

CAS DevOps Track

IBM Integration Bus (IIB) Delivery Pipeline Documentation

Document Information

Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Revision Number** | **Revision Date** | **Revision**  **By** | **Summary of Changes** | **Changes marked** |
| 1.0 | 2/14/2018 | Greg H | Draft |  |
| 1.1 | 2/21/2018 | Greg H | Added deploy documentation. |  |
| 1.2 | 3/2/2018 | Greg H | Refactored deploy documentation and added guidance for using .broker files. Also filled out troubleshooting sections. |  |
| 1.3 | 3/7/2018 | Greg H | Added reference section for mapping variables to artifacts in Git. Added guidance for using bar restart. |  |
| 1.4 | 3/9/2018 | Greg H | Added documentation for enabling continuous integration as well as for scheduled deploys. |  |
| 1.5 | 3/14/2018 | Greg H | Made CI a part of default build steps. Defined pipeline of environments, example workflow, and usage reference. |  |
| 1.6 | 3/16/2018 | Greg H | Added additional version control guidance for merging and updated guidance on cloning. Added guidance for approvals and deferred deployments. |  |
| 1.7 | 5/21/2018 | Greg H | Moved to Humana template. |  |
| 1.8 | 6/4/2018 | Greg H | Updated build and deploy creation steps with new TFS screens. |  |
| 1.9 | 6/6/2018 | Greg H | Updated with diagrams from design artifacts produced for Encounters Management. |  |
| 1.10 | 7/27/2018 | Greg H | Added support for variable groups. |  |
| 1.11 | 8/6/2018 | Greg H | Added setting of override file variable. Added steps for creating a build definition template and a release definition template. |  |

Approvals

This document requires following approvals.

|  |  |  |
| --- | --- | --- |
| Name | Title | Comments |
|  |  |  |

Table of Contents

[I. Introduction 6](#_Toc521306814)

[I.1. Purpose 6](#_Toc521306815)

[I.2. Audience 6](#_Toc521306816)

[I.3. Abbreviations 6](#_Toc521306817)

[II. Toolchain and Pipeline 7](#_Toc521306818)

[II.1. Delivery Pipeline Context 7](#_Toc521306819)

[II.2. Toolchain Flow 7](#_Toc521306820)

[II.3. Deployment Pipeline Environments 8](#_Toc521306821)

[II.4. Automation Steps 9](#_Toc521306822)

[II.5. Example Toolchain and Pipeline Workflow 9](#_Toc521306823)

[II.6. Configuring the Toolchain to Support a Pipeline 11](#_Toc521306824)

[III. Usage 12](#_Toc521306825)

[III.1. Version Control 12](#_Toc521306826)

[III.1.1. Create a new Git repository 12](#_Toc521306827)

[III.1.2. Cloning your Git repository from IIT 13](#_Toc521306828)

[III.1.3. Share source code in a Git repository 18](#_Toc521306829)

[III.1.4. Synchronizing changes with a remote repository 21](#_Toc521306830)

[III.2. Build 27](#_Toc521306831)

[III.2.1. Setting up a new build 27](#_Toc521306832)

[III.2.2. Triggering a build 31](#_Toc521306833)

[III.2.3. Disabling continuous integration on your build 33](#_Toc521306834)

[III.2.4. Troubleshooting a failed build 34](#_Toc521306835)

[III.2.5. Creating a build definition template 36](#_Toc521306836)

[III.3. Deploy 37](#_Toc521306837)

[III.3.1. Setting up a new release definition 37](#_Toc521306838)

[III.3.2. Adding a new environment to a release definition 40](#_Toc521306839)

[III.3.3. Creating a release 44](#_Toc521306840)

[III.3.4. Manually trigger the deploy of a release to an environment 45](#_Toc521306841)

[III.3.5. Configuring a timetable-based scheduled deploy 47](#_Toc521306842)

[III.3.6. Configuring deploy approval and deferring deployment 48](#_Toc521306843)

[III.3.7. Troubleshooting a failed deploy 52](#_Toc521306844)

[III.3.8. Creating a release definition template 56](#_Toc521306845)

[IV. Upgrade Considerations 57](#_Toc521306846)

[IV.1. Upgrade of TFS 57](#_Toc521306847)

[IV.2. Upgrade of Docker 57](#_Toc521306848)

[IV.3. Upgrade of PBC 57](#_Toc521306849)

[IV.4. Upgrade of IIB and IIT 57](#_Toc521306850)

[V. Reference 58](#_Toc521306851)

[V.1. Mapping Variables to Artifacts 58](#_Toc521306852)

[V.1.1. BAR application name 58](#_Toc521306853)

[V.1.2. Override file 59](#_Toc521306854)

[V.1.3. .broker file 60](#_Toc521306855)

[V.2. Configuring Your Notifications 61](#_Toc521306856)

[V.3. Links 62](#_Toc521306857)

# Introduction

A DevOps delivery pipeline for managing the delivery of IBM Integration Bus application changes has been established which delivers changes through the test environments and on into production.

## Purpose

The purpose of this document is to provide a reference for understanding and using the DevOps delivery pipeline for IBM Integration Bus application changes.

## Audience

This document is for anyone with an interest in delivery of IBM Integration Bus application changes through the DevOps delivery pipeline.

## Abbreviations

|  |  |
| --- | --- |
| ANT | Another Nice Tool |
| BAR | Broker Archive |
| IIB | IBM Integration Bus |
| IIT | IBM Integration Toolkit |
| PBC | Prolifics Build Conductor |
| TFS | Microsoft Team Foundation Server |

# Toolchain and Pipeline

A set of tools forms a toolchain to support the delivery of IIB application changes through the delivery pipeline.

## Delivery Pipeline Context

The context diagram shows the input and output artifacts of the delivery pipeline.

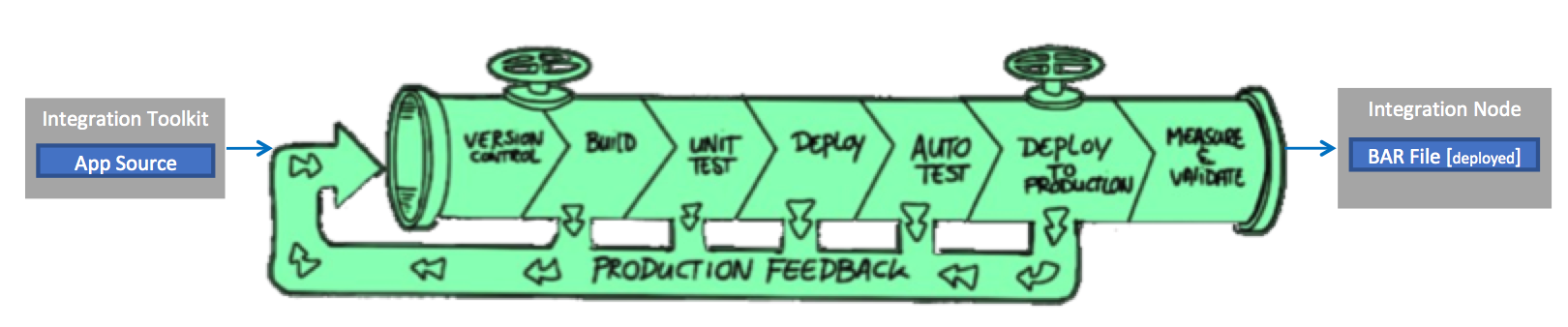


Figure 1 – Delivery pipeline context

## Toolchain Flow

The flow diagram shows how the toolchain supports the delivery pipeline.

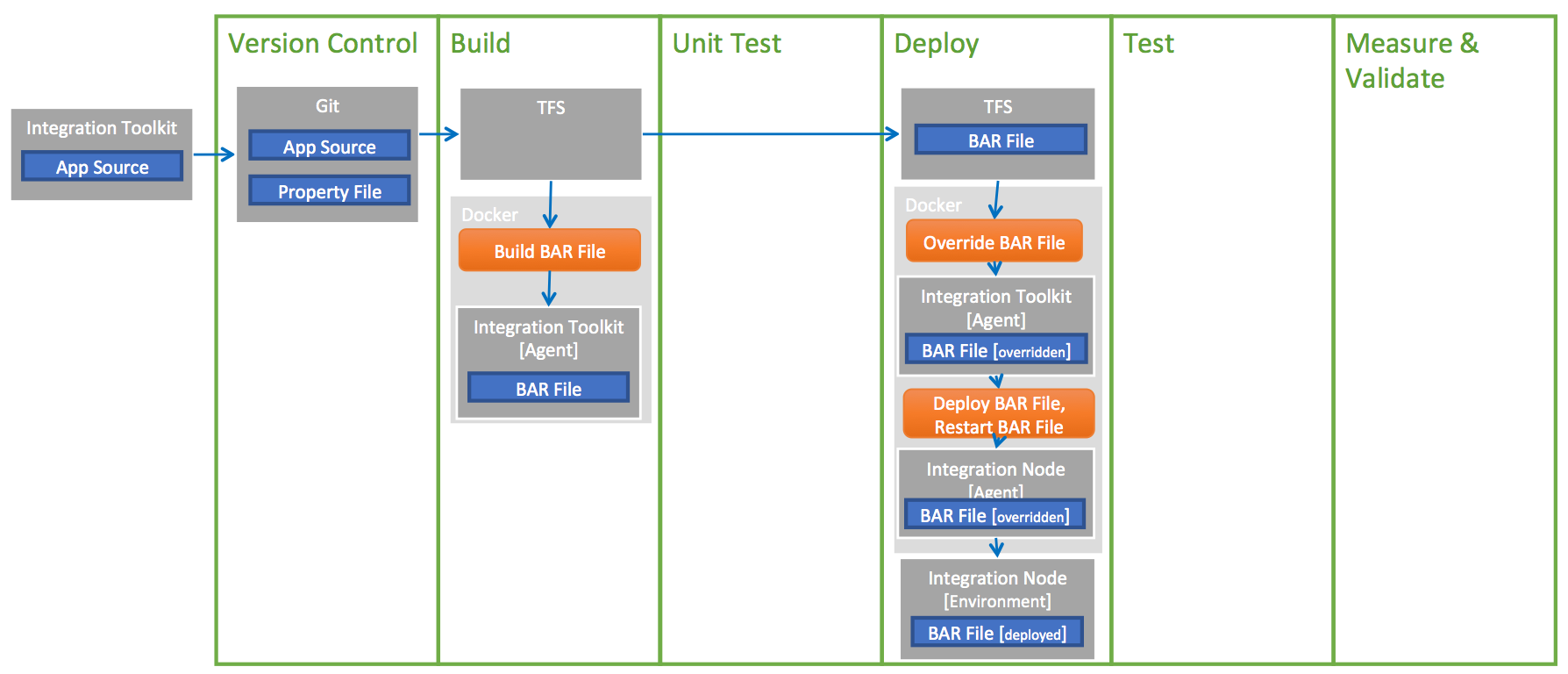


Figure 2 – Toolchain flow

The table below summarizes the tools in the toolchain.

|  |  |  |
| --- | --- | --- |
| Capability | Tools | Usage |
| IDE | IBM Integration Toolkit (IIT) | IIT is used to write IIB source code |
| Version Control | Git, Egit | Git is used to version control the source code. EGit is an Eclipse plugin that allows interaction with a Git repository from IIT. |
| Build | TFS, Docker, PBC, Integration Toolkit | An automated build triggered in TFS results in the startup of a Docker container that contains PBC, which then executes an automated build of a BAR file. PBC uses Integration Toolkit to build the BAR. |
| Unit Test | - | - |
| Deploy | TFS, Docker, PBC, Integration Toolkit, Integration Node | An automated deploy triggered in TFS results in the startup of a Docker container that contains PBC, which then executes an automated deployment of a BAR file to an IIB node. PBC first uses Integration Toolkit to do a BAR override, and then uses Integration Server to deploy the BAR file. |
| Auto Test | - | - |
| Deploy to Production | TFS, Docker, PBC , Integration Toolkit, Integration Node | See “Deploy”. |
| Measure & Validate | - | - |
| Provision | - | - |
| Pipeline Analytics | - | - |

## Deployment Pipeline Environments

The environments that the deployment pipeline deploys to are shown.



Figure 3 – Deployment pipeline environments

## Automation Steps

The specific tools used by each of the automation steps are shown.

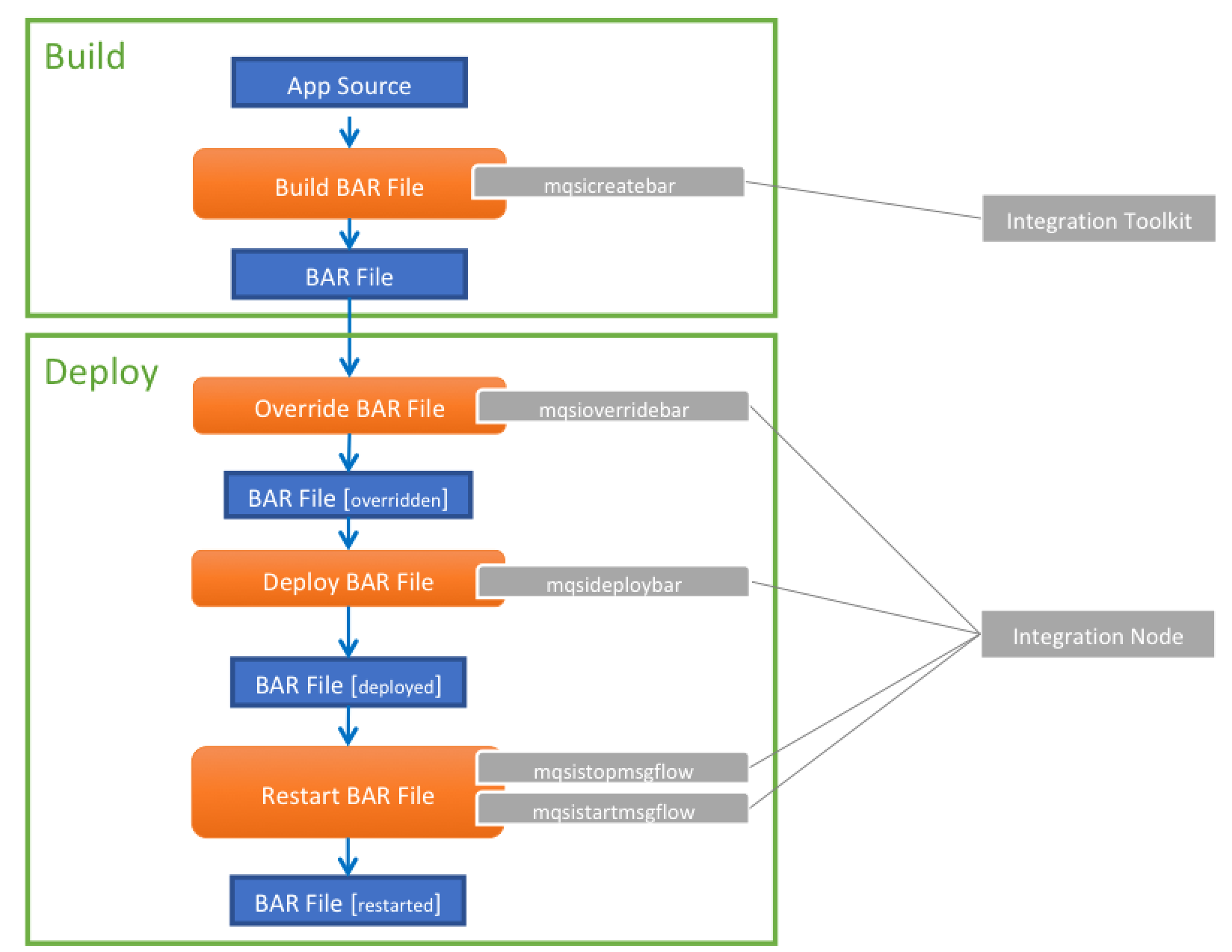


Figure 4 – Automation step design

## Example Toolchain and Pipeline Workflow

1. Developer commits code to Git.
2. Developer additionally commits property override files and .broker files (to provide connection details) for each environment to deploy to.
3. Continuous integration build kicks off, retrieves code from Git, and builds the BAR file.
4. On a successful build, a release is automatically created, and immediately deployed to the **Dev** environment:
   1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
   2. The BAR file first has its properties overridden using the override file for Dev.
   3. The BAR file is then deployed to Dev.
   4. After deployment, the application(s) in the BAR file are automatically restarted in Dev.
5. On a successful deploy to Dev, a deployment to **INT** is automatically triggered and carried out at the next time dictated by a defined SIT deploy schedule.
   1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
   2. The BAR file first has its properties overridden using the override file for SIT.
   3. The BAR file is then deployed to SIT.
   4. After deployment, the application(s) in the BAR file are automatically restarted in SIT.
6. Once a QA deploy is approved in the ETP system, the deployment to **QA** is manually triggered, dependent on an approval that provides the actual date/time for deployment.
   1. Approval occurs, with a deferred deployment date/time provided.
7. On reaching the deferred date/time, the deployment to QA occurs.
   1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
   2. The BAR file first has its properties overridden using the override file for QA.
   3. The BAR file is then deployed to QA.
   4. After deployment, the application(s) in the BAR file are automatically restarted in QA.
8. On a successful deploy to QA, a deployment to **Load** is automatically triggered
   1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
   2. The BAR file first has its properties overridden using the override file for Load.
   3. The BAR file is then deployed to Load.
   4. After deployment, the application(s) in the BAR file are automatically restarted in Load.
9. Once a production deploy is approved in the ETP system, the deployment to **Stage** is manually triggered.
   1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
   2. The BAR file first has its properties overridden using the override file for Stage.
   3. The BAR file is then deployed to Stage.
   4. After deployment, the application(s) in the BAR file are automatically restarted in Stage.
10. On a successful deploy to Stage, a deployment to **Prod** is automatically triggered, dependent on an approval that provides the actual date/time for deployment.
    1. Approval occurs, with a deferred deployment date/time provided.
11. On reaching the deferred date/time, the deployment to Prod occurs.
    1. The BAR file is retrieved from the build, and the override file is retrieved from Git.
    2. The BAR file first has its properties overridden using the override file for Prod.
    3. The BAR file is then deployed to Prod.
    4. After deployment, the application(s) in the BAR file are automatically restarted in Prod.

## Configuring the Toolchain to Support a Pipeline

The table below provides a handy reference for which sections in the Usage section to consult to setup the pipeline described in section II.4.

|  |  |
| --- | --- |
| **Setup Step** | **Usage Guidance Section** |
| Setup a Git repo for your source code, override property files and .broker files. | Section III.1. Version Control |
| Setting up a new build. | Section III.2.1 Setting up a new build |
| Setting up a new release definition | Section III.3.1 Setting up a new release definition (this will automatically include your dev environment) |
| Adding the SIT environment | Section III.3.2 Adding a new environment to a release definition  Section III.3.5 Configuring a timetable-based scheduled deploy |
| Adding the QA environment | Section III.3.2 Adding a new environment to a release definition |
| Adding the Load environment | Section III.3.2 Adding a new environment to a release definition |
| Adding the Stage environment | Section III.3.2 Adding a new environment to a release definition |
| Adding the Prod environment | Section III.3.2 Adding a new environment to a release definition  Section III.3.6 Configuring deploy approval and deferring deployment |

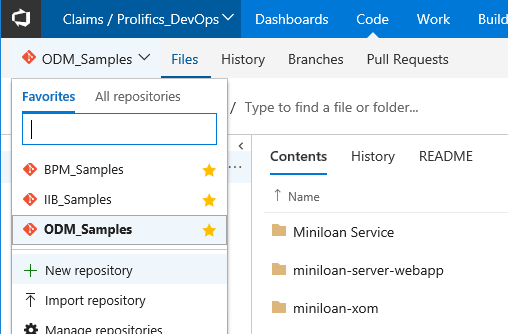
# Usage

## Version Control

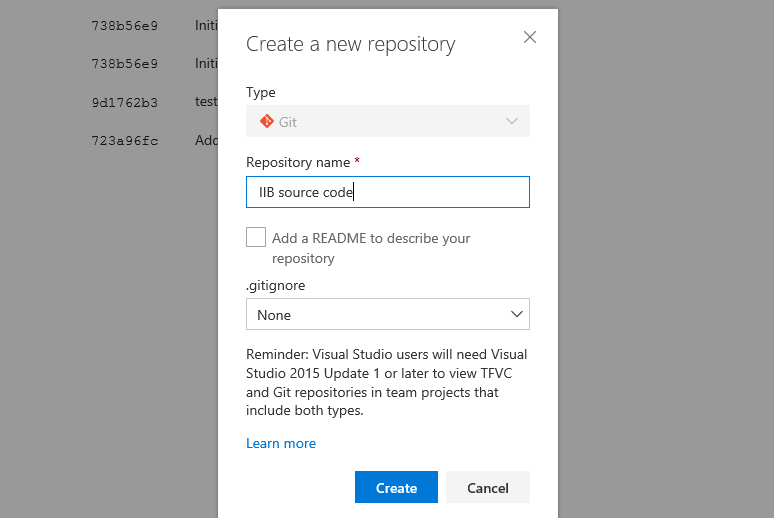
### Create a new Git repository

Create a Git repository from TFS to version control your IIB source code.

1. **Code** > click on the drop-down > **New repository**.



2. Provide a **Repository name** > optionally modify the settings > **Create**.

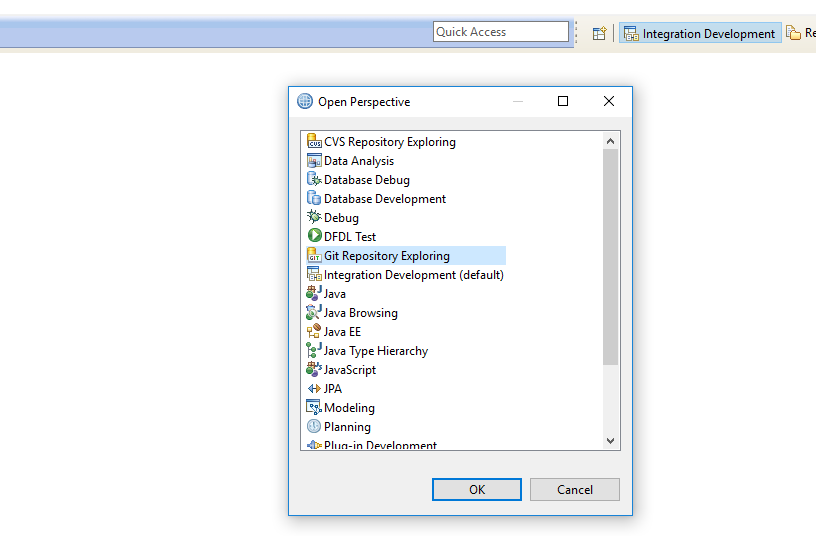


3. Your repository is ready to start using.

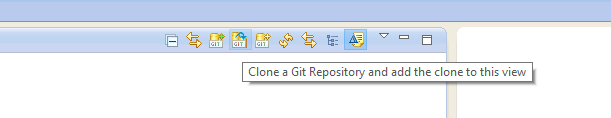
### Cloning your Git repository from IIT

Clone your repository from IIT using Egit.

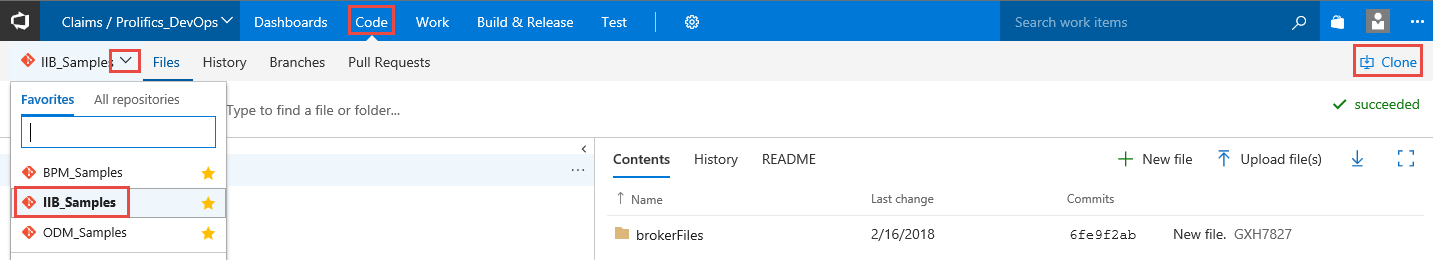
1. From IIT, click on the **Open Perspective** button > select **Git Repository Exploring** > **OK**.



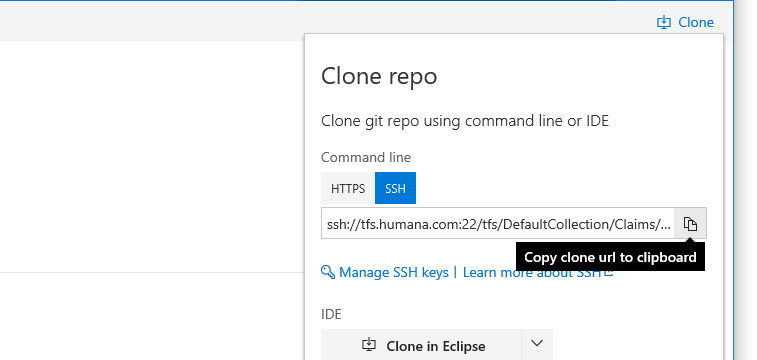
2. Click on **Clone a Git Repository and add the clone to this view**.



3. Switch to TFS > Code > with your repository open, click the **Clone** link.



4. **SSH** > **Copy clone url to clipboard**.

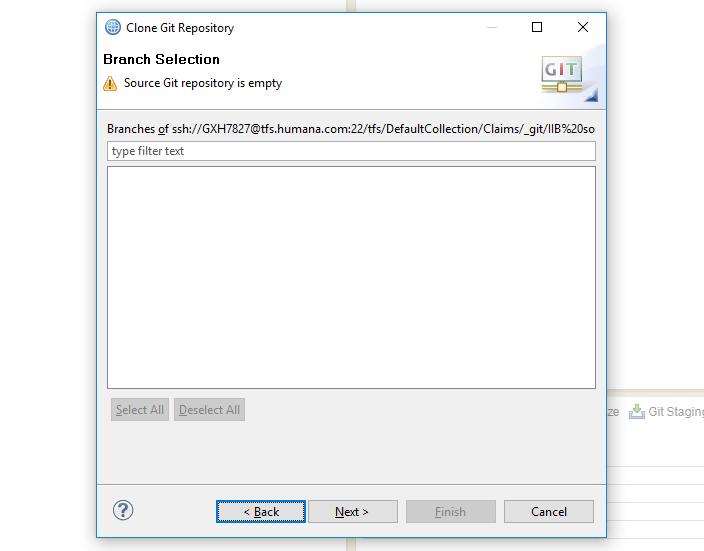


5. Switching back to IIT, paste the url into the **URI** field > enter your **User** and **Password** > **Next**.

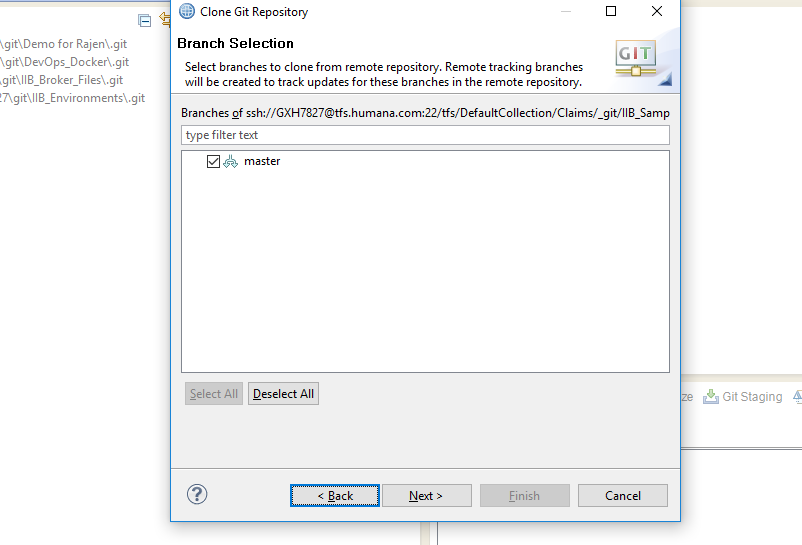


6. The next screen differs depending on whether there is any code in the repository or not.

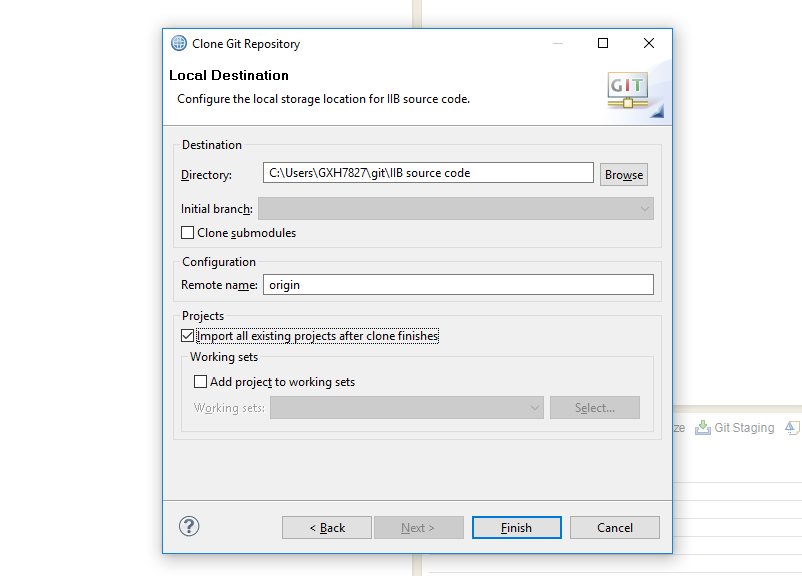
6a. If you have not yet shared any source code in the repository you will be prompted to tell you that the repository is empty > **Next**.



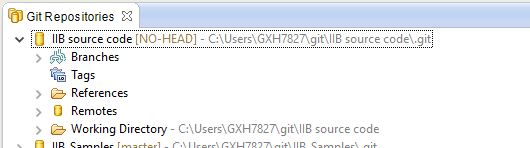
6b. If there is already source code in the repository then review the selected branch > **Next**.



7. Ensure **Import all existing projects after clone finishes** is selected > **Finish**.

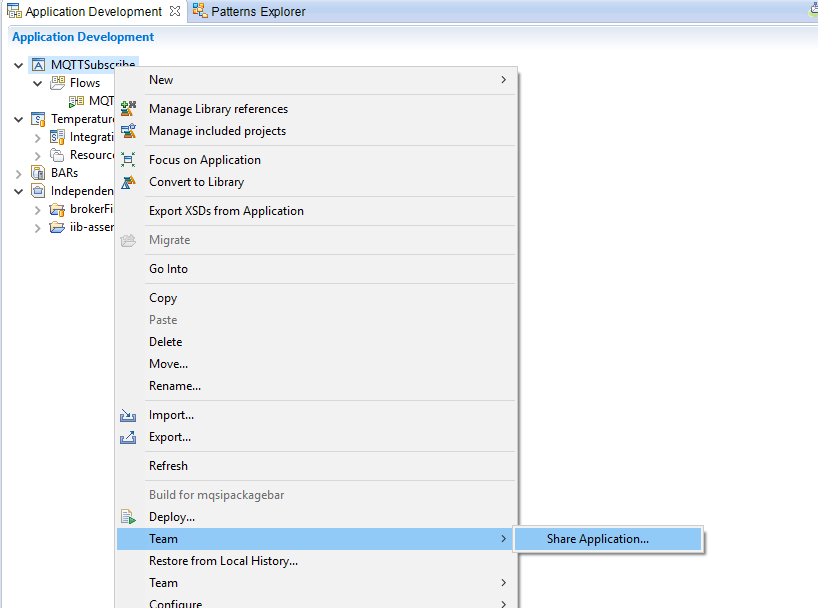


8. Your cloned repository will appear in the **Git Repositories** view.

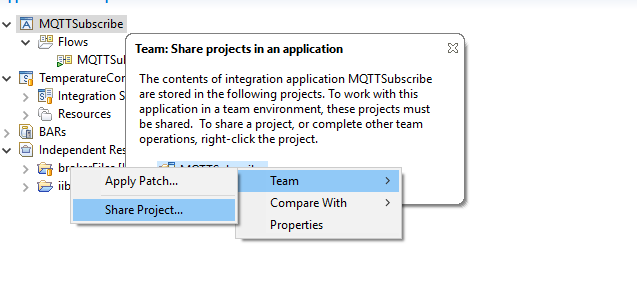


### Share source code in a Git repository

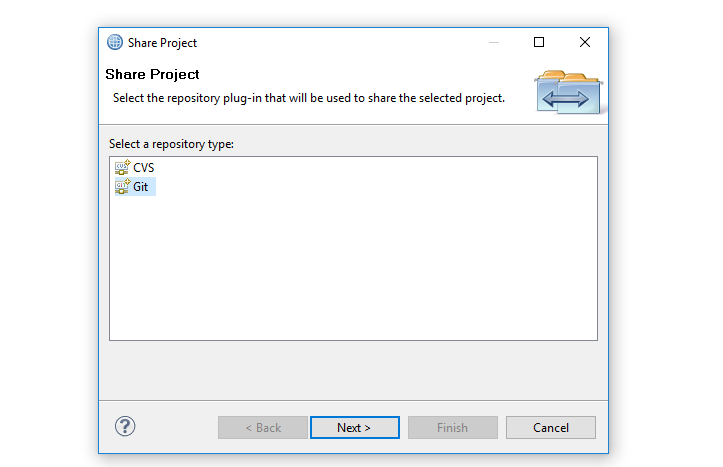
1. From IIT in the **Application Development** view > right-click on the application you wish to share > **Team** > **Share Application…**.



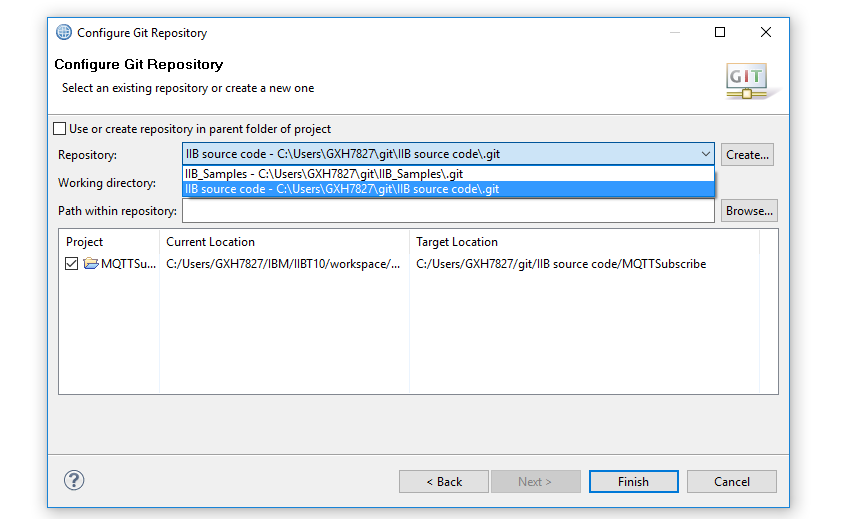
2. From the **Team: Share projects in an application** screen, right-click on the project you wish to share > **Team** > **Share Project…**.



3. Select **Git** > **Next**.



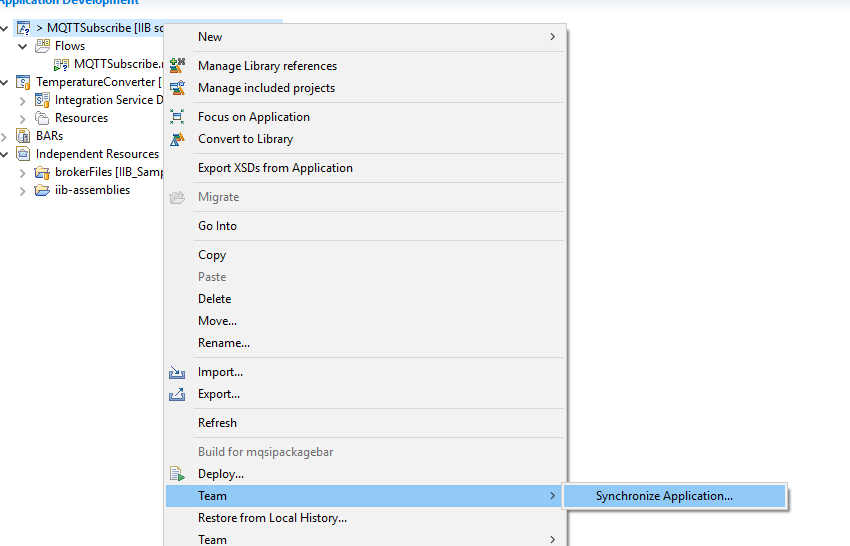
4. From the **Repository** drop-down, select the Git repo you wish to share your code in > **Finish**.



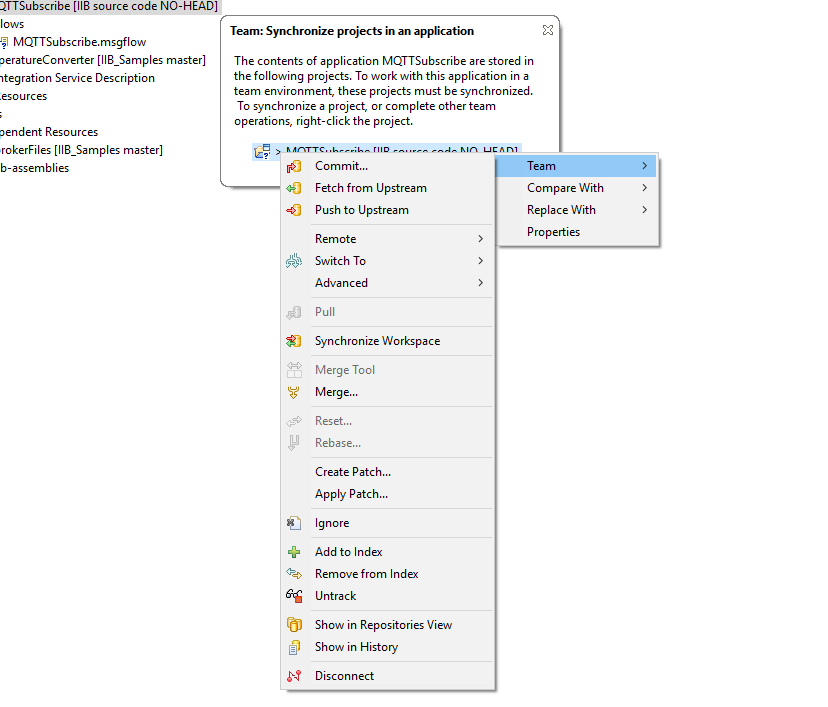
5. Your code is shared with your local cloned repository. The next section shows you how to deliver it to the remote repository.

### Synchronizing changes with a remote repository

1. From IIT in the **Application Development** view > right-click on the application you wish to synchronize > **Team** > **Synchronize Application…**.

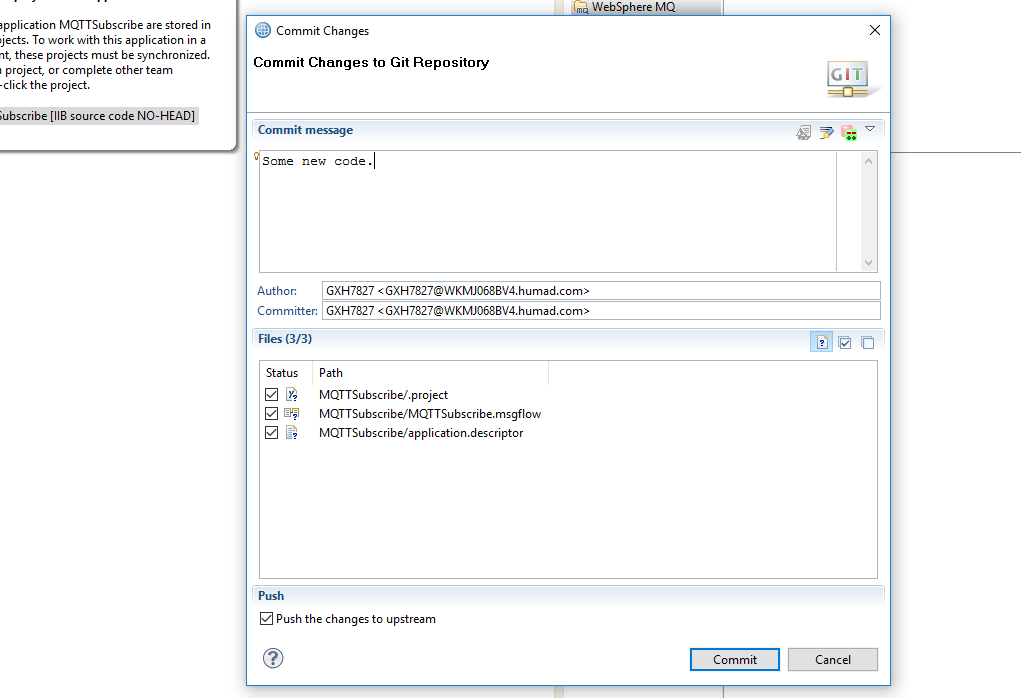


2. From the **Team: Synchronize projects in an application** screen, right-click on the project you wish to synchronize > **Team** > you will then be presented with a menu of actions to choose from.



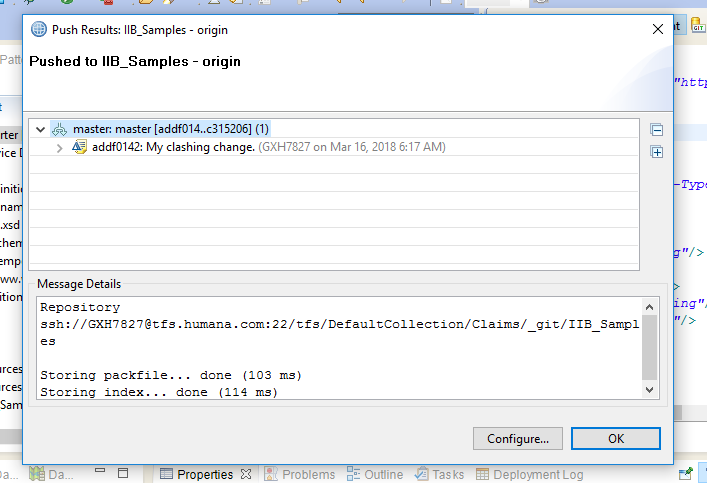
2a. If you have changes that have not yet been committed (e.g. you have just shared a project, or you have changed the files in your project):

2a1. **Commit** > enter a **Commit message** > select the files you wish to commit > make sure **Push the changes to upstream** is selected > **Commit** >

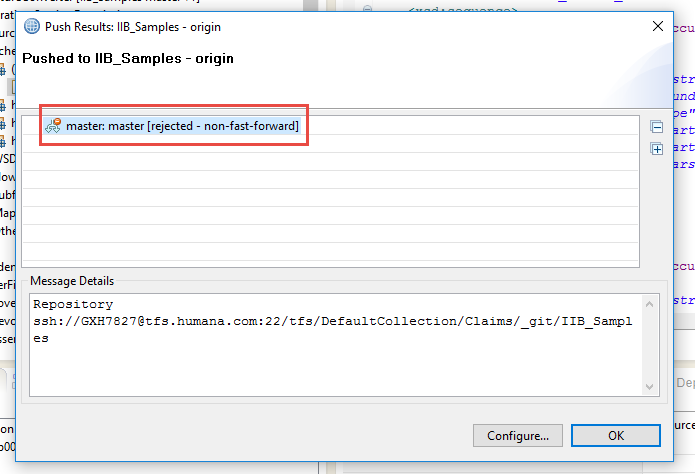


2a2.You will then either get a success or failure results screen.

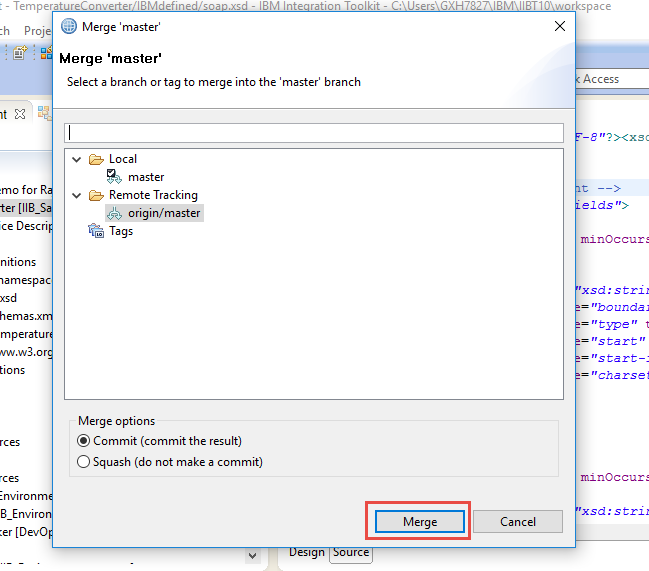
2a2a. On success you will see a results screen like the one below > click **OK**.



2a2b. A failure will appear as shown below: “rejected – non-fast-forward”, which means that you first need to pull down someone else’s changes and merge those with your own > **OK**.



2a2b1. Right-click on your application > **Team** > **Synchronize integration service…** > right-click on project > **Team** > **Merge…** > **Merge**.



2a2b2. Click OK on result merge confirmation screen.

Now we try the commit again. Resume at step 1 and follow the steps for committing a change.

2b. If you wish to accept changes that other developers have pushed to the remote repository:

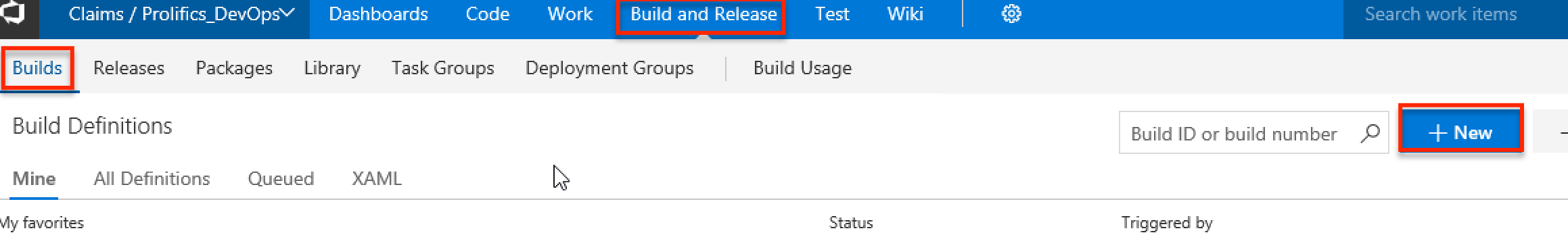
2b1. **Fetch from Upstream** > click **OK** on the results screen when it appears.

## Build

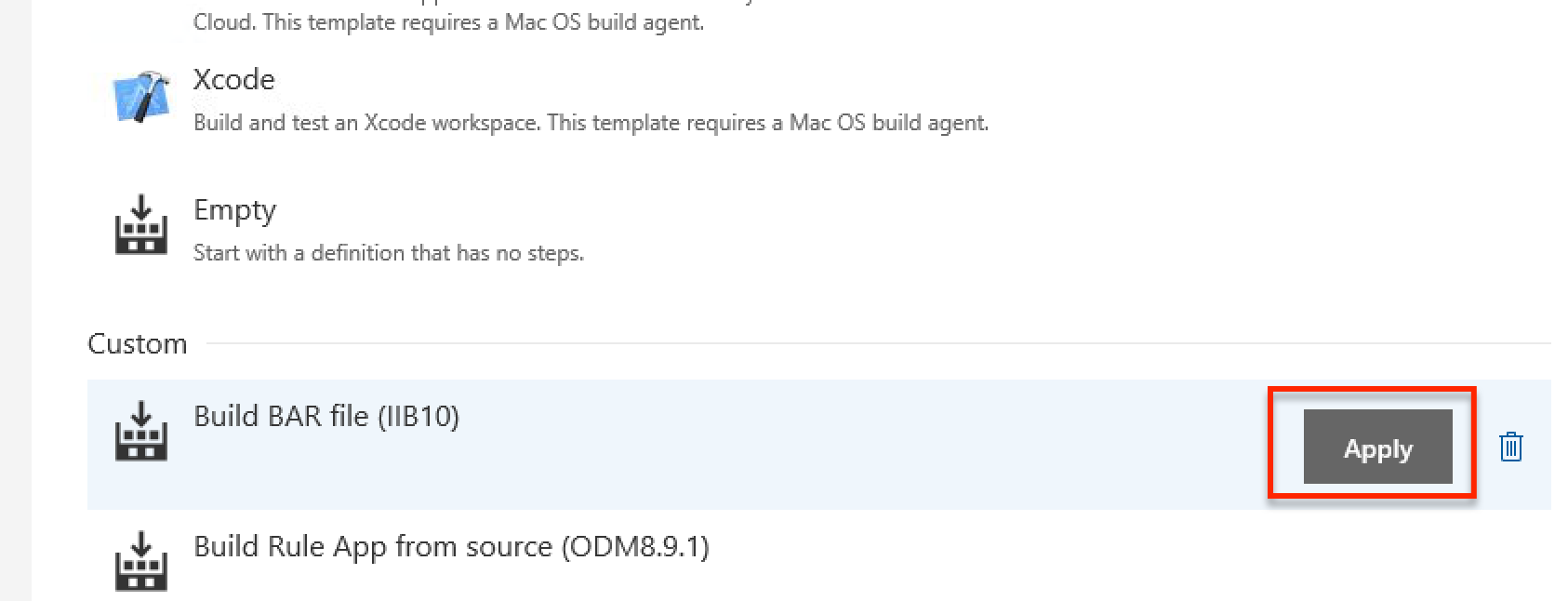
### Setting up a new build

Once you have delivered code changes that you wish to build and deploy, use the template to define a new build. You only have to do this once for an application.

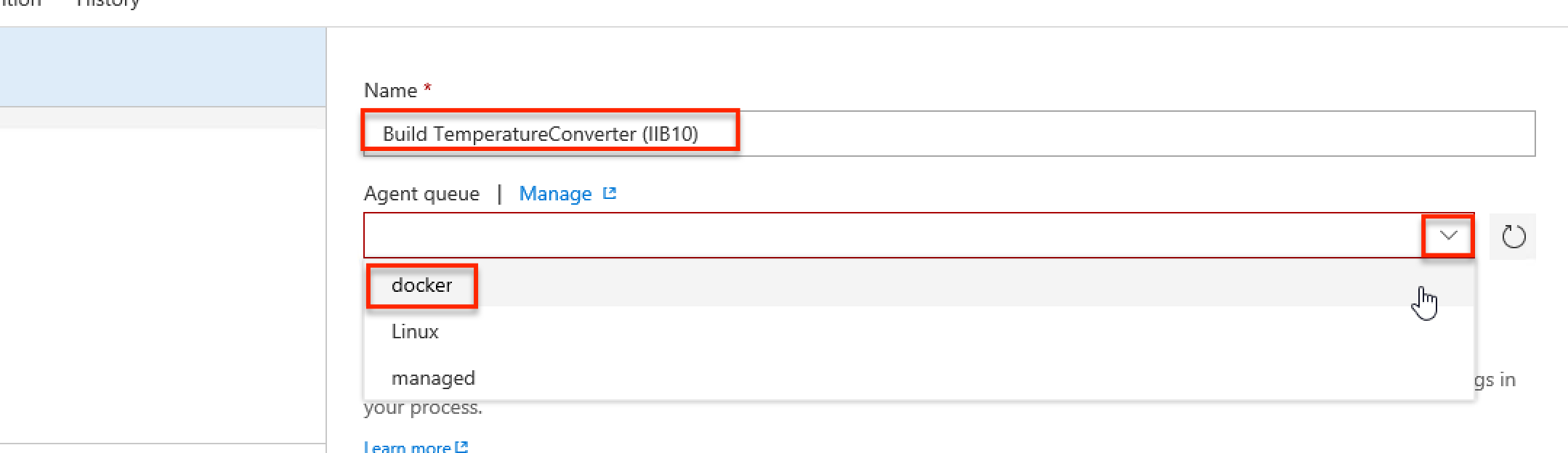
1. **Build and Release** > **Builds** > **New**.



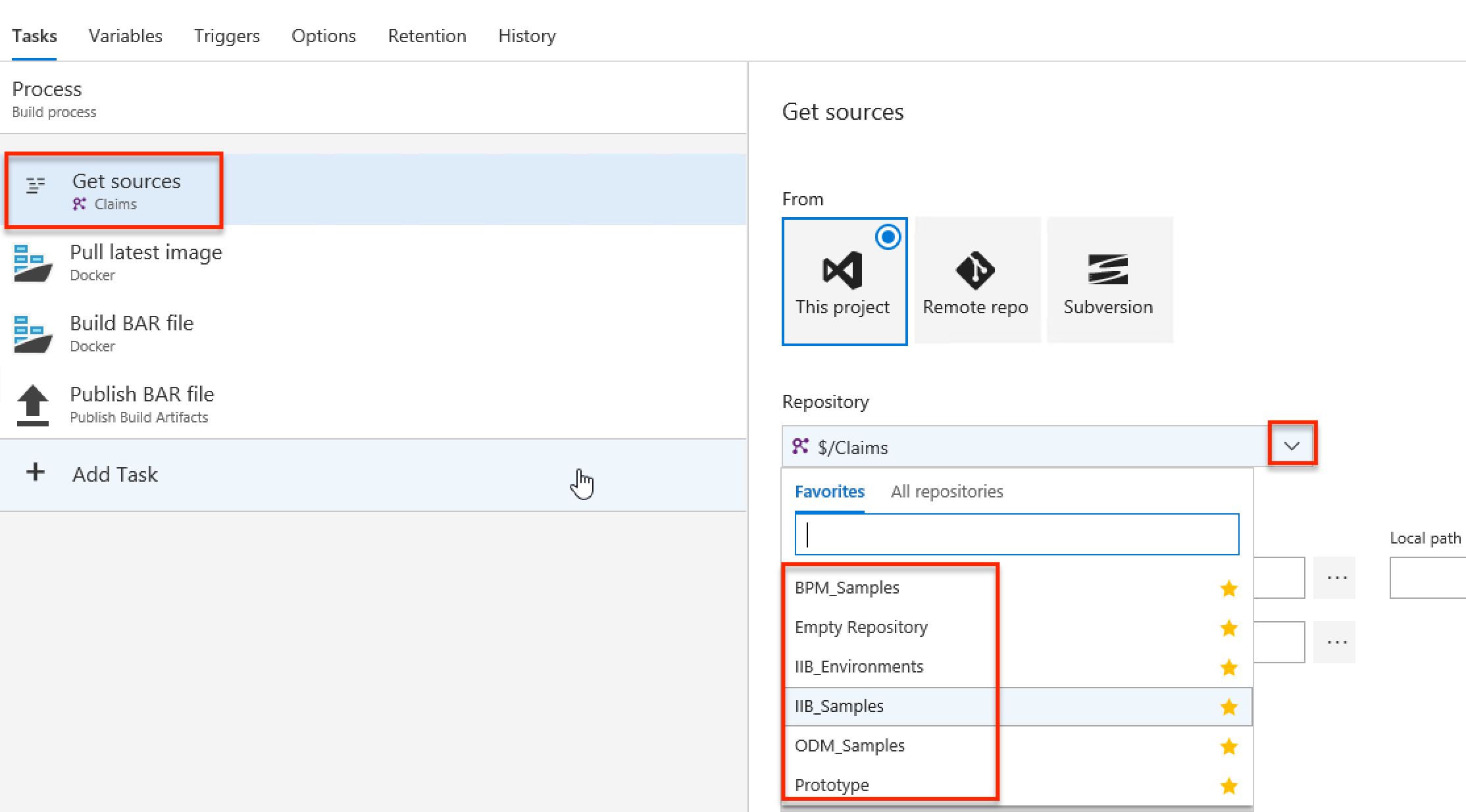
2. Under **Select a Template**, scroll down to the **Custom** section find **Build BAR file (IIB 10)** > **Apply**.



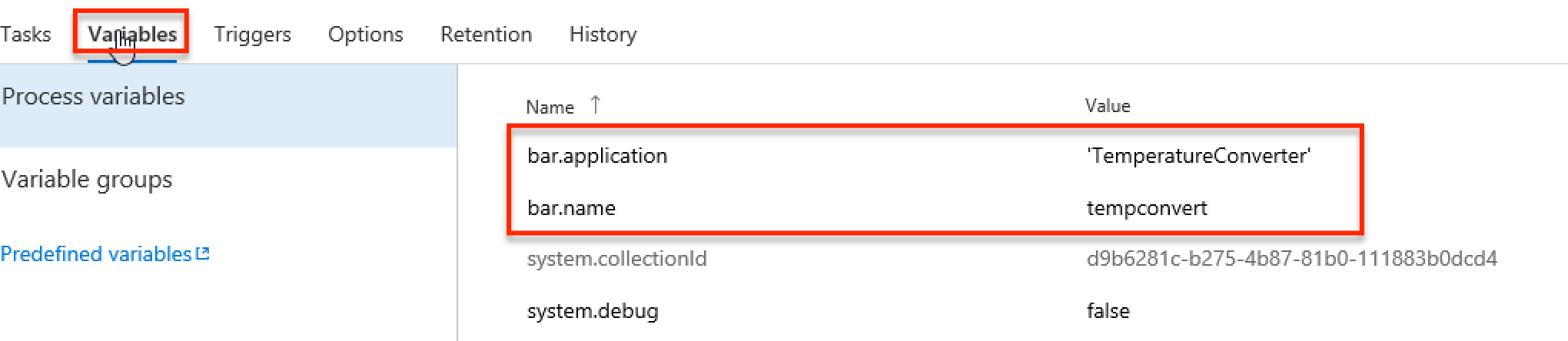
3. Provide a suitable **Name** for the build definition > click the drop-down for **Agent queue** and select **docker**.



4. On theTasks tab click the **Get sources** step in the **Process** > click the drop-down for **Repository** and select the correct repository to use as input for your build.

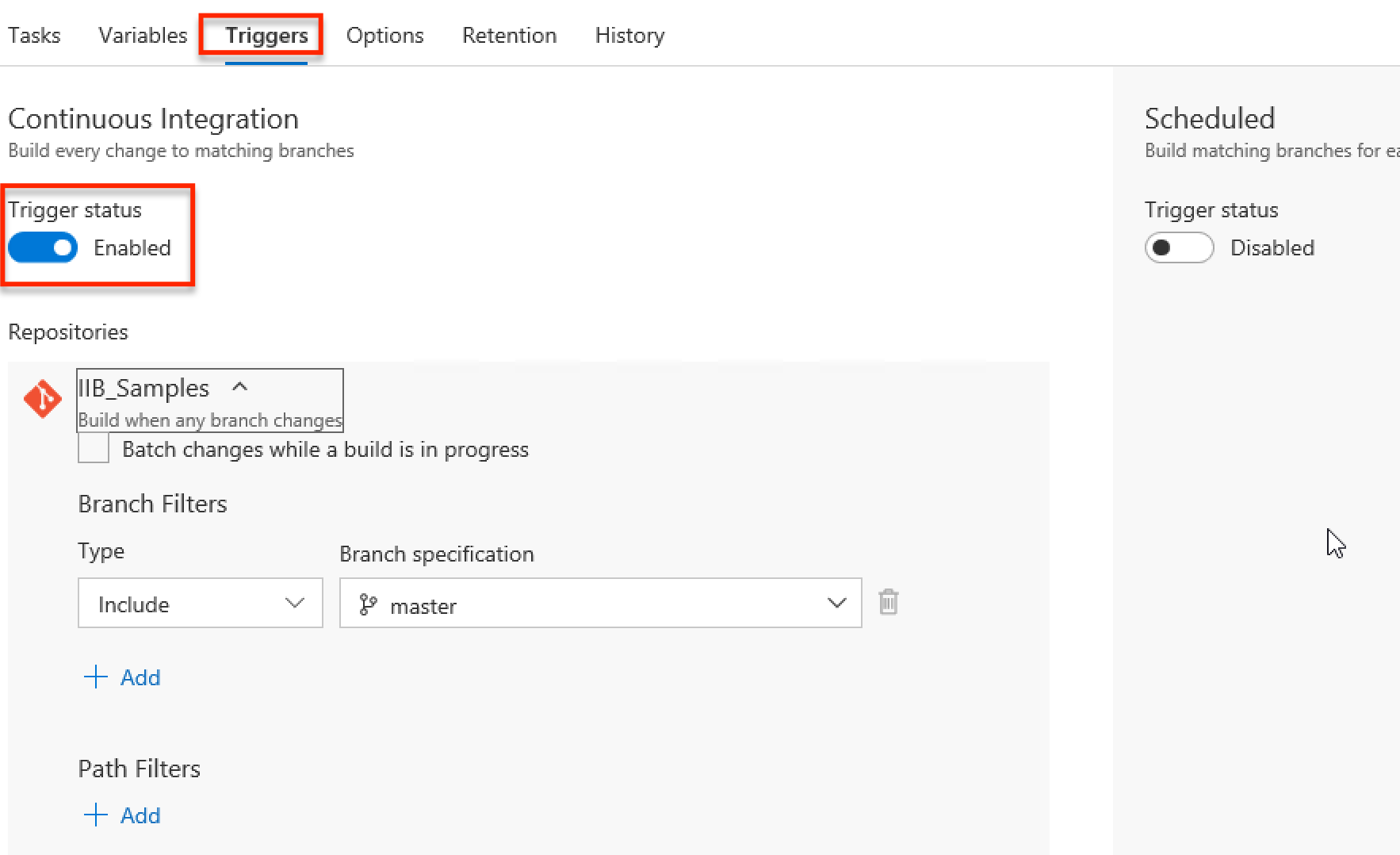


5. **Variables** > enter a **Value** for the **bar.name** **and bar.application** variables.

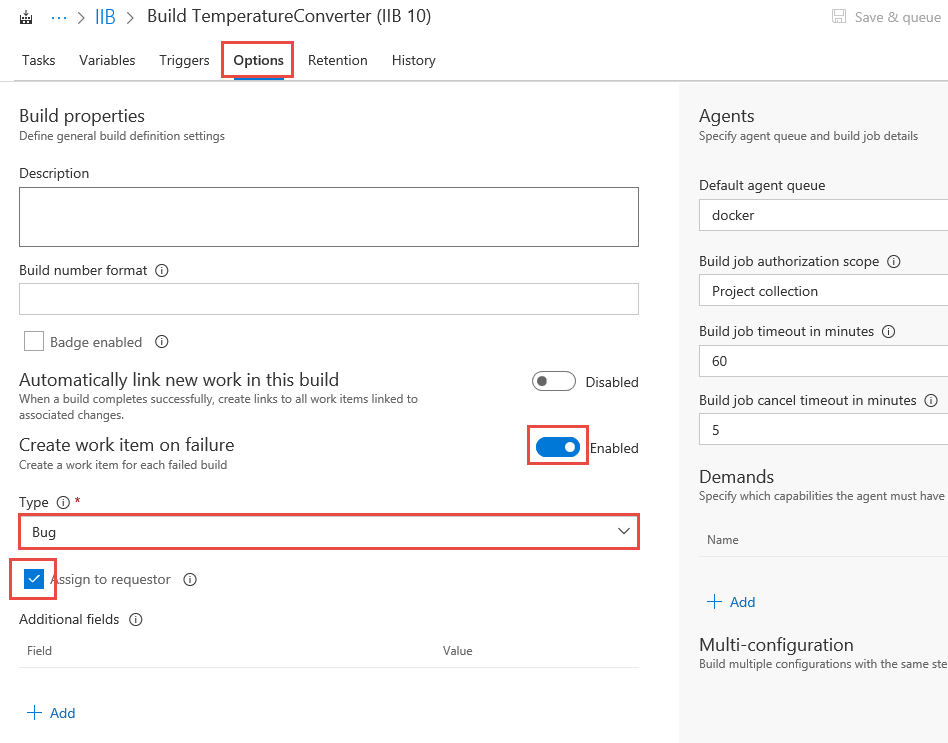


*See section V.1 for guidance on how to map variables to artifacts in Git.*

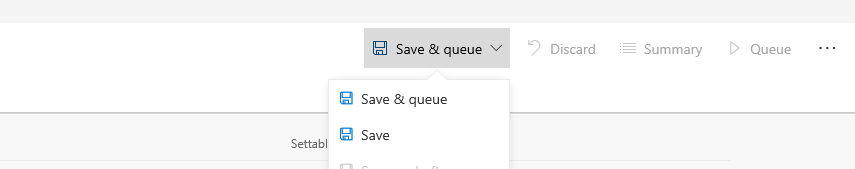
6. **Triggers** > in the **Continuous Integration** section, click to **Enable this trigger**.



7. **Options** > click to enable **Create work item on failure** > select **Bug** from the **Type** drop-down > ensure that **Assign to requestor** is selected.



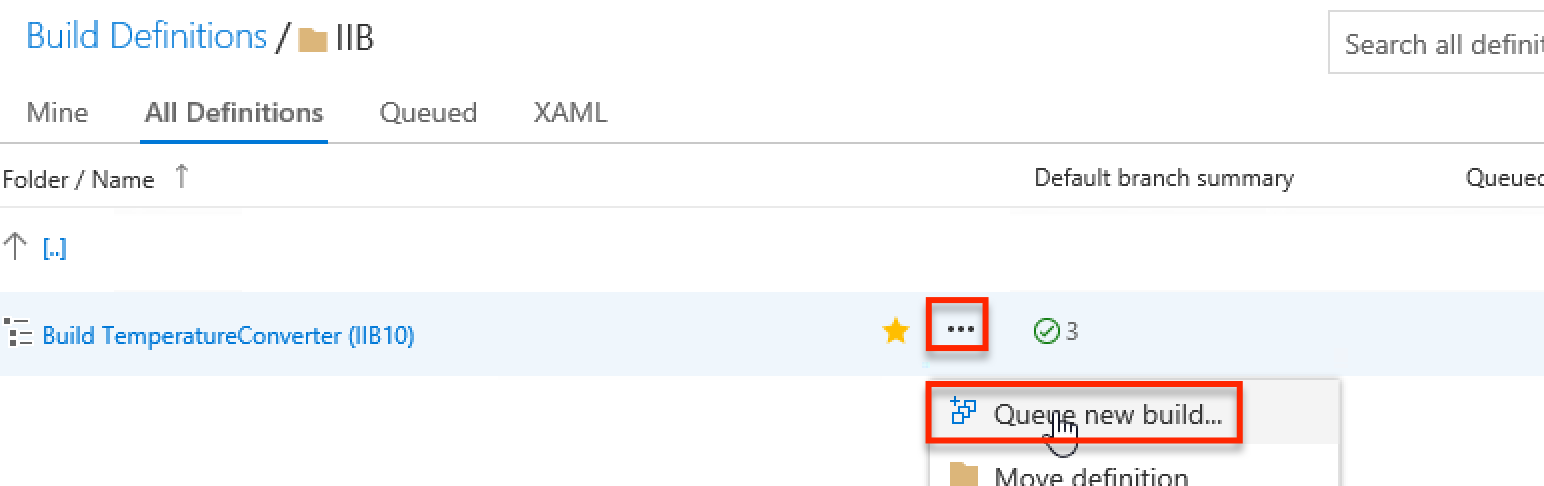
8. **Save & queue** > **Save** > confirm the **folder** and optionally add a **Comment** > **Save**.



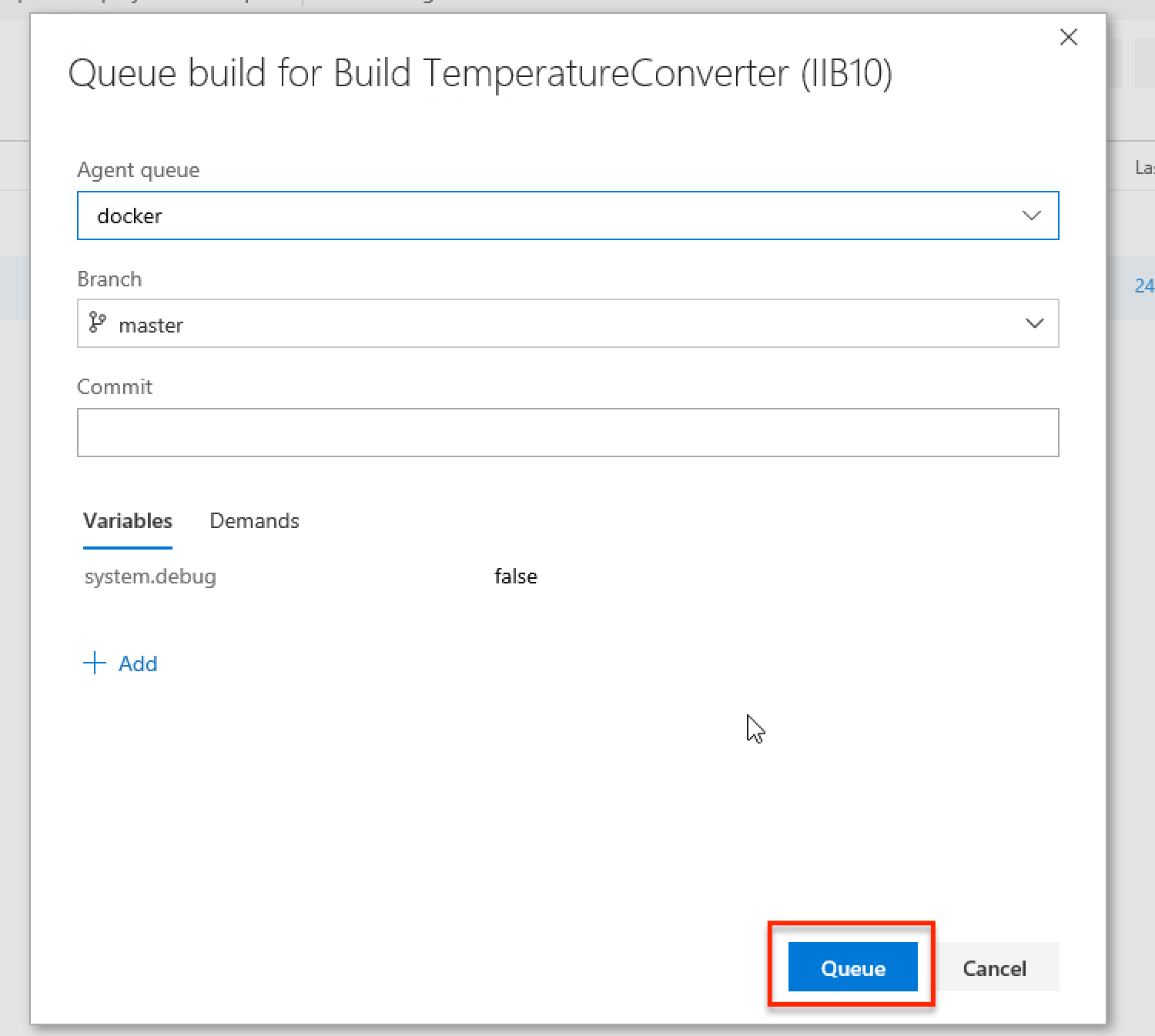
### Triggering a build

When you are ready, trigger a build from your build definition as follows.

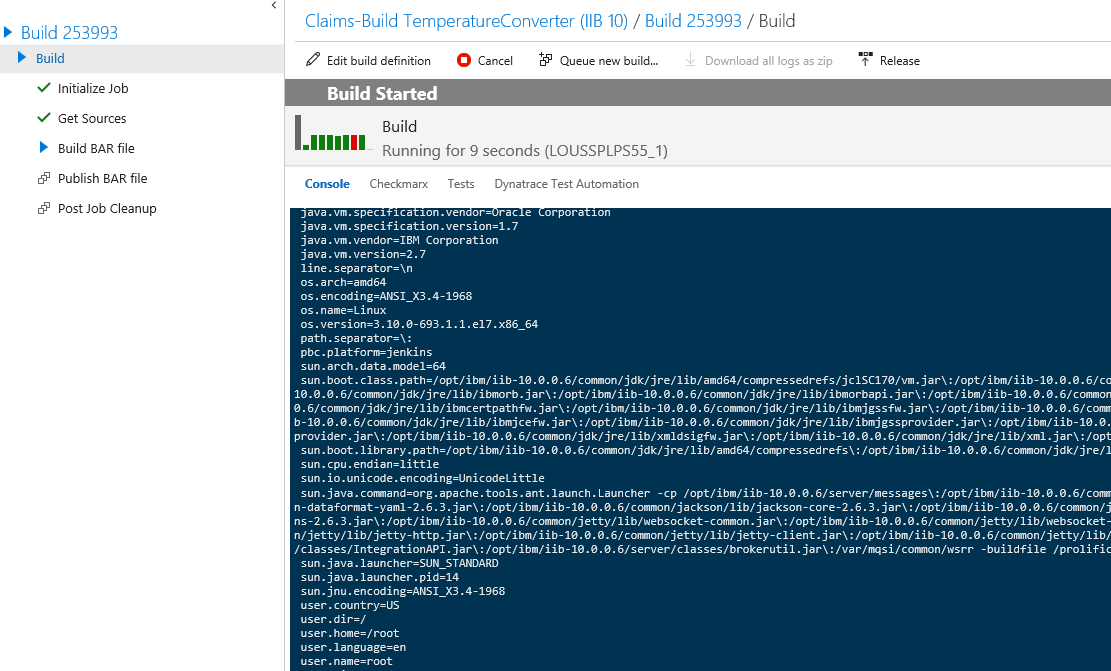
1. **Build and Release** > **Builds** > **All Definitions** > **IIB** > highlight the build definition you want to trigger a build for > from the menu (**…**) select **Queue new build…**.



2. Optionally review and modify the settings that will be used for your build > **OK**.



3. You will see a new build start up, picked up by a build queue, and then work through the build steps.



4. Once finished, your build will report either a success (as below) or failure.



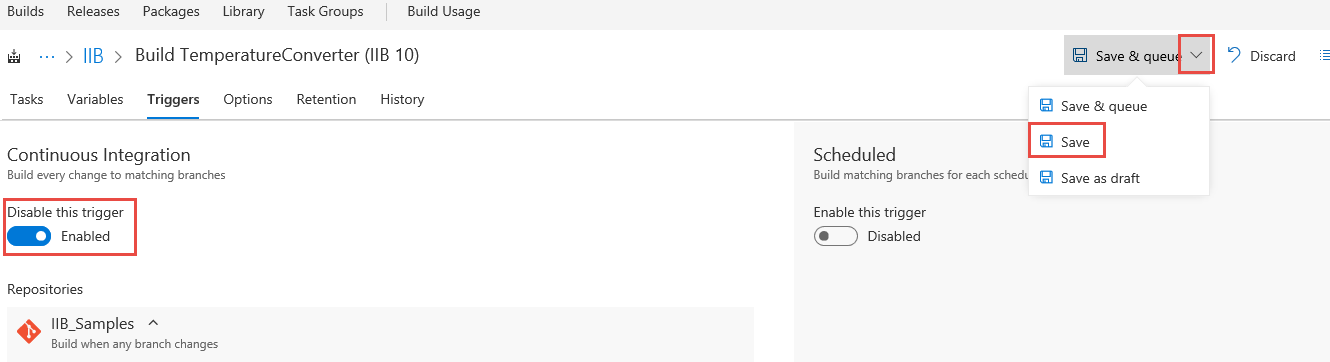
### Disabling continuous integration on your build

The steps for setting up a new build include the step to enable continuous integration. The steps below show switching that off.

1. **Build and Release** > **Builds** > **All Definitions** > **IIB** > click on the name of the build you wish to enable continuous integration for > **Edit**.

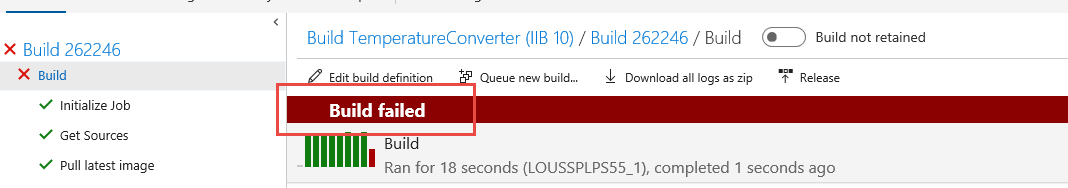


2. **Triggers** > in the **Continuous Integration** section, click to **Disable this trigger** > **Save & queue** drop-down > **Save**.

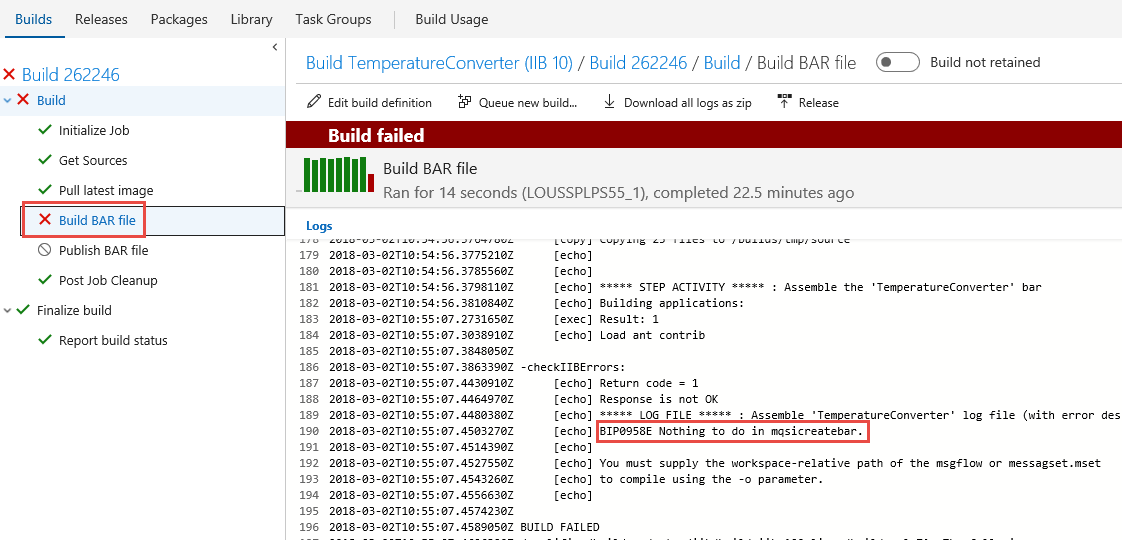


### Troubleshooting a failed build

1. A failed deploy will be marked as **Build failed**.



2. Click on the failed build step (marked with ) to see the logs > scan through the log to see what caused the failure.

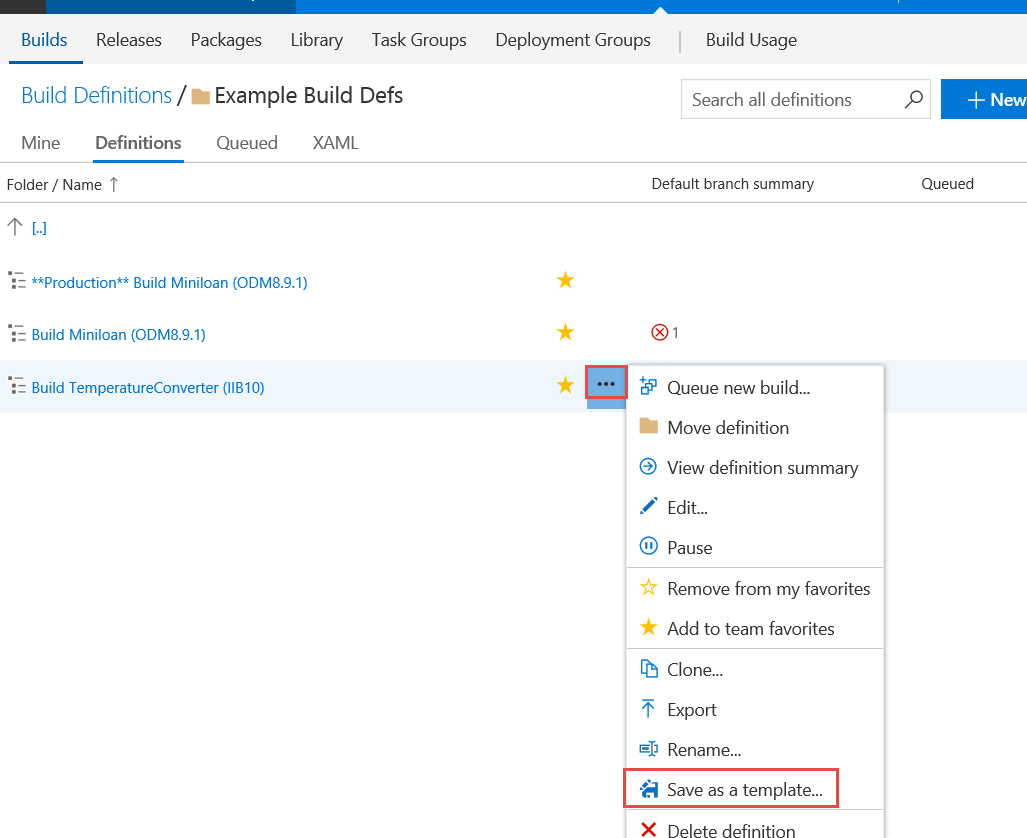


The list below shows some common failures.

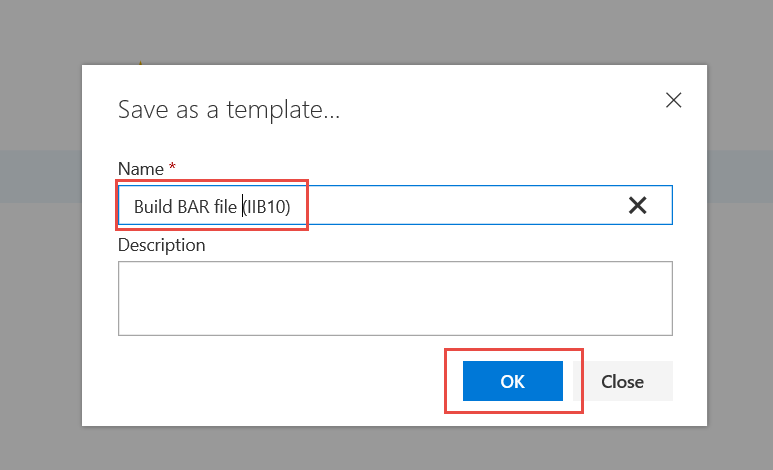
|  |  |  |
| --- | --- | --- |
| **Text in Log** | **Problem** | **Fix** |
| BIP0958E Nothing to do in mqsicreatebar | * No value provided for **bar.name** | Provide a value for **bar.name**. |
| BIP0960E incorrect “-a”, “-l”, “-s”, “-p”, or “-o” argument supplied to mqsicreatebar.  Project directory “…” does not exist. | * The Git repo you have linked to does not contain the application you wish to build. * Incorrect value provided for **bar.application**. | Ensure you have linked to the correct Git repo, and that the application you wish to build actually exists in that Git repo.  Correct the value applied to **bar.application** to make sure it corresponds to a project in the Git repo you are building from. |
| Workspace has errors on it after the build. In order to create bar there should be no errors in the workspace. | * There are errors in your source code in Git. * You are missing source files in Git. | Pull down the code from Git into IIT and check that there are no errors. Fix any errors you find.  Ensure that all of the source files required to build the application are in Git. |

### Creating a build definition template

1. From the list of build definitions > click the **…** menu for the build definition you wish to create a template from > **Save as a template…**.



2. Provide an appropriate **Name** for the template > **OK**.

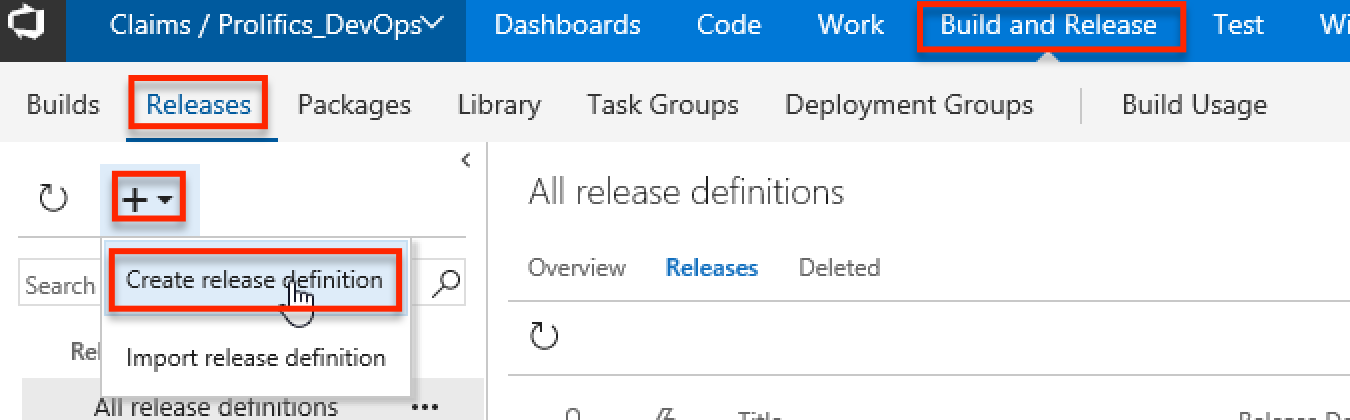


## Deploy

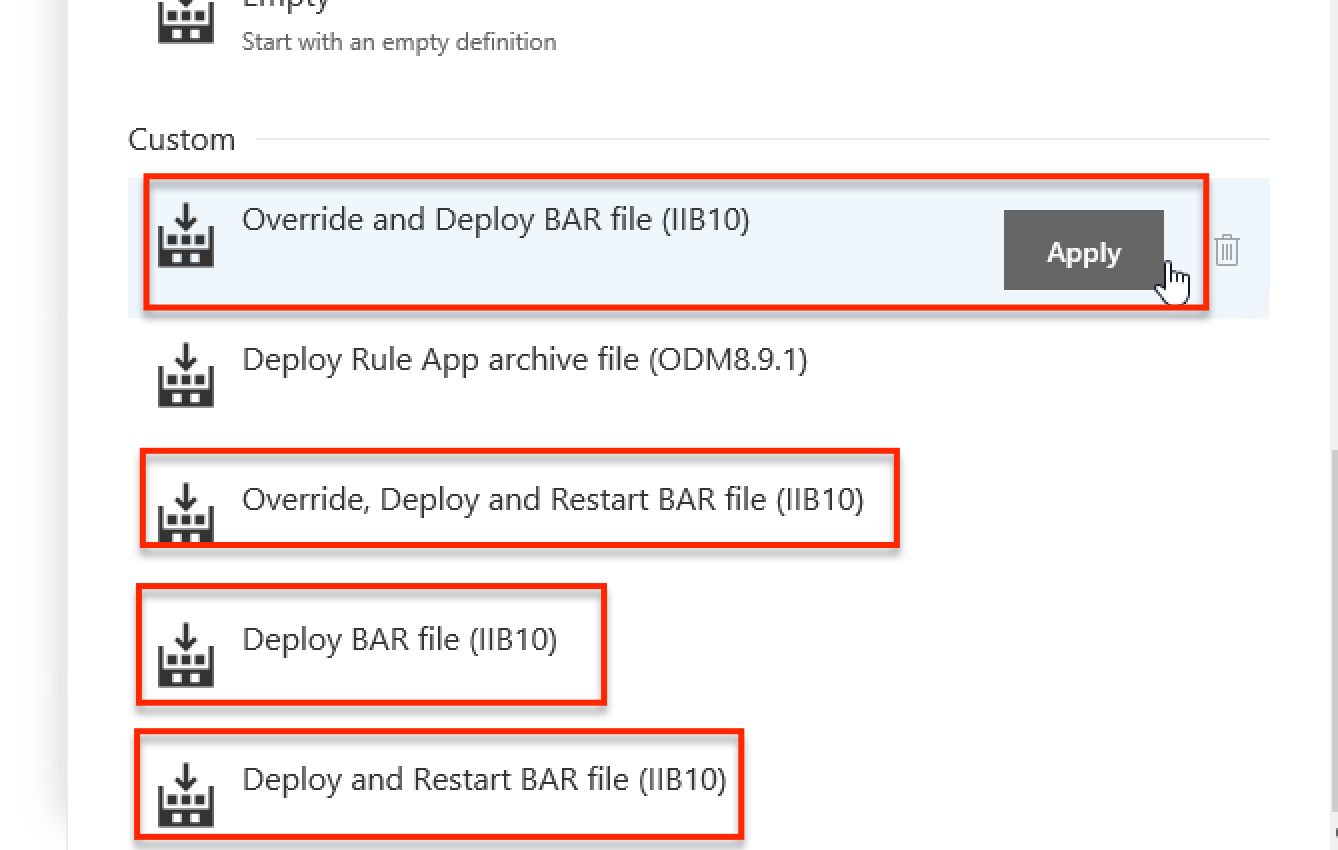
### Setting up a new release definition

When you are ready to start deploying your application, create a new release definition using the template. You will only have to do this once for an application.

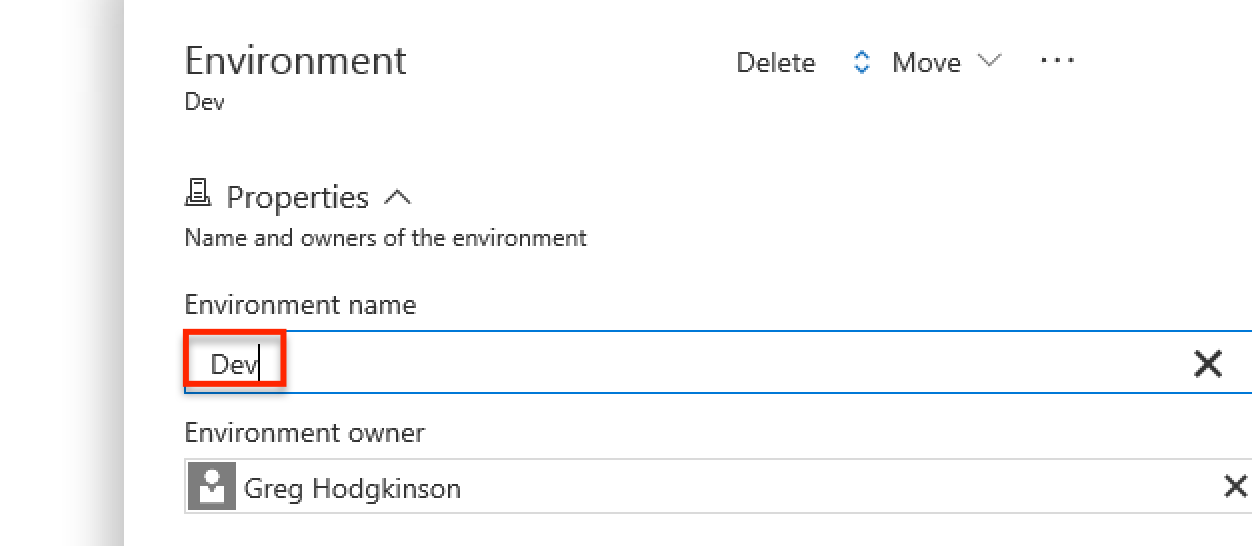
1. **Build and Release** > **Releases** > **+** > **Create release definition.**



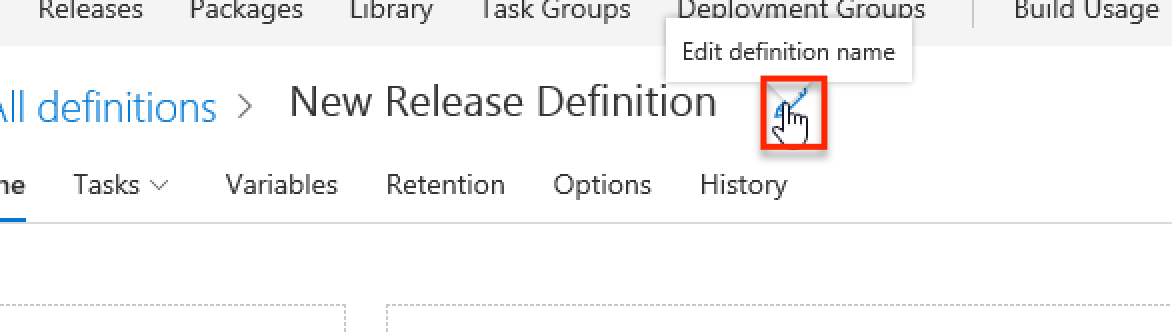
2. Under **Select a Template**, scroll down to the **Custom** section > select one of the deployment templates **> Apply**.



3. Set the **Environment name** for the first environment.

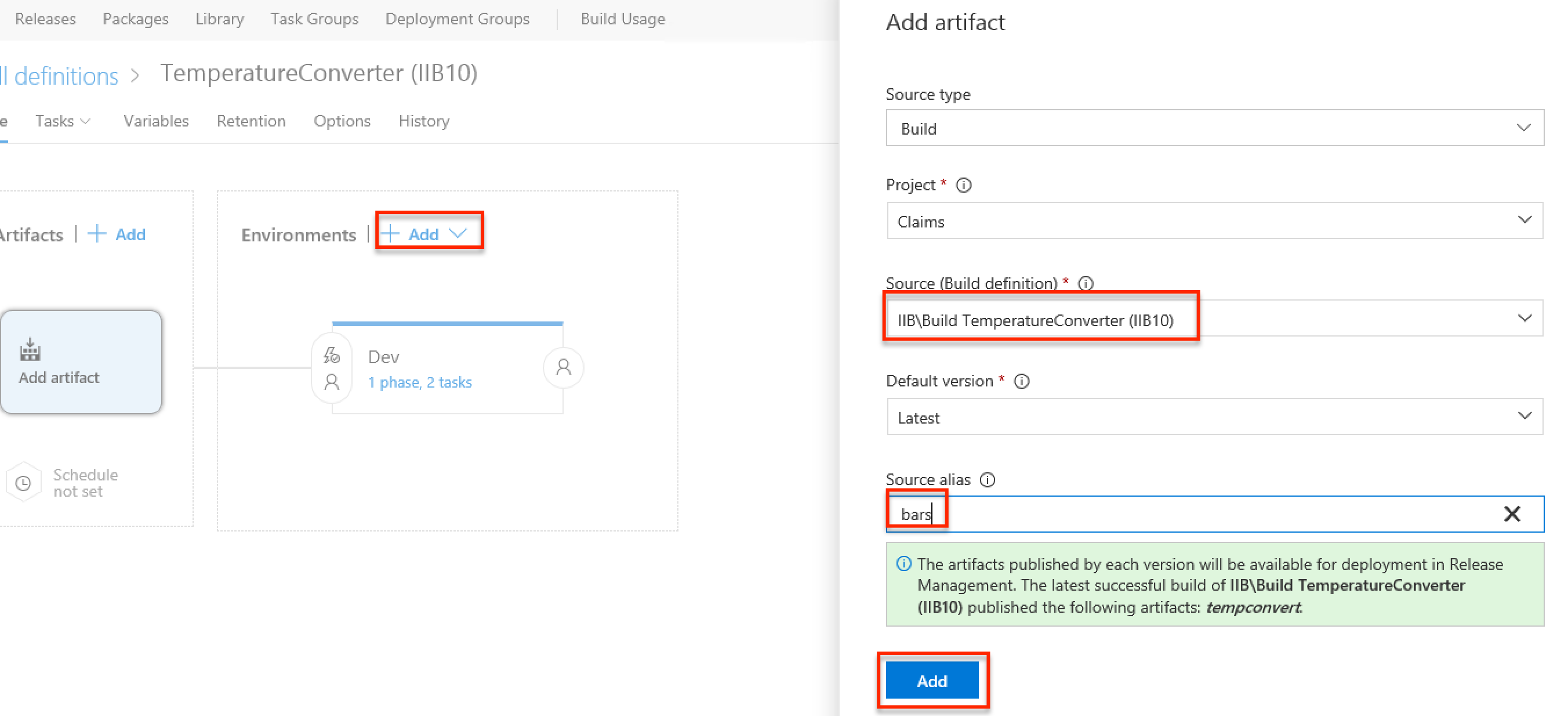


4. Click the **Edit** **definition name** button (it appears as a pencil next to default name of **New Release Definition**) > enter a suitable release definition name.

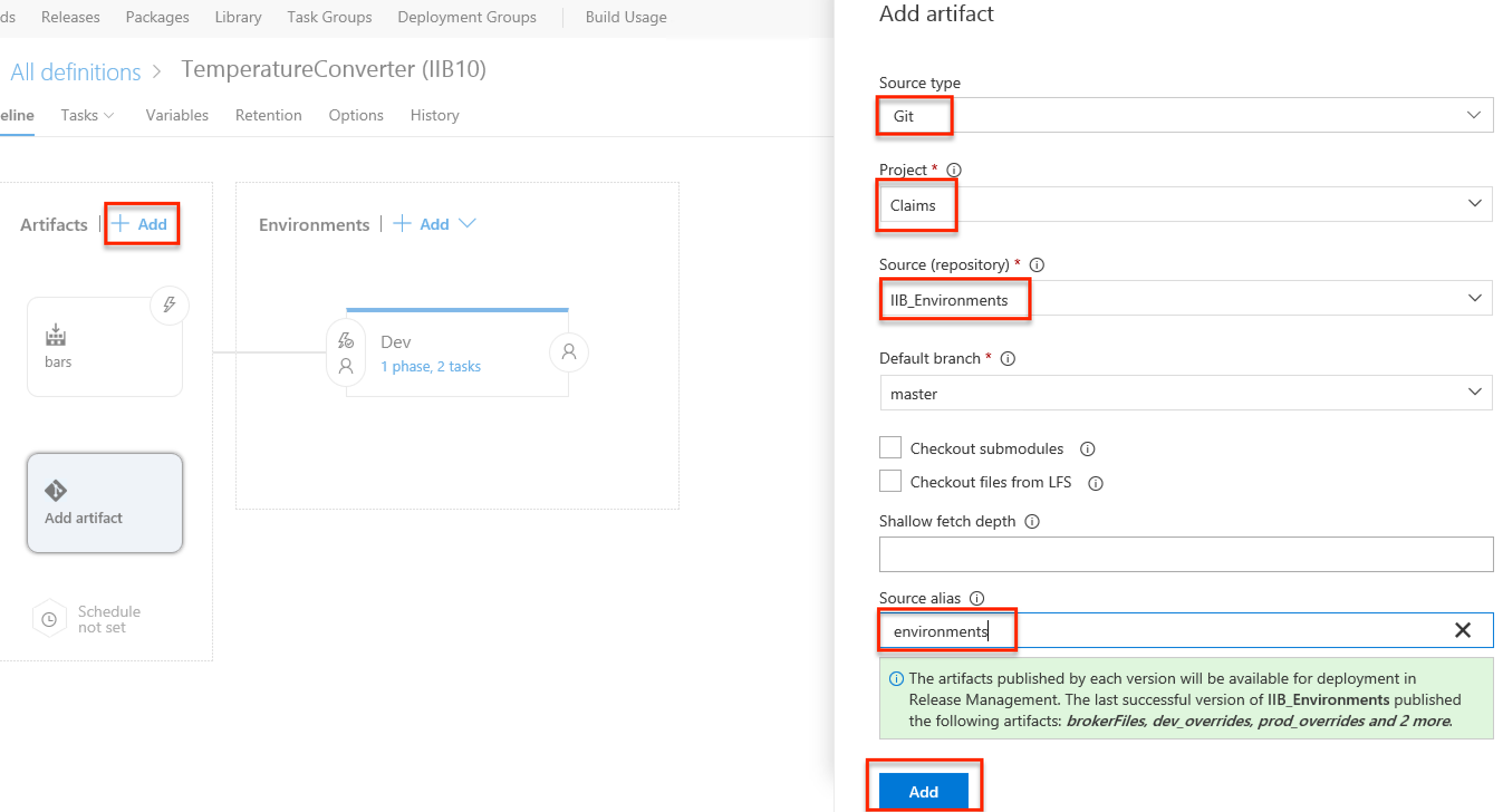


*For example, a good name for the release definition linked to the* ***Build TemperatureConverter (IIB10)*** *build definition would be* ***TemperatureConverter (IIB10)****.*

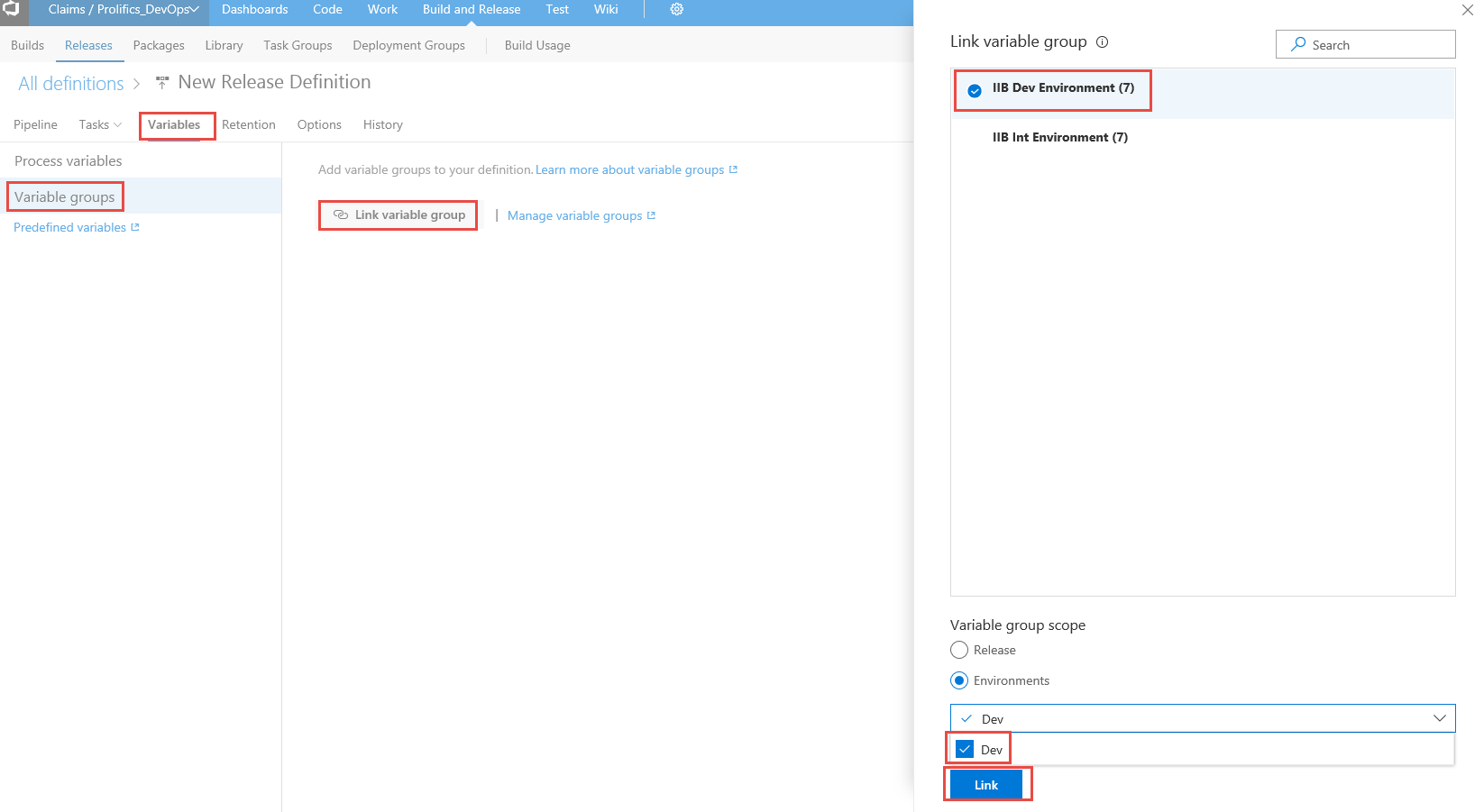
5. In the **Artifacts** section, click **+ Add** > select the **Project** that contains the build definition > select the build definition that you wish to define releases of from the **Source (Build definition)** drop-down > enter **bars** as your **Source alias** (*Important – don’t forget this step!)* > **Add**.



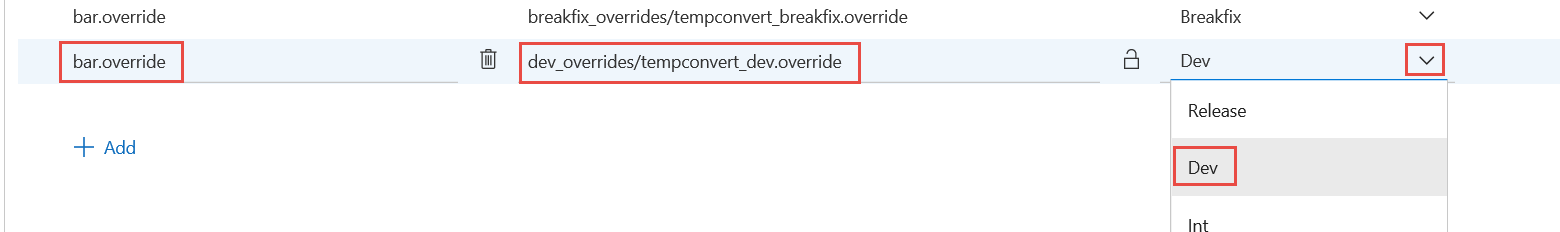
6. If you are using either a .broker file or a bar override file then you need to add a second source alias for the Git repository that holds those files. Click **+ Add** > select the **Project** that contains the Git repo > select the repo that your .broker/property files are in from the **Source (Repository)** drop-down > enter **environment** as your **Source alias** (*Important – don’t forget this step!)* > **Add**.



7. **Variables** > **Variable groups** > **Link variable group** > select the variable group that corresponds to the environment e.g. **IIB Dev Environment** for **Dev** > **set Variable group scope** to **Environment** > select your environment e.g. **Dev** > **Link**.



8. **Variables** > **Process variables** > **+ Add** to add a new variable > enter **bar.override** in **Name** field and then fill in the path to the bar override file in source control to use for overrides in the **Value** field > click the **Scope** drop-down and select the environment that you want this override to be applied to.

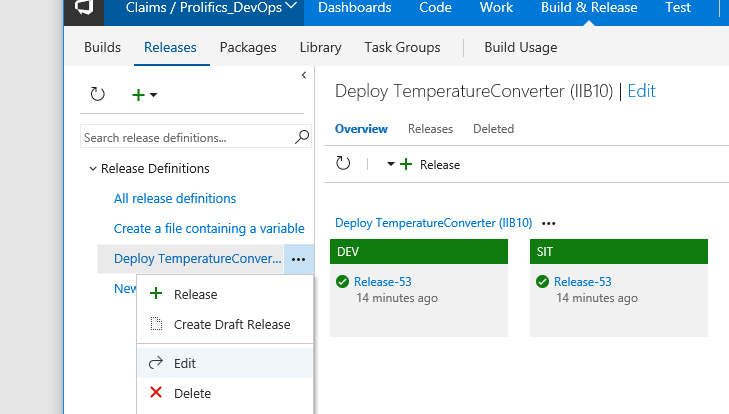


9. Click **Save** to persist all changes.

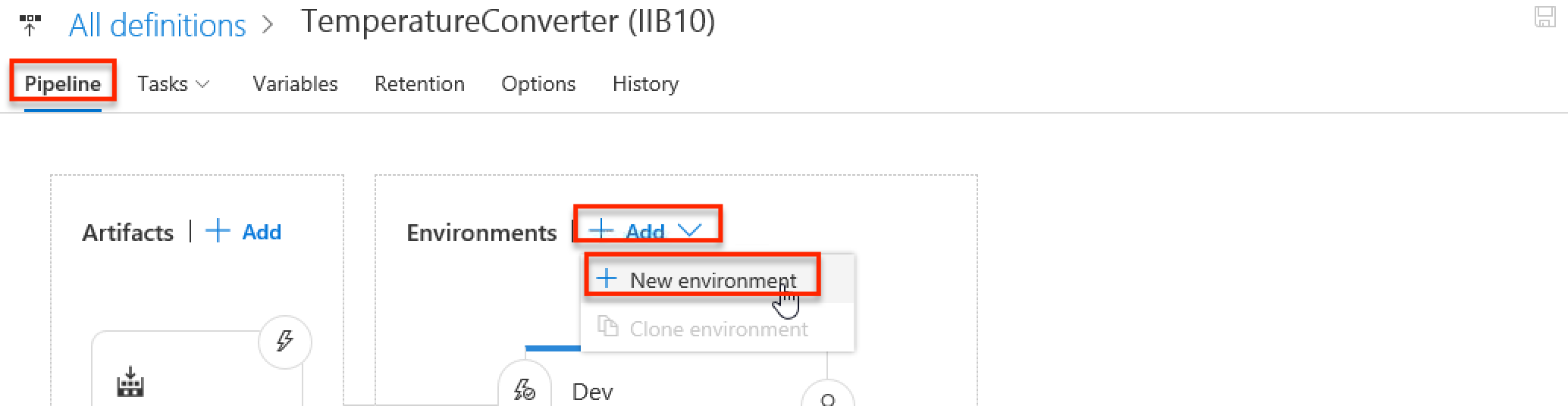
### Adding a new environment to a release definition

Release definitions have a single environment on creation. Follow the steps below to add additional environments for deployment.

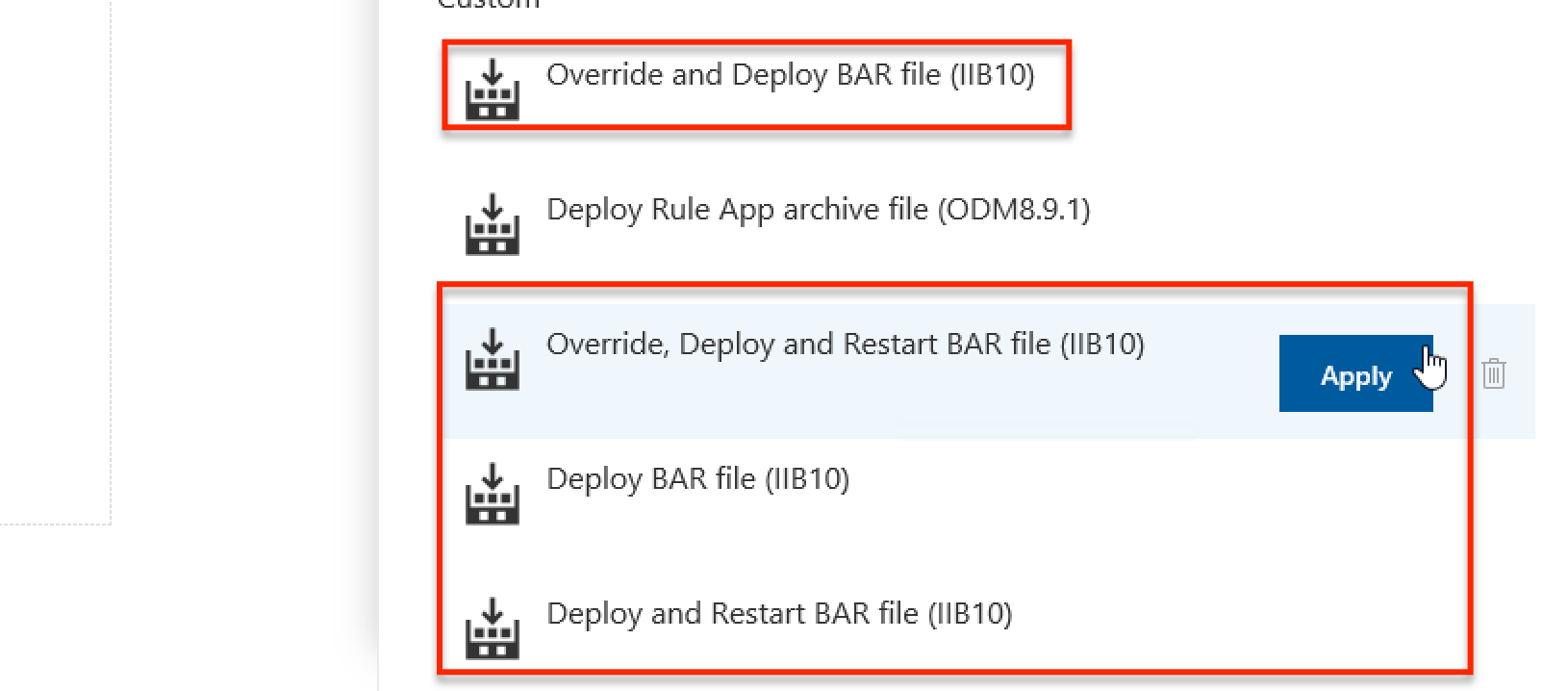
1. **Build and Release** > **Releases** > click on the (**…**) menu for the release definition you wish to add a new environment to > **Edit**.



2. In the **Pipeline** section, for **Environments**, click **+ Add >** **New environment**.

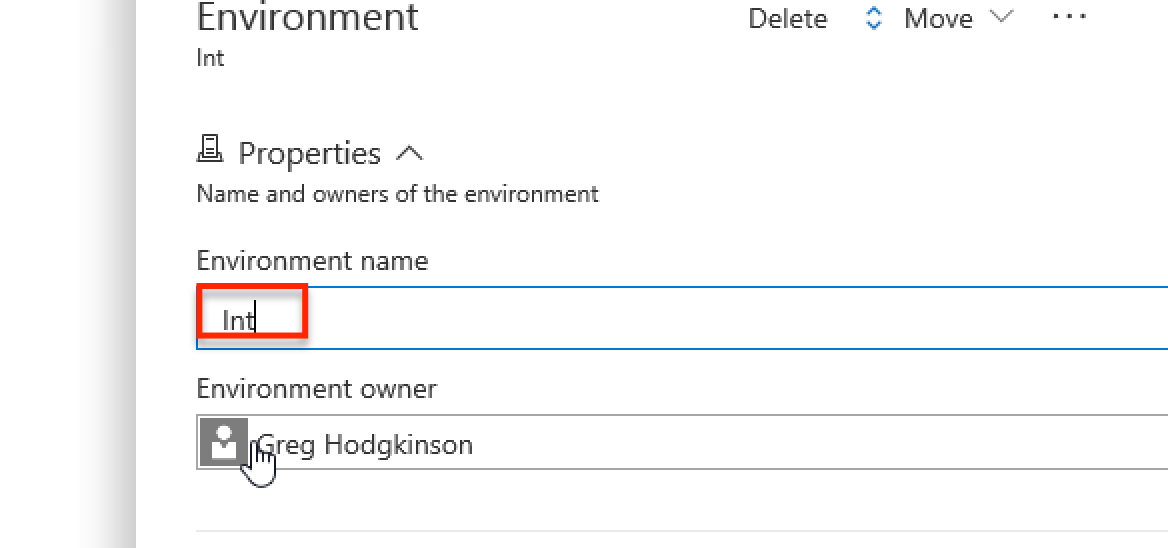


3. Under **Select a Template**, under the **Custom** section > then select a template (template name describes the actions it will perform) > **Apply**.

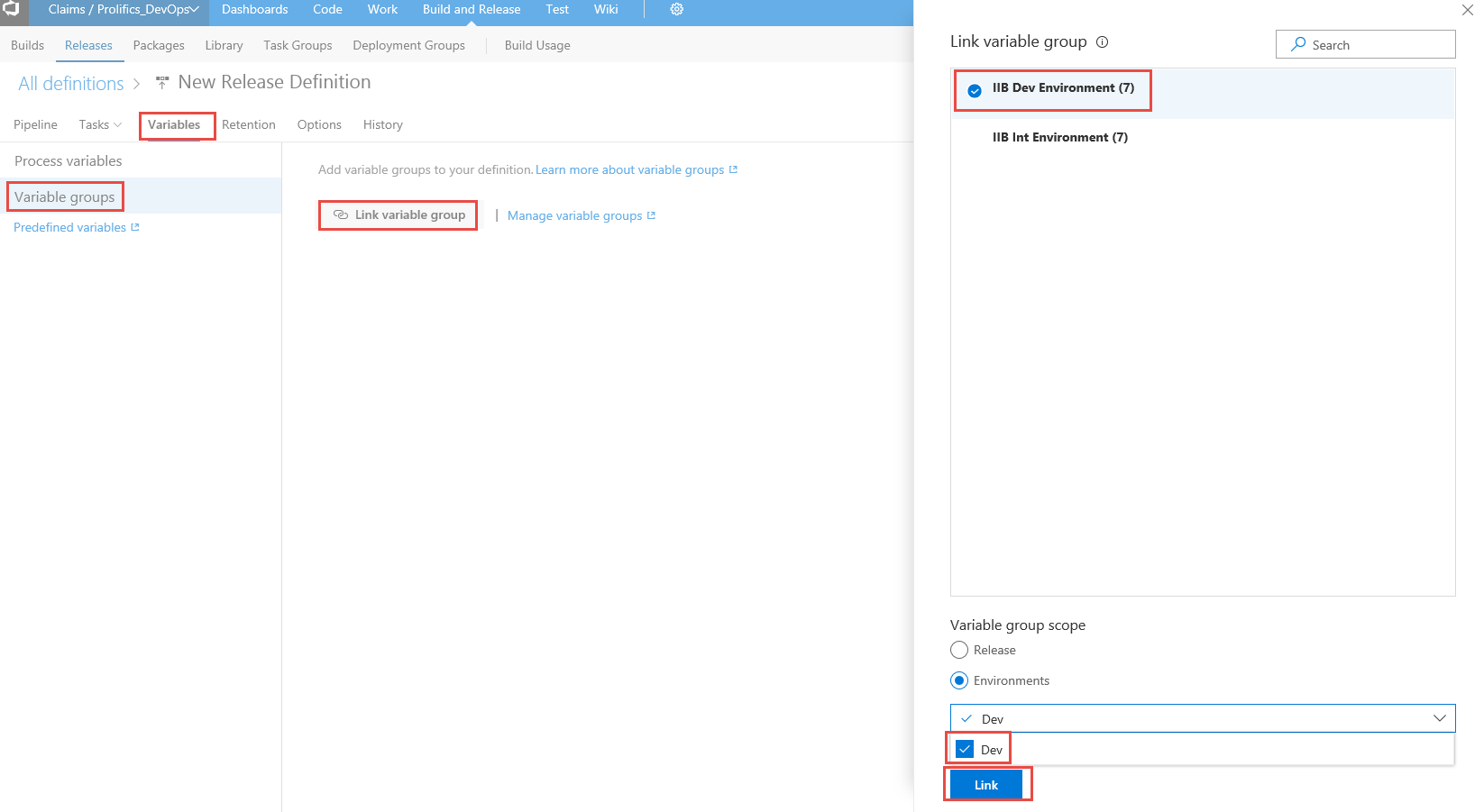


4. Click on the **Environment name** and replace it with a more meaningful name.

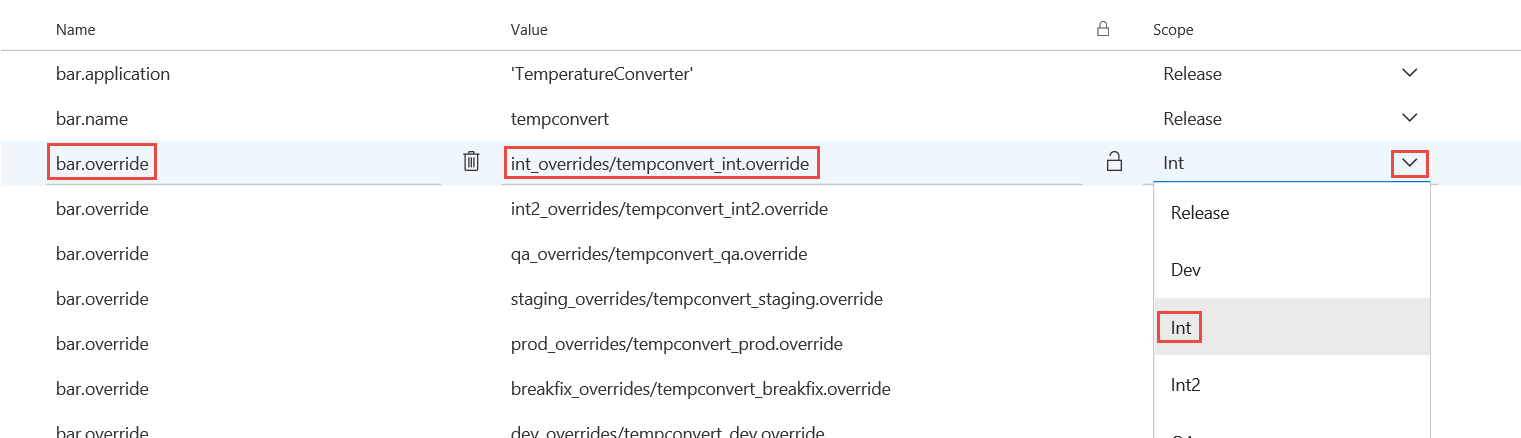
*For example, a good name for your first testing environment would be* ***Int****.*



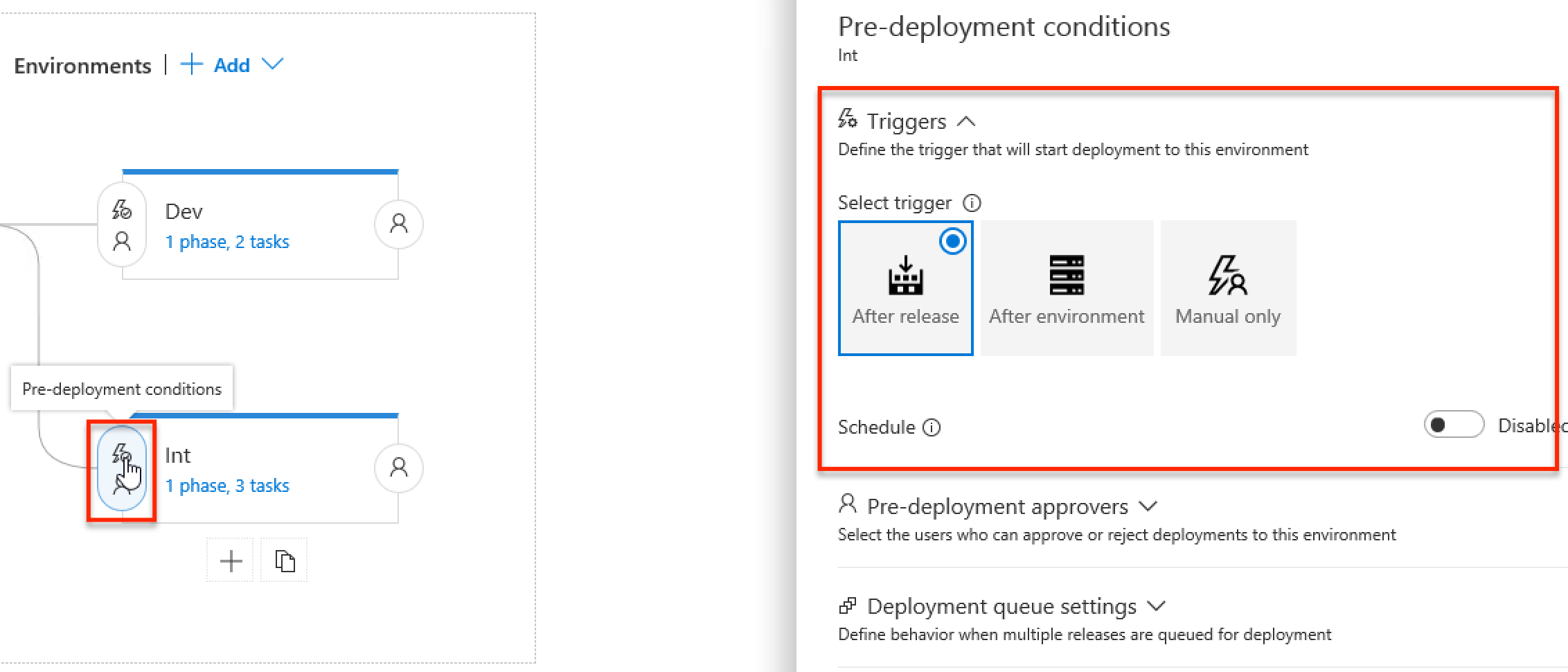
5. **Variables** > **Variable groups** > **Link variable group** > select the variable group that corresponds to the environment e.g. **IIB Int Environment** for **Int** > **set Variable group scope** to **Environment** > select your environment e.g. **Int** > **Link**.



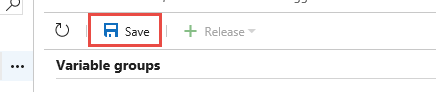
6. **Variables** > **Process variables** > **+ Add** to add a new variable > enter **bar.override** in **Name** field and then fill in the path to the bar override file in source control to use for overrides in the **Value** field > click the **Scope** drop-down and select the environment that you want this override to be applied to.



7. **Pre-Deployment Conditions** > optionally adjust the triggers for deployment to each of your environments.



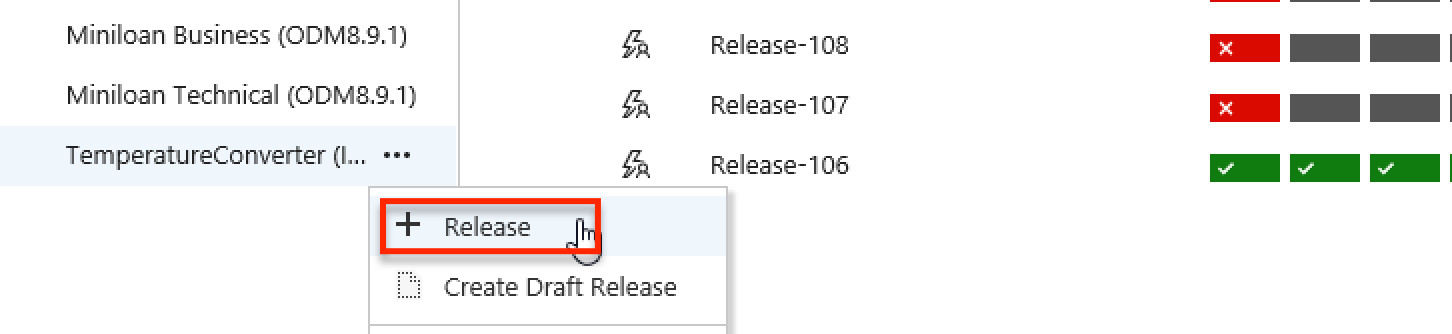
8. **Save**.



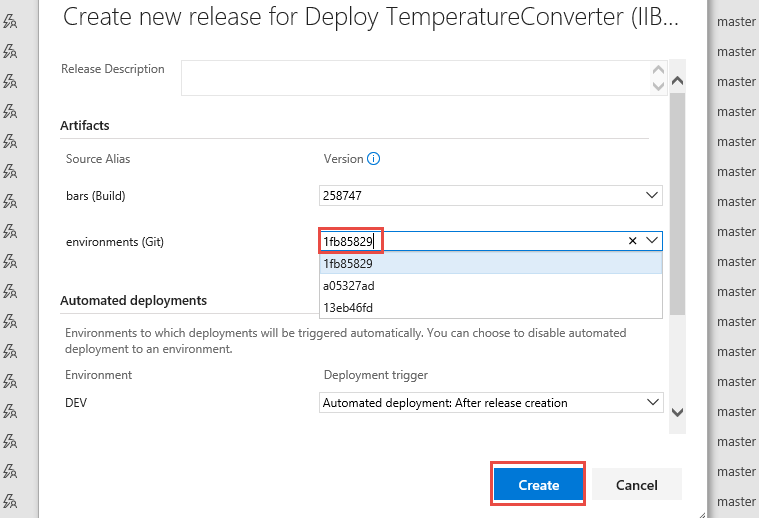
### Creating a release

Once you have run a build, you can create a matching release in order to push that build through the release pipeline of environments.

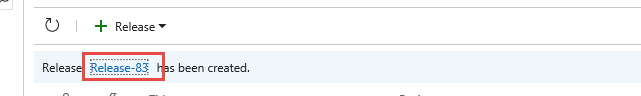
1. **Build and Release** > **Releases** > click on the (**…**) menu for the release definition you wish to create a release for > **Release**.

****

2. Ensure the correct **Version** is selected (that corresponds to the build you wish to release) > optionally, change the **Deployment trigger** > **Create**.



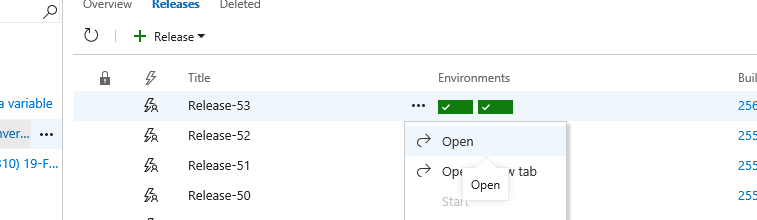
3. Click on the displayed shortcut to see the release you have created.



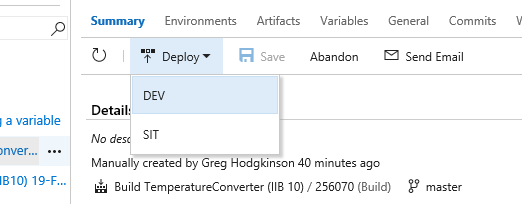
### Manually trigger the deploy of a release to an environment

If your release is not set up to automatically deploy to a specific environment, you can kick it off manually by following these steps.

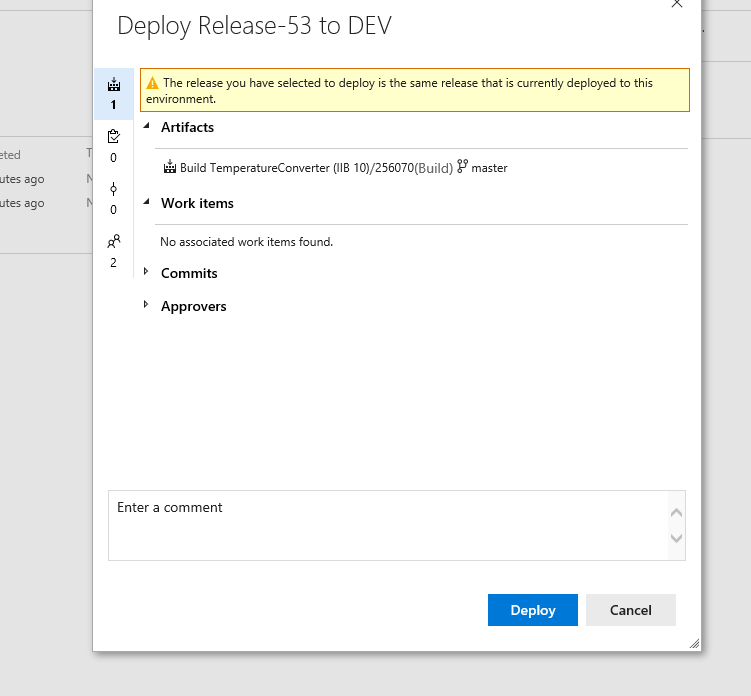
1. **Build and Release** > **Releases** > click on the name of the release definition you wish to do a deployment for > **Releases** > click on the (**…**) menu for the release you wish to deploy > **Open**.



2. Click on the **Deploy** drop-down and then select the environment to which you want to deploy.



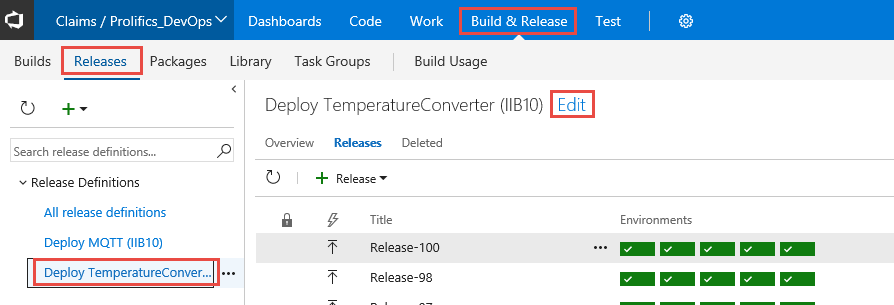
3. Review the deployment settings > **Deploy**.



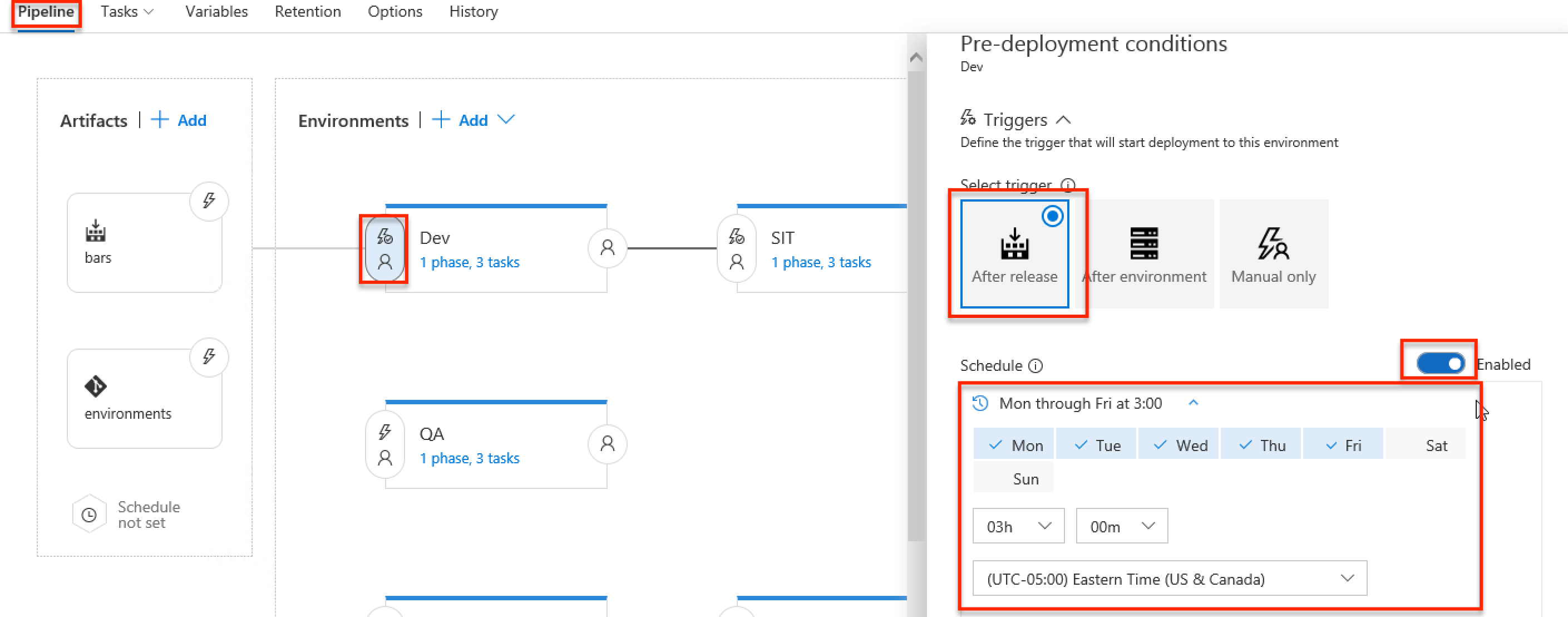
### Configuring a timetable-based scheduled deploy

It is possible to set up a release so that once it has been deployed successfully to a certain environment (like staging), it will then be automatically deployed to a further environment (like production) at a specified time based on a defined timetable/schedule.

1. **Build and Release** > **Releases** > click on the name of the release definition you wish to configure the scheduled deploy for > **Edit**.



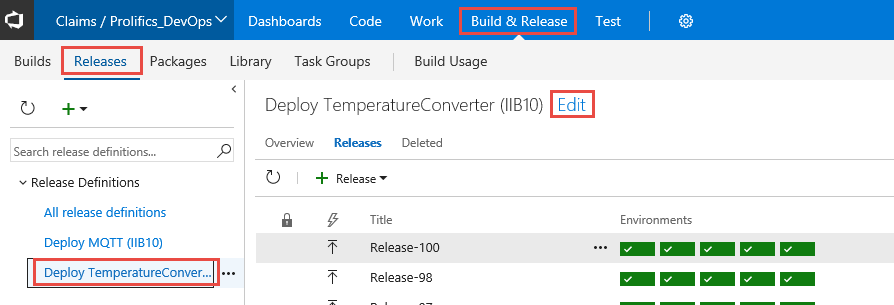
2. In the **Pipeline** section > click on the **Pre-deployment Conditions** for the environment you wish to configure > **Triggers** > select **After release** > select **Enabled** > configure **Schedule** > **Save**.



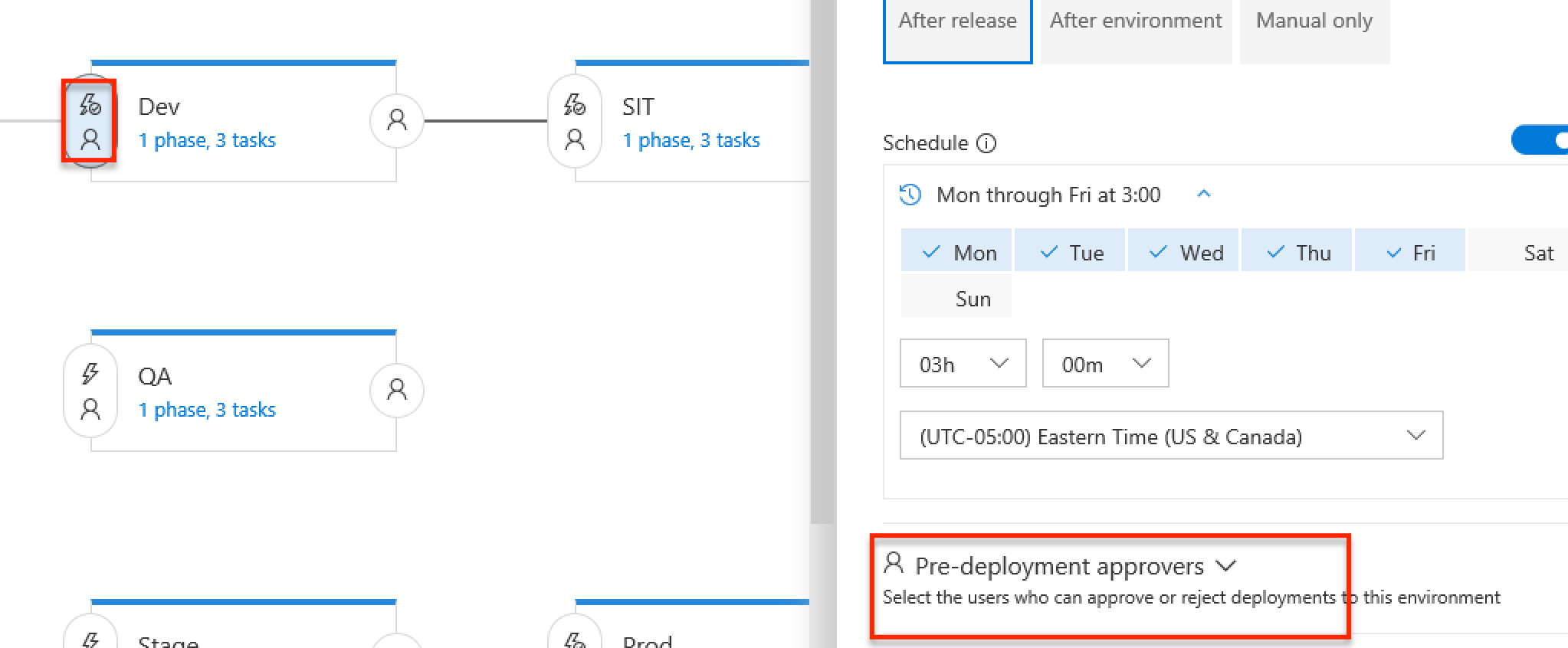
### Configuring deploy approval and deferring deployment

You can set up manual approval for a deployment, after which the approver can defer deployment to a specific date/time. In order to setup manual approval for deployment, follow these steps.

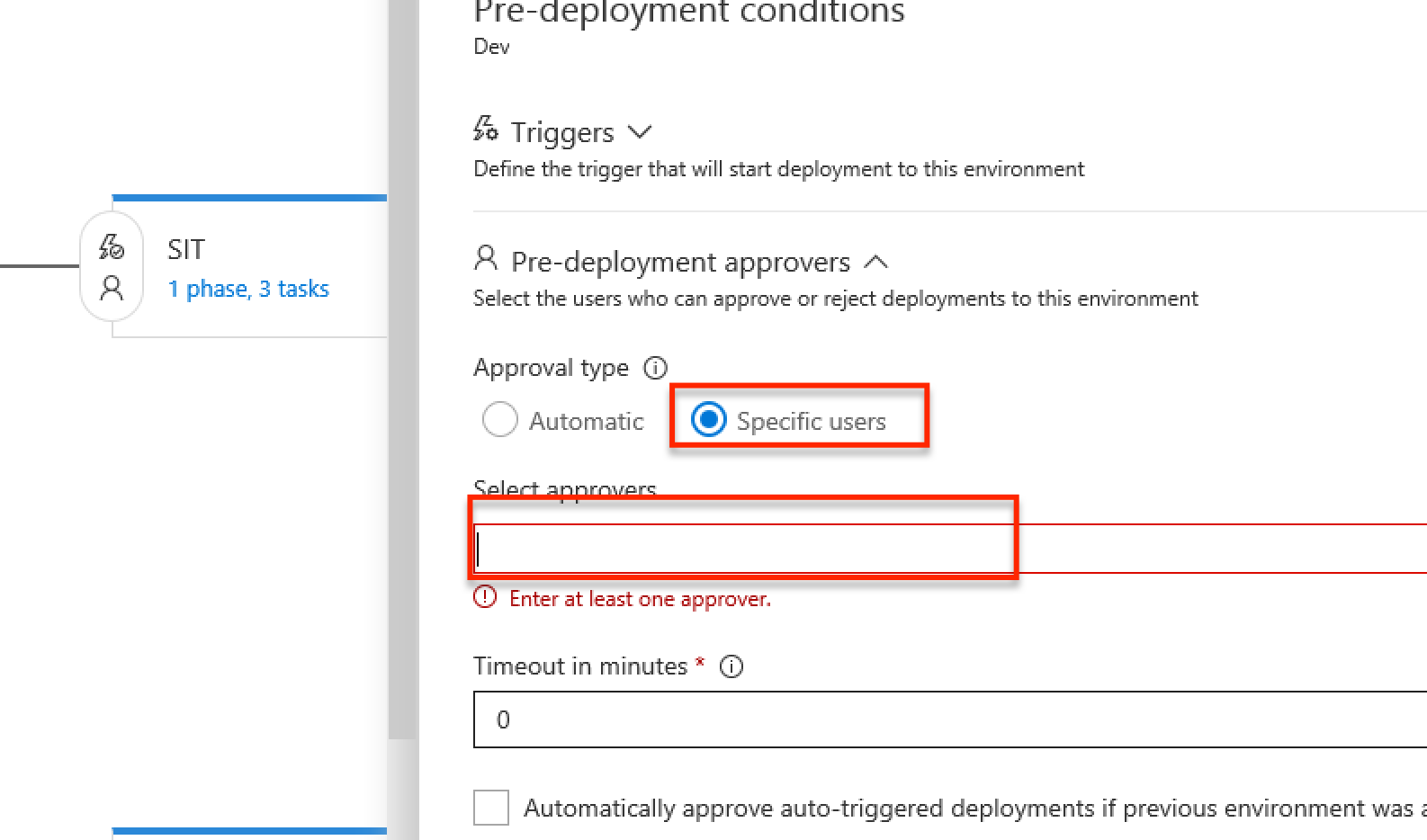
1. **Build and Release** > **Releases** > click on the name of the release definition you wish to configure the scheduled deploy for > **Edit**.



2. In the **Pipeline** section > click on the **Pre-deployment Conditions** for the environment you wish to configure > **Triggers** > select **Pre-deployment approvers** > **Save**.



3. For **Pre-deployment approver**, select **Specific Users** > enter the names of the users you wish to approve deployment > **Save**.

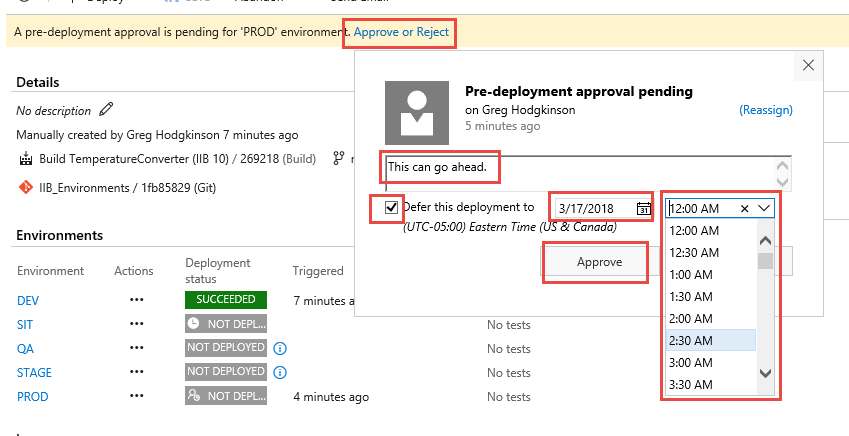


On triggering of the deployment, the following steps will occur.

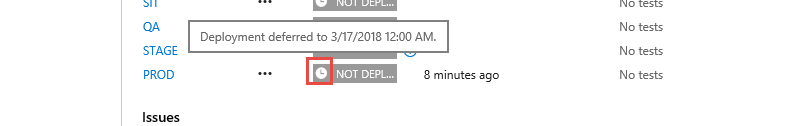
1. Each approver will receive an email such as the one below. Click on the **View approval request** link to open the approval.



2. Click on the **Approve or Reject** link > enter comments > select **Defer this deployment to** > enter a date for deployment > enter a time for deployment > **Approve**.

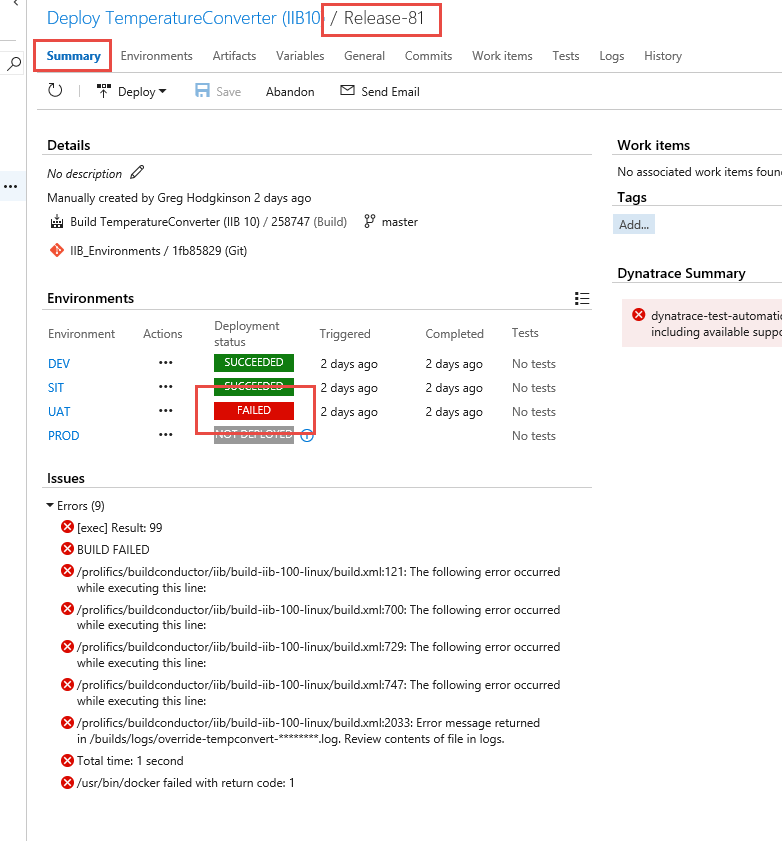


3. The clock icon will show the deploy is happening in the future and hovering over it will show when it is scheduled for.

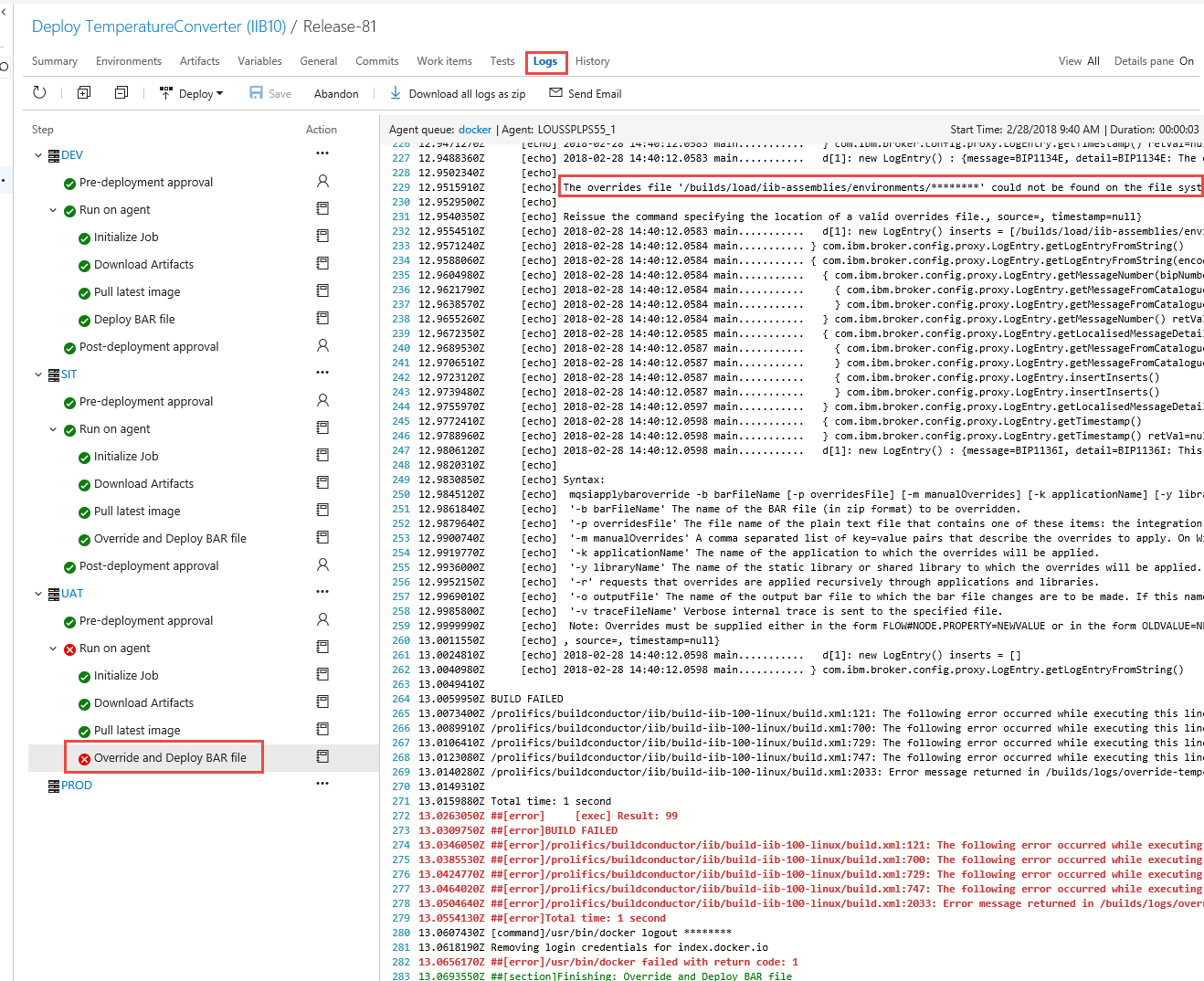


### Troubleshooting a failed deploy

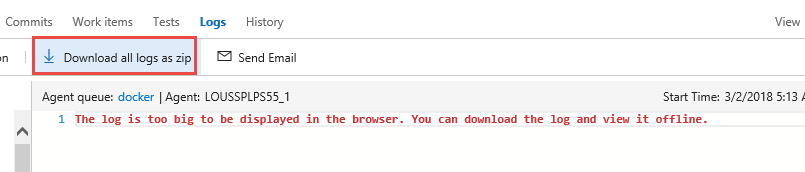
1. A failed deploy will be marked as such on the release **Summary** page.



2. To see more details, switch to the **Logs** page to see the full logs produced > select on failed **Step** (indicated with the  icon) > scan through the log file to see what caused the failure.



If you see the message, **The log is too big to be displayed in the browser. You can download the log and view it offline**,then click on the **Download all logs as zip** link to download an offline copy of all the logs.

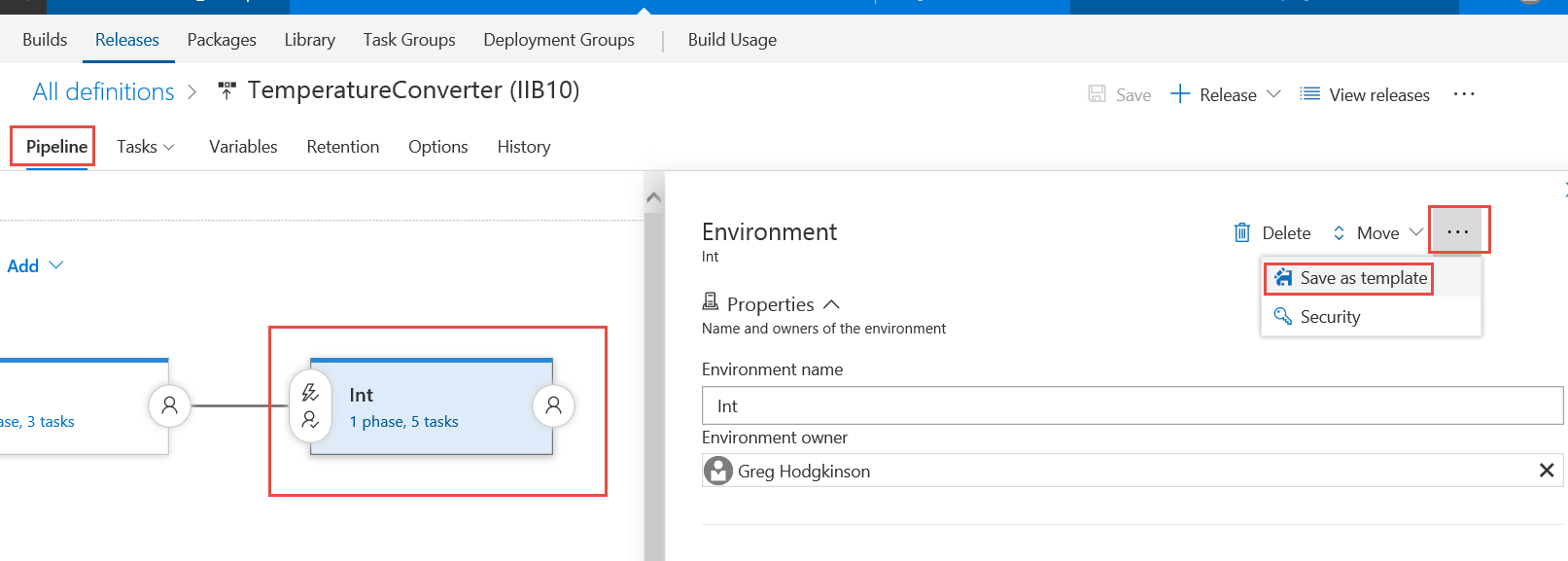


The list below shows some common failures.

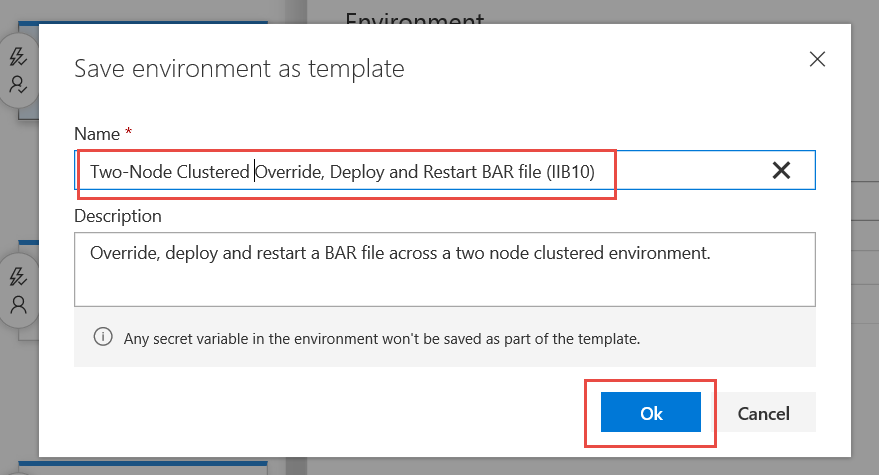
|  |  |  |
| --- | --- | --- |
| **Text in Log** | **Problem** | **Fix** |
| The override file “…” could not be found on the file system. | * No value provided for **bar.override** * Incorrect value provided for **bar.override** * Forgot to link environments Git repo, or forgot to rename source alias to **environments**. | Ensure that you have linked the Git repo containing the override file, renamed the repo’s source alias to **environments**, and specified a value for **bar.override** that indicates the path to the override file. |
| A utility was requested to read connection parameters from the properties file ‘.broker’, but this file was not found, or contains invalid information. | * Incorrect value provided for **iib.brokerFile** * Forgot to link environments Git repo, or forgot to rename source alias to **environments**. | Ensure that you have linked the Git repo containing the .broker file, renamed the repo’s source alias to **environments**, and specified a value for **iib.brokerFile** that indicates the path to the .broker file. |
| Authentication failure when connecting to the secured remote integration node ‘…’. Please specify a valid userid and password. | * Incorrect or missing values for **iib.userId** or **iib.password**. * If using a .broker file, incorrect value for user id provided in the .broker file used. | Update the values for **iib.userId** and **iib.password** on the environment to be correct.  If using a .broker file, make sure the correct value is provided for the user id in the file. |
| BIP1921S: The integration node cannot be reached. Check that the integration node is running… | * Integration node might not be running. * Incorrect values provided for **iib.host**, **iib.port**, **iib.qMgr**, **iib.execGrp**. * If using a .broker file, incorrect values provided for host, node or listener port in that file. | Ensure the node is running.  Check the values for the environment variables or the .broker file contents to ensure they are correct. |
| The utility was asked to deploy ‘…’ but the file could not be read… | * **bar.name** variable not correct. * Forgot to rename the source alias for the build to **bars**. | Fix the value for **bar.name** so it corresponds to the value used in the matching build. Remember that you must not provide a .bar extension in the name.  Check that you renamed the source alias for the build to **bars**. |
| A request was made to access the file ‘…appzip’ in ‘…’ and this file could not be found. | * Either missing or incorrect value provided for **bar.application.** | Check that the correct value has been provided for bar.application, that this corresponds to the name of the application in the bar file, and that the name of the application is surrounded in single quotes i.e. 'appname' |

### Creating a release definition template

1. From the **Pipeline** of the release definition you want to create a template from > select the environment you want to create a template from > **…** > **Save as template**.



2. Enter a **Name** for the template > **OK**.



# Upgrade Considerations

## Upgrade of TFS

TFS upgrades should not have any effect on the pipeline.

## Upgrade of Docker

Upgrades of Docker on the TFS agent machines should not have any effect on the pipeline.

## Upgrade of PBC

Updated PBC images published to the Docker registry are automatically found and pulled down before the PBC automation steps run.

## Upgrade of IIB and IIT

If the development team moves to a new version of IIB, this may require the creation of a new PBC image that supports automation for that specific version of IIB.

Steps:

1. Get the install files for the new version of IIB and place them on the file share referenced by the PBC image Dockerfile.

2. Update references to the IIB install files in the Dockerfile to refer to this new version.

3. Rebuild the Docker image using the Dockerfile.

4. Push the new image to the Docker registry used during PBC automation steps.

5. Create new versions of the Build BAR file (IIB x.x) and Deploy BAR file (IIB x.x) templates with the new version being reflected in the template name i.e instead of x.x it will reflect the new IIB version.

6. Setup new release pipelines using the new templates.

# Reference

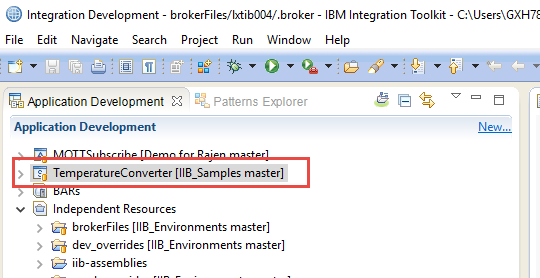
## Mapping Variables to Artifacts

This section illustrates how variable values map to various artifacts.

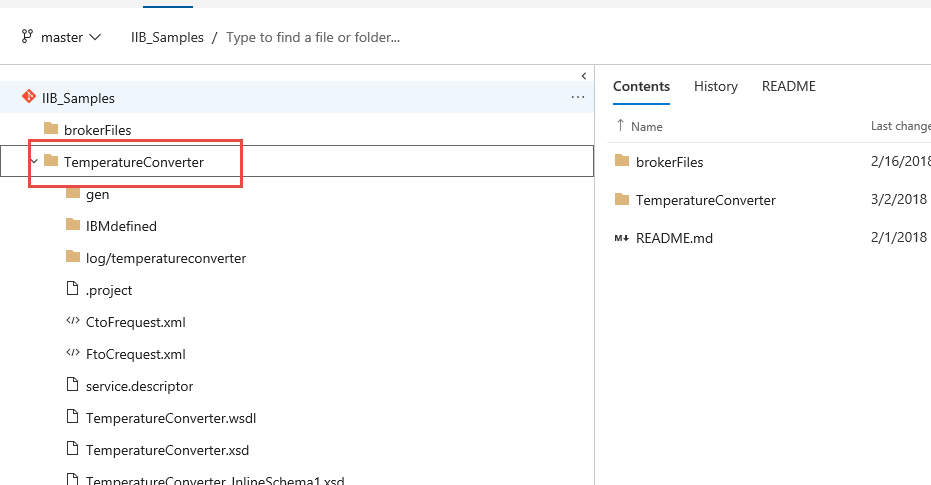
### BAR application name

The **bar.application** variable must be set to the name of the applications that you wish to include in the BAR file. You can find this either in your Toolkit, or by looking in the Git repo. Note that you can add multiple applications by placing a space between each application name and surrounding the entire list in single quotes.

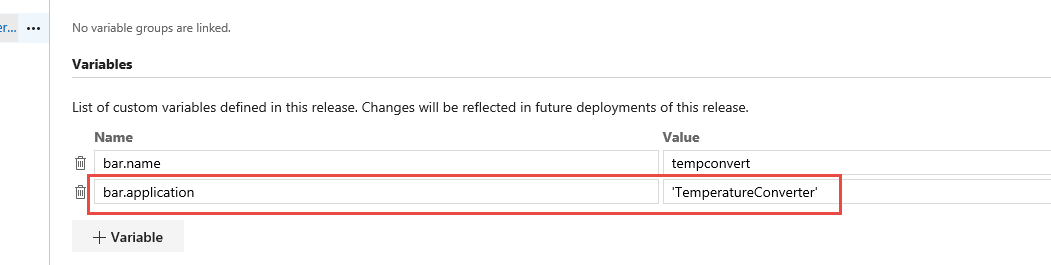
Application in toolkit…



Application in Git repo…



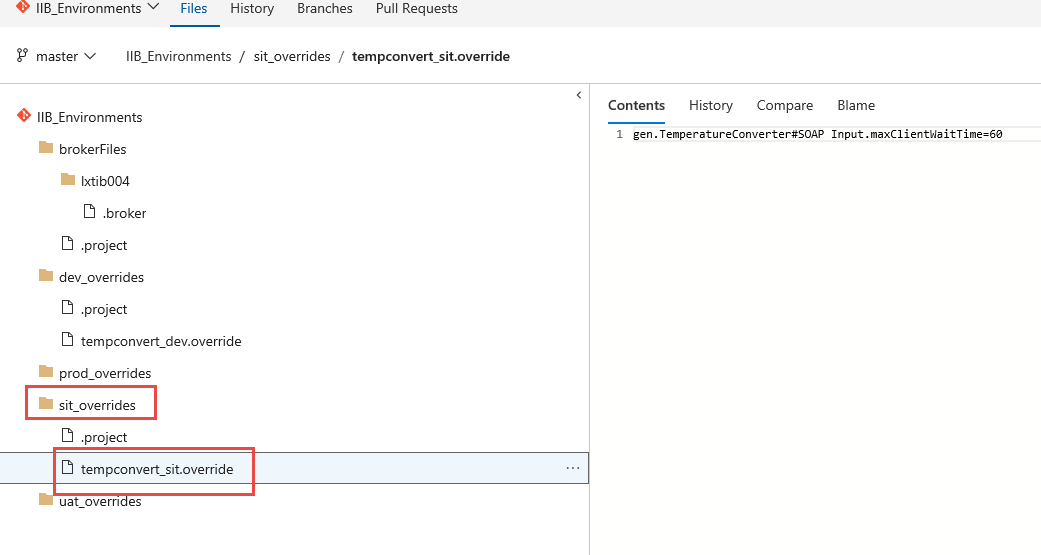
Variable (release) with value set for this application…



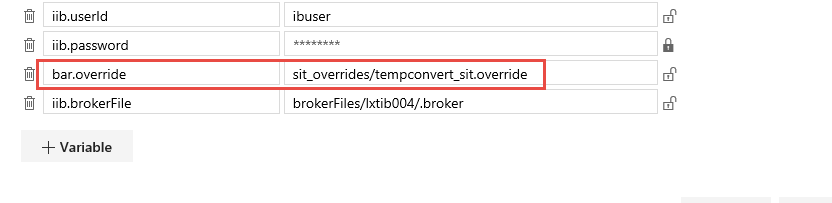
### Override file

The **bar.override** variable must be set to the path of the override file that provides variable override values. You can find this by looking in the Git repo.

Override file in Git repo…



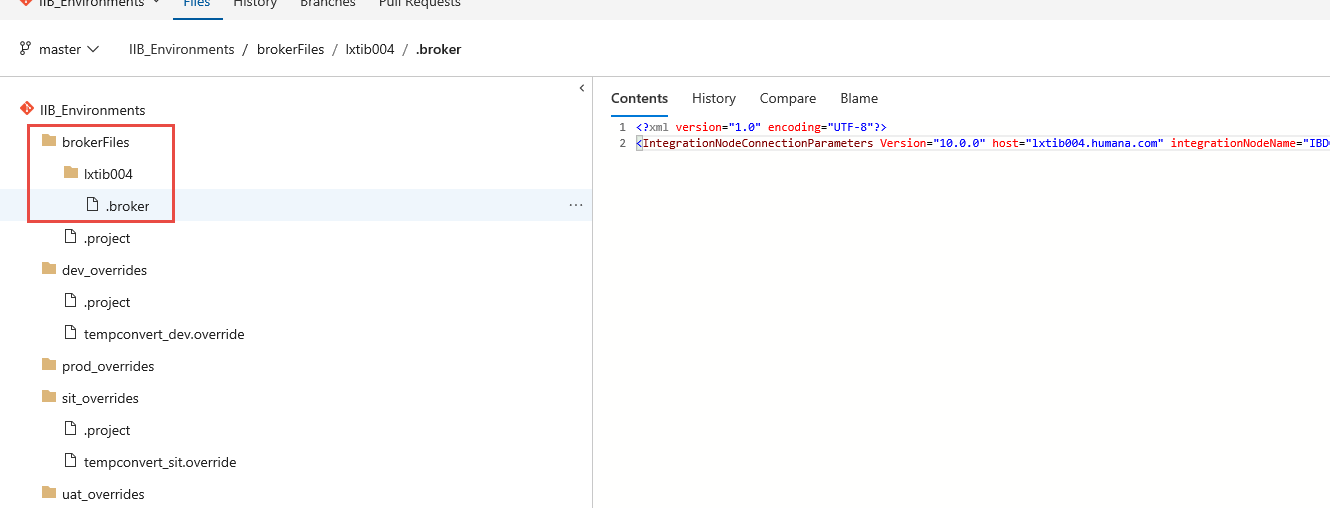
Variable (environment) with value set for this override file…



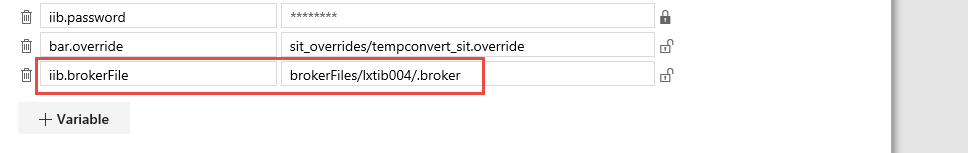
### .broker file

The **iib.brokerFile** variable must be set to the path of the .broker file that provides connection details for the broker you wish to deploy to. You can find this by looking in the Git repo.

.broker file in Git repo…



Variable (environment) with value set for this .broker file…



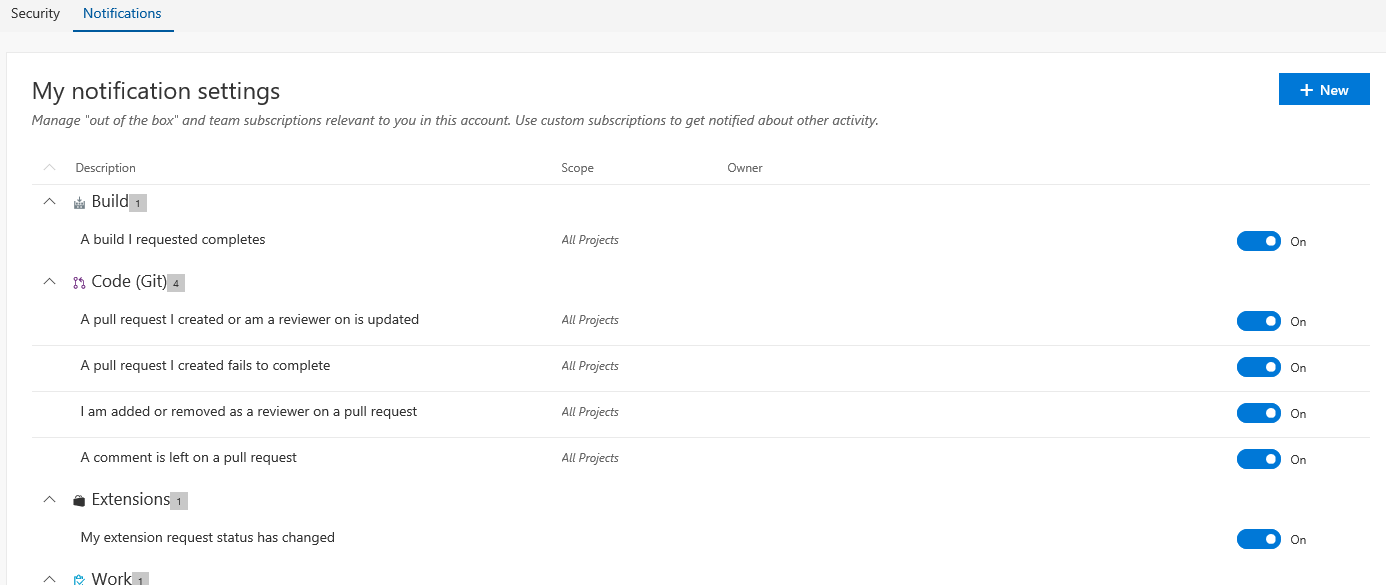
## Configuring Your Notifications

It is possible to configure the notifications you wish to receive from TFS.

1. Click on your user icon > **Notification settings**.



2. Switch notifications on or off or add a **New** notification.



## Links

1. …