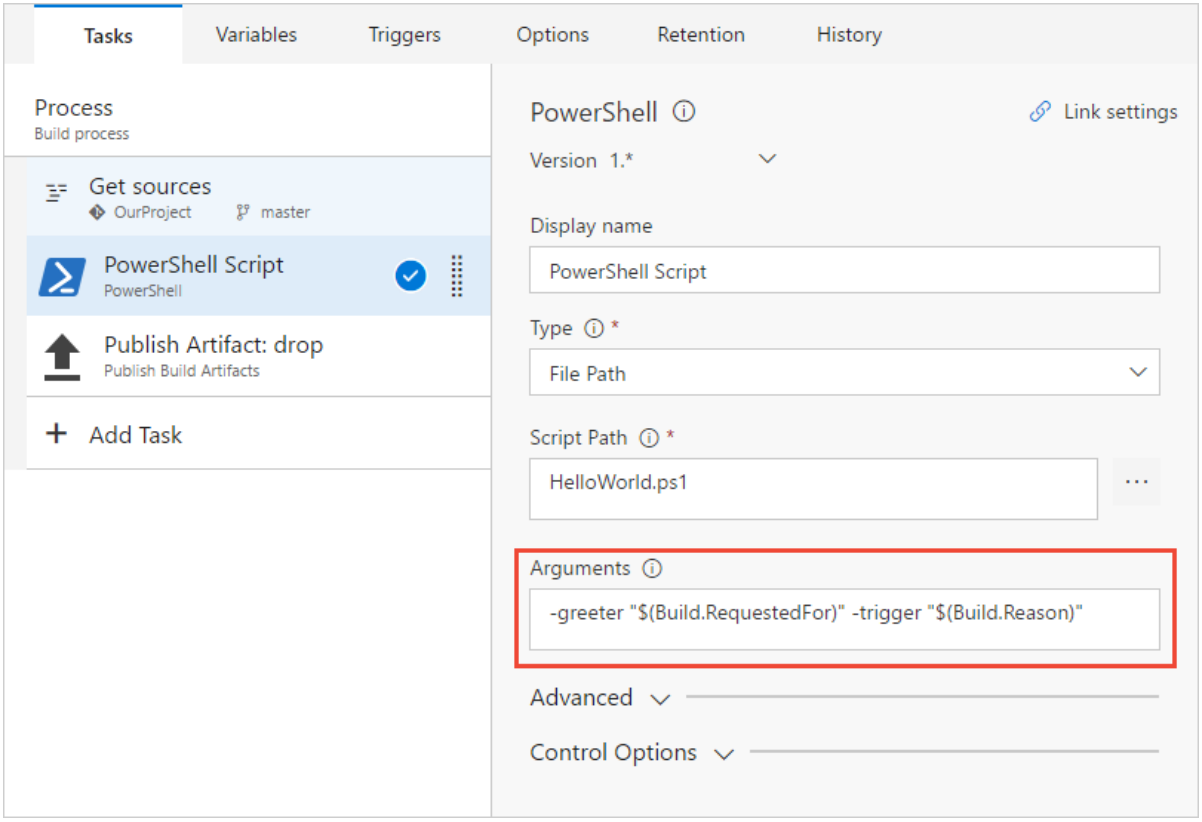


> You can view a summary of all the builds or drill into the logs for each build at any time by navigating to the **Builds** tab in the **Build & Release** hub. For each build, you can also view a list of commits that were built and the work items associated with each commit. You can also run tests in each build and analyze the test failures.

Add some variables and commit a change to your script

We'll pass some build variables to the script to make our process a bit more interesting. Then we'll commit a change to a script and watch the CI process run automatically to validate the change.

1. Edit your build definition.
2. On the **Tasks** tab, click the PowerShell script task.
3. Add these arguments.



Arguments

```
-greeter "$(Build.RequestedFor)" -trigger "$(Build.Reason)"
```

4. Save the build definition.

5. Go to the **Code** hub, **Files** tab.

6. Select the **HelloWorld.ps1** file, and then **Edit** the file.

7. Change the script as follows:

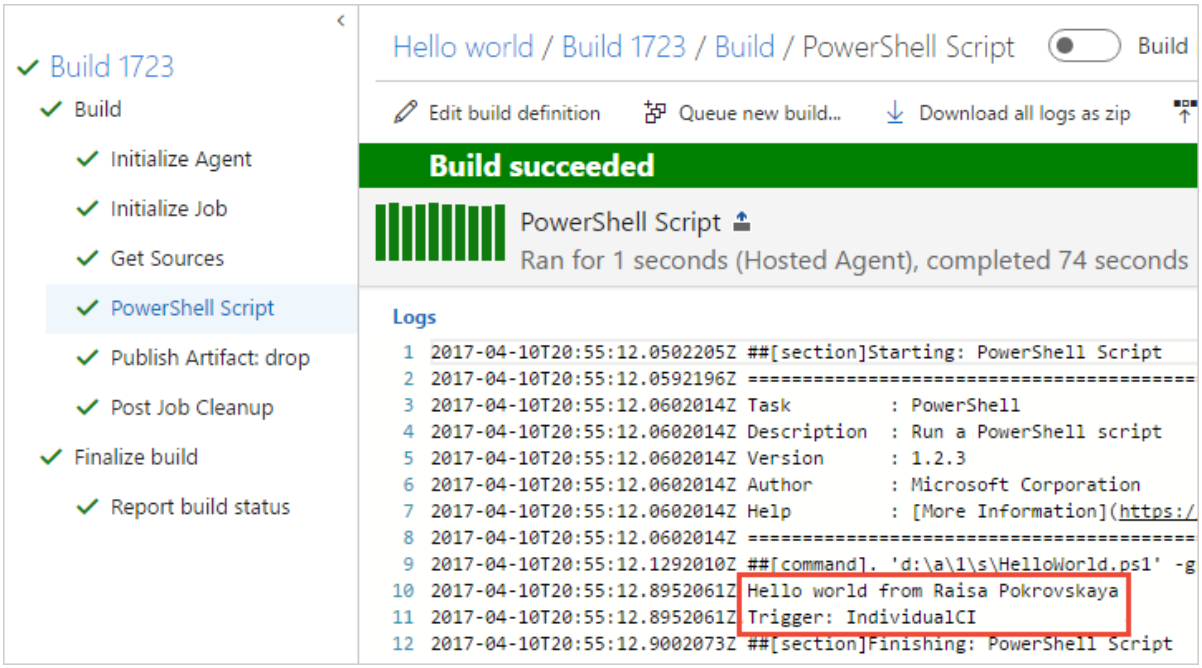
```ps
Param(
[string]$greeter,
[string]$trigger
)
Write-Host "Hello world" from $greeter
Write-Host Trigger: $trigger
```

8. **Commit** (save) the script.

9. Go to the **Build & Release** hub, and notice that a build is automatically triggered by the change that you committed.

10. Select the new build that was created and view its log.

11. Notice that the person who changed the code has their name printed in the greeting message. You also see printed that this was a CI build.



> We just introduced the concept of build variables in these steps. We printed the value of a variable that is automatically predefined and initialized by the system. You can also define custom variables and use them either in arguments to your tasks, or as environment variables within your scripts. To learn more about variables, see Build variables.

Create a release definition

Define the process for running the script in two environments.

1. Go to the **Build & Release** hub, and then to the **Releases** tab.

2. Select the action to create a **New definition**.

3. On the dialog box, select the **Empty** template and click **Next**.

4. Make sure that your **Hello world** build definition that you created above is selected. Select **Continuous deployment**, and then click **Create**.

5. Click **Add tasks** in the environment.

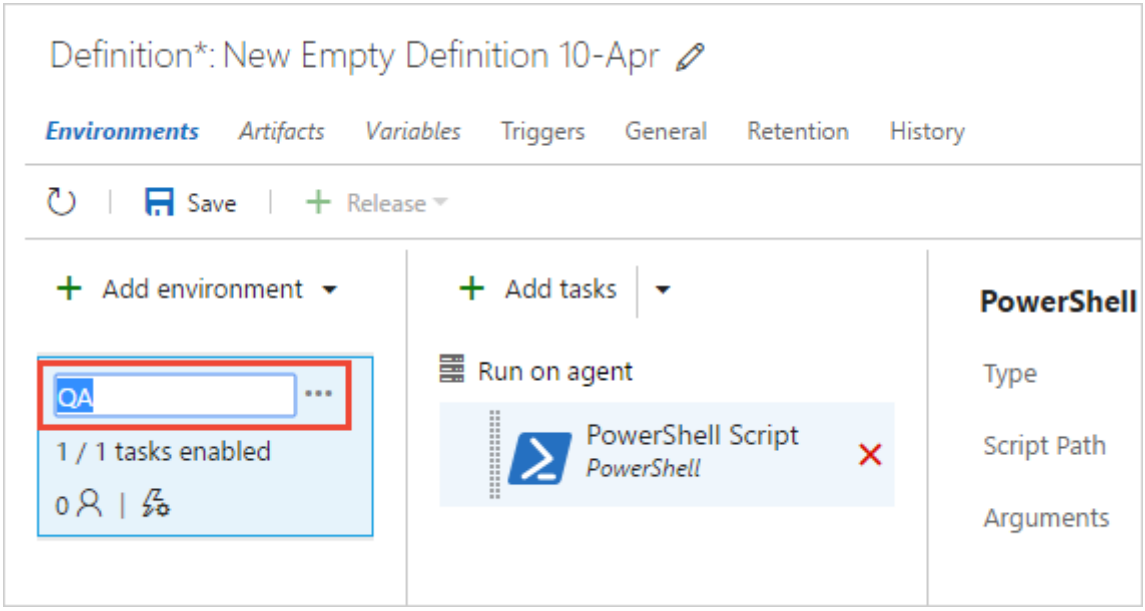
6. On the **Task catalog** dialog box, click **Utility**, locate the **PowerShell** task, and then click its **Add** button. Click the **Close** button.

7. For the **Script Path** argument, click the ... button to browse your artifacts and select the script you created.

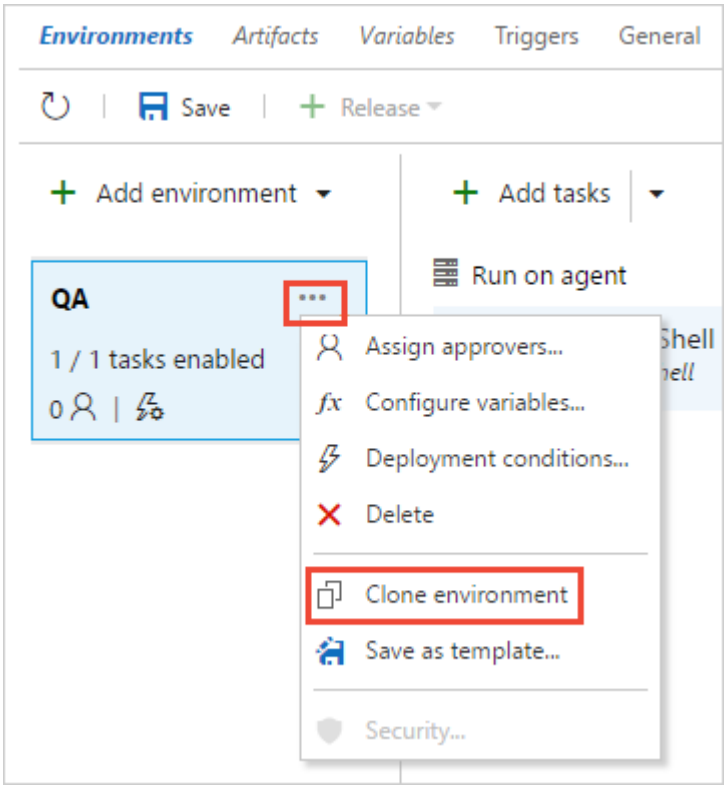
8. Add these **Arguments**:

```
-greeter "$(Release.RequestedFor)" -trigger "$(Build.DefinitionName)"
```

9. Rename the environment **QA**.



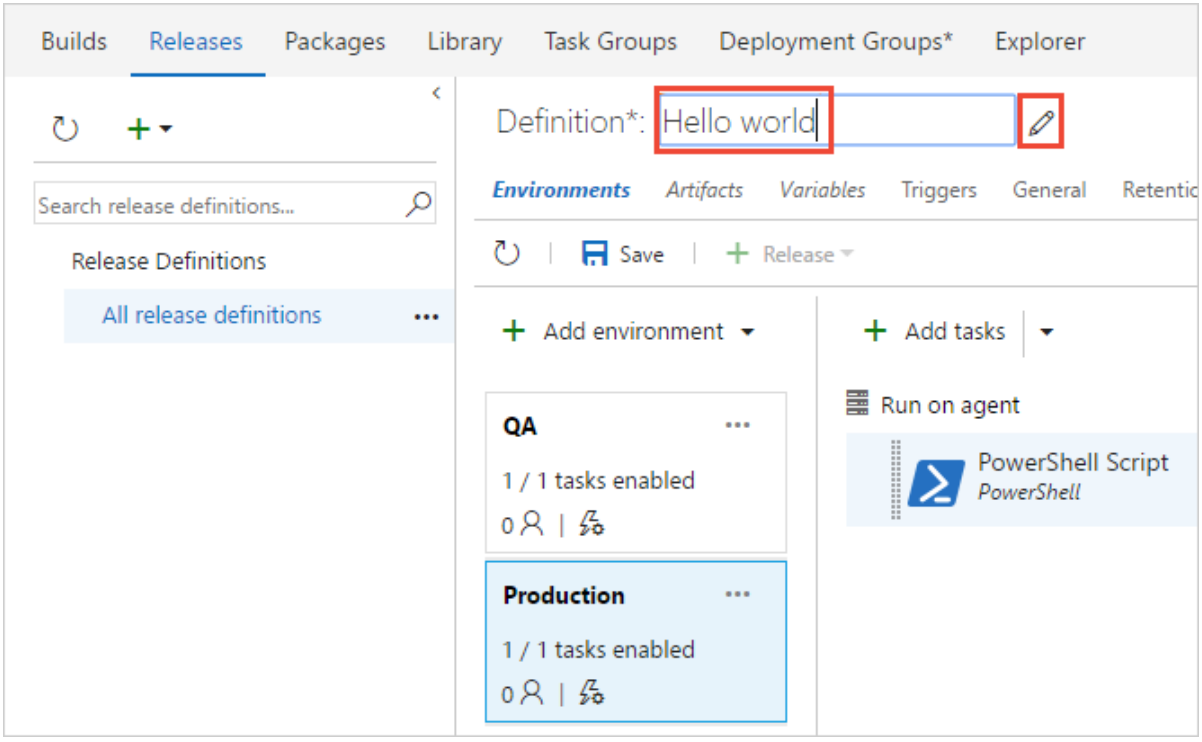
10. **Clone** the **QA** environment.



Leave **Automatically approve** and **Deploy automatically...** selected, and click **Create**.

11. Rename the new environment **Production**.

12. Rename the release definition **Hello world**.



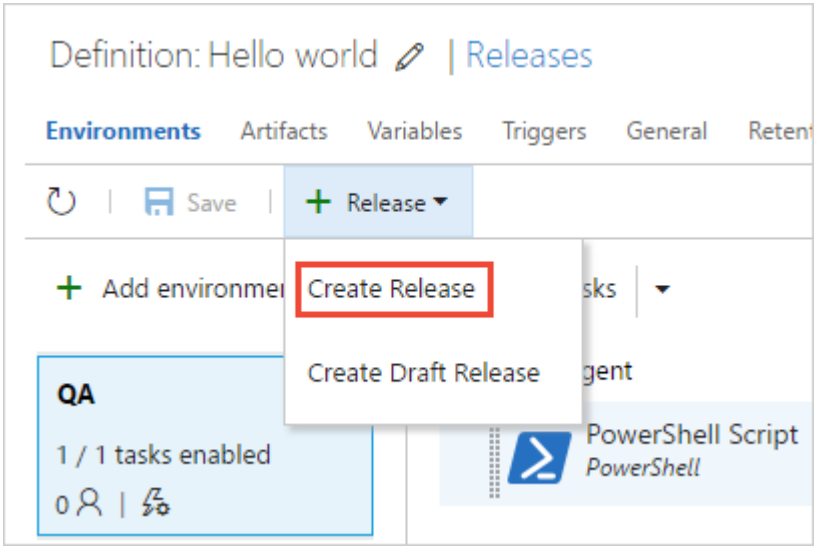
13. Save the release definition.

> A release definition is a collection of environments to which the application build artifacts are deployed. It also defines the actual deployment process for each environment, as well as how the artifacts are promoted from one environment to another.
>
> Also, notice that we used some variables in our script arguments. In this case, we used release variables instead of the build variables we used for the build definition.

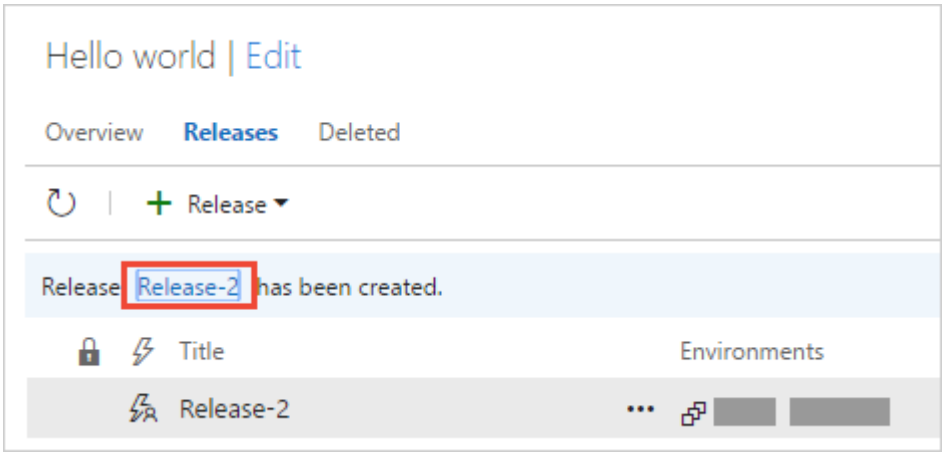
Deploy a release

Run the script in each environment.

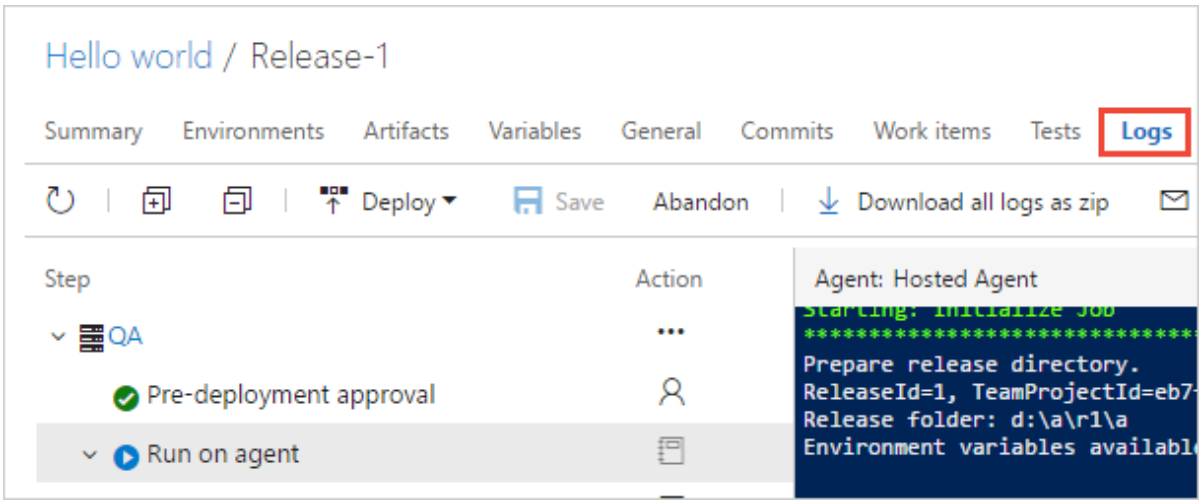
1. Create a new release.



2. Open the release that you just created.



3. View the logs to get real-time data about the release.



> You can track the progress of each release to see if it has been deployed to all the environments. You can track the commits that are part of each release, the associated work items, and the results of any test runs that you've added to the release process.

Change your code and watch it automatically deploy to production

We'll make one more change to the script. This time it will automatically build and then get deployed all the way to the production environment.

1. Go to the **Code** hub, **Files** tab, edit the **HelloWorld.ps1** file, and change it as follows:

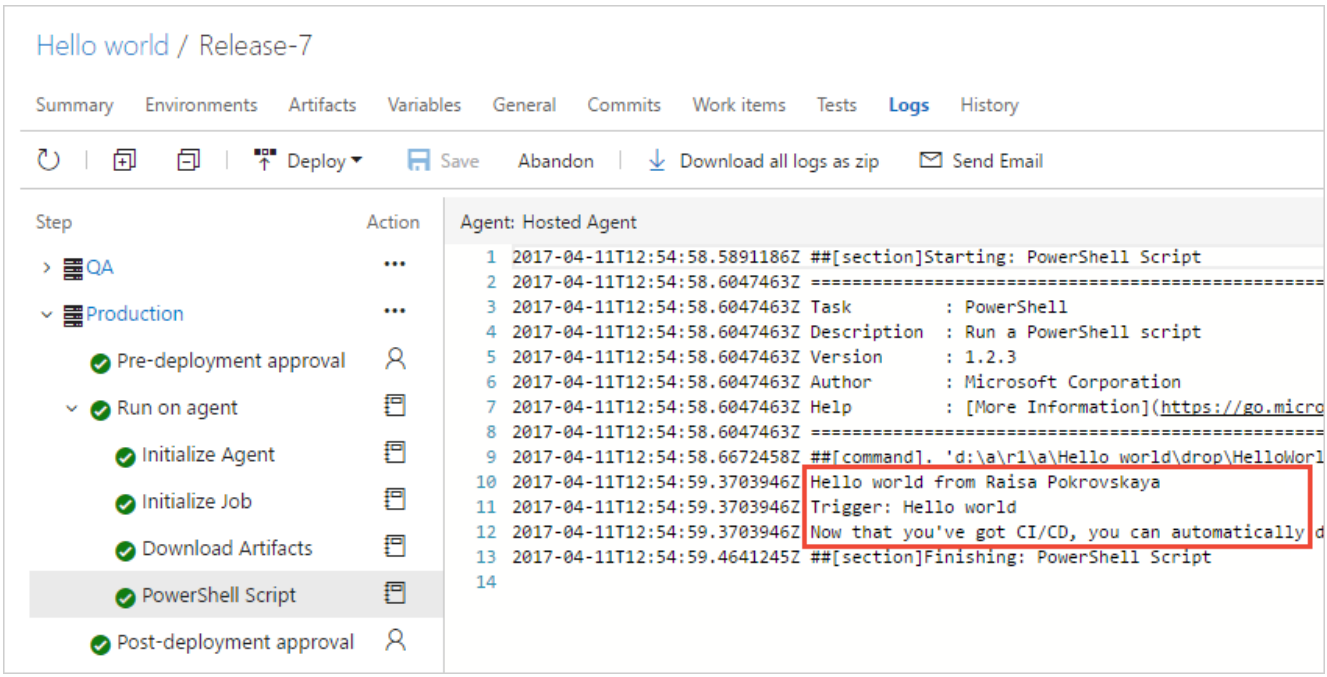
```ps
Param(
[string]$greeter,
[string]$trigger
)
Write-Host "Hello world" from $greeter
Write-Host Trigger: $trigger
Write-Host "Now that you've got CI/CD, you can automatically deploy your app ev
```

2. Save the script.

3. Click the **Builds** tab to see the build queued and run.

4. After the build is completed, click the **Releases** tab, open the new release, and then go to the **Logs**.

   Your new code automatically is deployed in the **QA** environment, and then in the **Production** environment.



> In many cases, you probably would want to edit the release process so that the production deployment happens only after some testing and approvals are in place. See Approvals and gates overview.

Put CI/CD to work for you

We hope this tutorial gave you an understanding of the basic concepts of Team Build and Release Management. To get started building and deploying an app, we suggest you proceed next to one of these topics:

- ASP.NET
- ASP.NET core
- Node.js
- Build and deploy your app

Q&A

Where can I read articles about DevOps and CI/CD?

What is Continuous Integration?

What is Continuous Delivery?

What is DevOps?

What kinds of version control can I use

We've used a Git repository in VSTS to keep things focused on CI/CD for this tutorial.

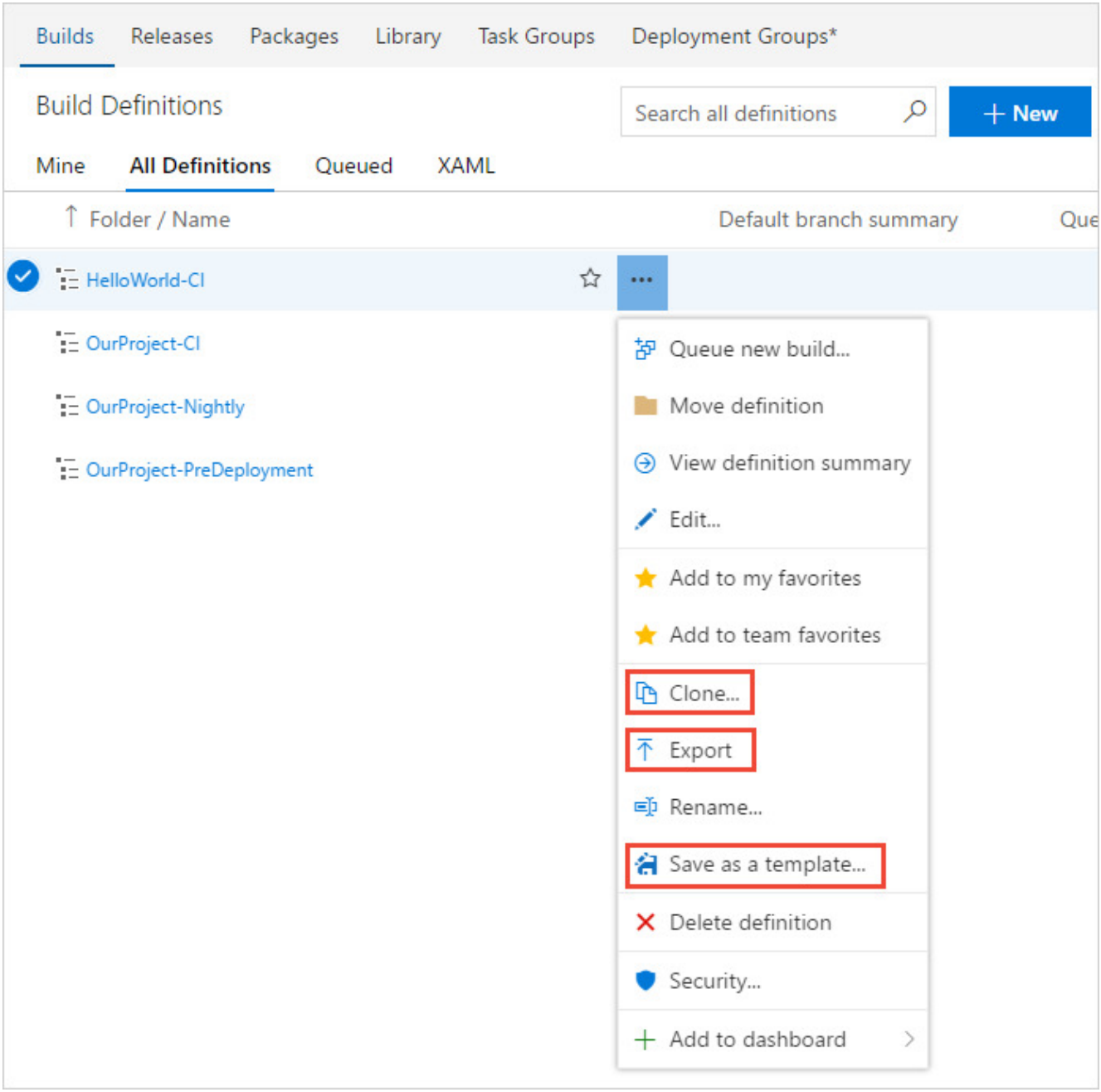
When you're ready to get going with CI/CD for your app, you can use the version control system of your choice:

- Clients
  - Visual Studio Code for Windows, macOS, and Linux
  - Visual Studio with Git for Windows or Visual Studio for Mac
  - Visual Studio with TFVC
  - Eclipse

    - Xcode
    - IntelliJ
    - Command line
- Services
    - VSTS
    - Git service providers such as GitHub and Bitbucket
    - Subversion

How do I replicate a definition?

If your definition has a pattern that you want to replicate in other definitions, clone it, export it, or save it as a template.



After you clone a definition, you can make changes and then save it.

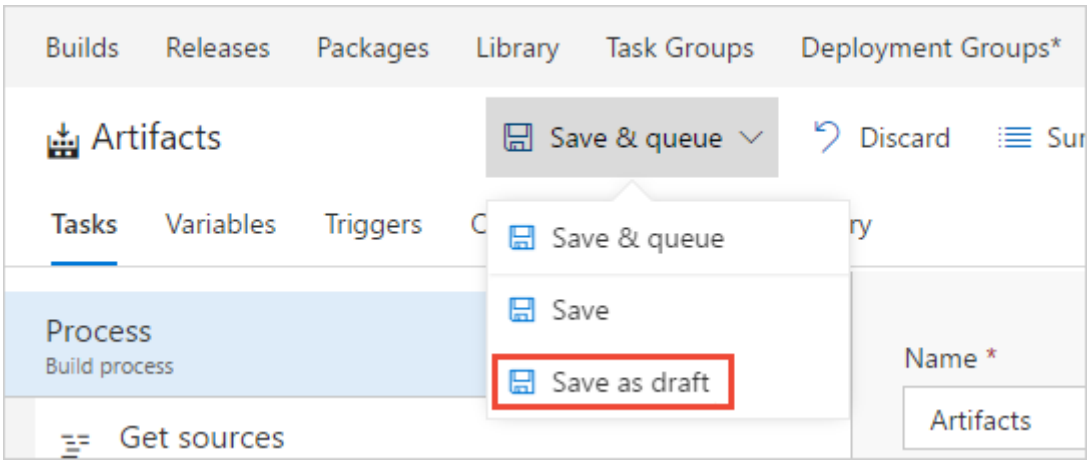
After you export a definition, you can import it from the **All Definitions** tab.

After you create a template, your team members can use it to follow the pattern in new definitions.

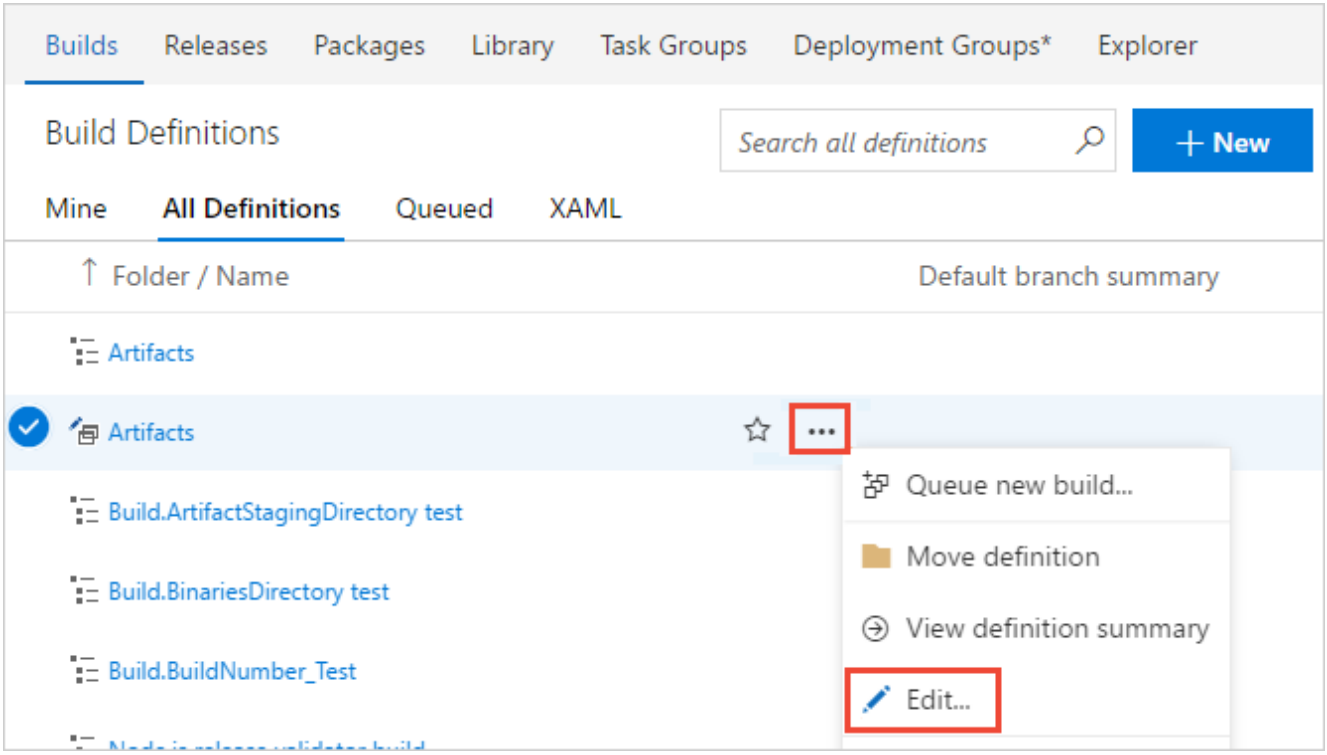
> Tip
>
> If you're using the **New Build Editor**, then your custom templates are shown at the bottom of the list.

How do I work with drafts?

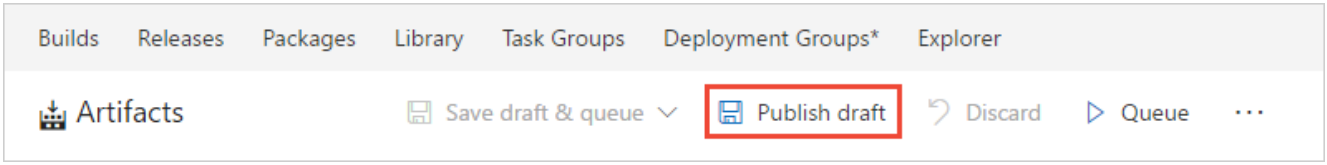
If you're editing a build definition and you want to test some changes that are not yet ready for production, you can save it as a draft.



You can edit and test your draft as needed.



When you're ready you can publish the draft to merge the changes into your build definition.



Or, if you decide to discard the draft, you can delete it from the **All Definition** tab shown above.

What else can I do when I queue a build?

You can queue builds automatically or manually.

When you manually queue a build, you can, for a single run of the build:

- Specify the queue into which the build goes.
- Add and modify some variables.
- Add demands.
- In a Git repository
  - Build a branch or a tag.
  - Build a commit.
- In a TFVC repository
  - Specify the source version as a label or changeset.
  - Run a private build of a shelveset. (You can use this option on either a hosted agent or a private windows agent. You cannot use it with a cross-platform agent.)

Where can I learn more about build definition settings?

To learn more about build definition settings, see:

- Getting sources
- Tasks
- Variables
- Triggers
- Options
- Retention
- History

How do I programmatically create a build definition?

REST API Reference: Create a build definition