

AZURE DATA FACTORY WITH AZURE DEVOPS

This whitepaper covers some of the best practices around continuous integration and deployment of Azure Data Factory

Written By-

Blesson John (Data Solution Architect-Microsoft)

Issagha BA (Data Solution Architect-Microsoft)

Reviewed By-

Ye Xu (Senior Program Manager-ADF)

Gaurav Malhotra (Principal Program Manager)

Data Factory
Continuous
Integration/Continuous
Deployment using
Azure DevOps

Contents

Azure data Factory – CI/CD	2
Environments	2
Azure DevOps.....	2
Set up the code repository for your Data Factory	3
Release Management and continuous deployment.....	8
Deploy to UAT and specify the dependencies between environment as well as the constraints.....	13
Useful links	15

Azure data Factory – CI/CD

Azure Data Factory integrates with both GitHub and Azure DevOps Git to enable source control, release management and CI/CD. With source control, developers can collaborate, track and save their changes to a branch of choice (in this case, it will be DEV branch). These changes will be merged into the main branch and deployed to the higher-level environments (QA, UAT, Prod), where it will also be tested and validated. This session of the white paper will describe how to implement continuous integration and continuous deployment for Azure Data Factory.

Note that we recommend using Git as version control system. Git is a distributed VCS. This means that each developer must create a branch to save a copy of work they do. Developers can create new branches, implement new features or fix bugs and commit their changes on their local branch. To share their work with the rest of the team, they must push their changes to the main branch.

To limit the risk of conflicts, we recommend to merge changes often. More details on how to push changes available [here](#).

Environments

The number of environments varies depending on the project constraints and every organization will have different requirements. We typically recommend four environments when implementing ADF projects: DEV, QA/INT, UAT and Production. While DEV and QA can be in the same resource group, we recommend having UAT and Production in separate resource groups. This will limit the risk of someone accidentally dropping the data factories in UAT/Production.

Note that before deploying ADF to an environment, you will need to **create the target data factory** in the resource group. For example, before deploying a Data Factory, Accounting_ADF to UAT, you need to make sure that there is a Data Factory named Accounting_ADF exists in the UAT resource group. The name can be different and parameterized.

Azure DevOps

Azure Data Factory integrates with both **GitHub** and **Azure DevOps Git**. We'll be working with Azure DevOps for this session. The first Step is to create a project in azure DevOps (dev.azure.com). Go through the [link](#) to decide whether you should create separate organization for different departments or projects or have one organization for all projects.

Select your organization and click on “create project” and provide the project details: name, visibility (private) and select the source control (git)


 + Create project


Create new project ✕

Project name *
ADF_Best_Practices ✓

Description

Visibility


Public
Anyone on the internet can view the project. Certain features like TPVC are not supported.


Private
Only people you give access to will be able to view this project.

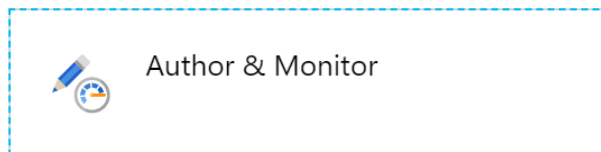
^ Advanced


Version control ⓘ
Git

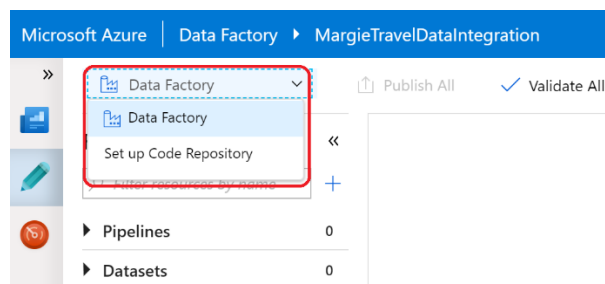
Work item process ⓘ
Basic

Set up the code repository for your Data Factory

In this section, we assume you've created your data factory. For more details on how to create a data Factory, refer to the following [link](#). You can setup the code repository while creating the data factory by checking the enable Git box or after the data factory is created. Once the Data factory has been created, click on the Author & Monitor tab



Once on this page, click on the icon  on the left pane. To add an existing data factory to a code repository, select "Set up Code Repository" in the Dropdown list



In the next window, enter the repository details referencing the project you created in the previous section

Repository Settings

×

Enter Git repository information to be associated with your Data Factory: MargieTravelDataIntegration

Repository Type *

Azure DevOps Git

Select a different tenant

☐

Azure DevOps Account *

funwithadf

Project Name *

ADF_Best_Practices

Git repository name *

☐ Create new ☒ Use Existing

ADF_Best_Practices

Collaboration branch *

master

Root folder *

/

☒ Import existing Data Factory resources to repository

Branch to import resources into *

☒ Use Collaboration ☐ Create new ☐ Use Existing

master

Cancel

Save

After you save the repository details, your data factory is under source control. You will be prompted to select a working branch. Chose “*Create new*” and create a DEV branch.

Select working branch

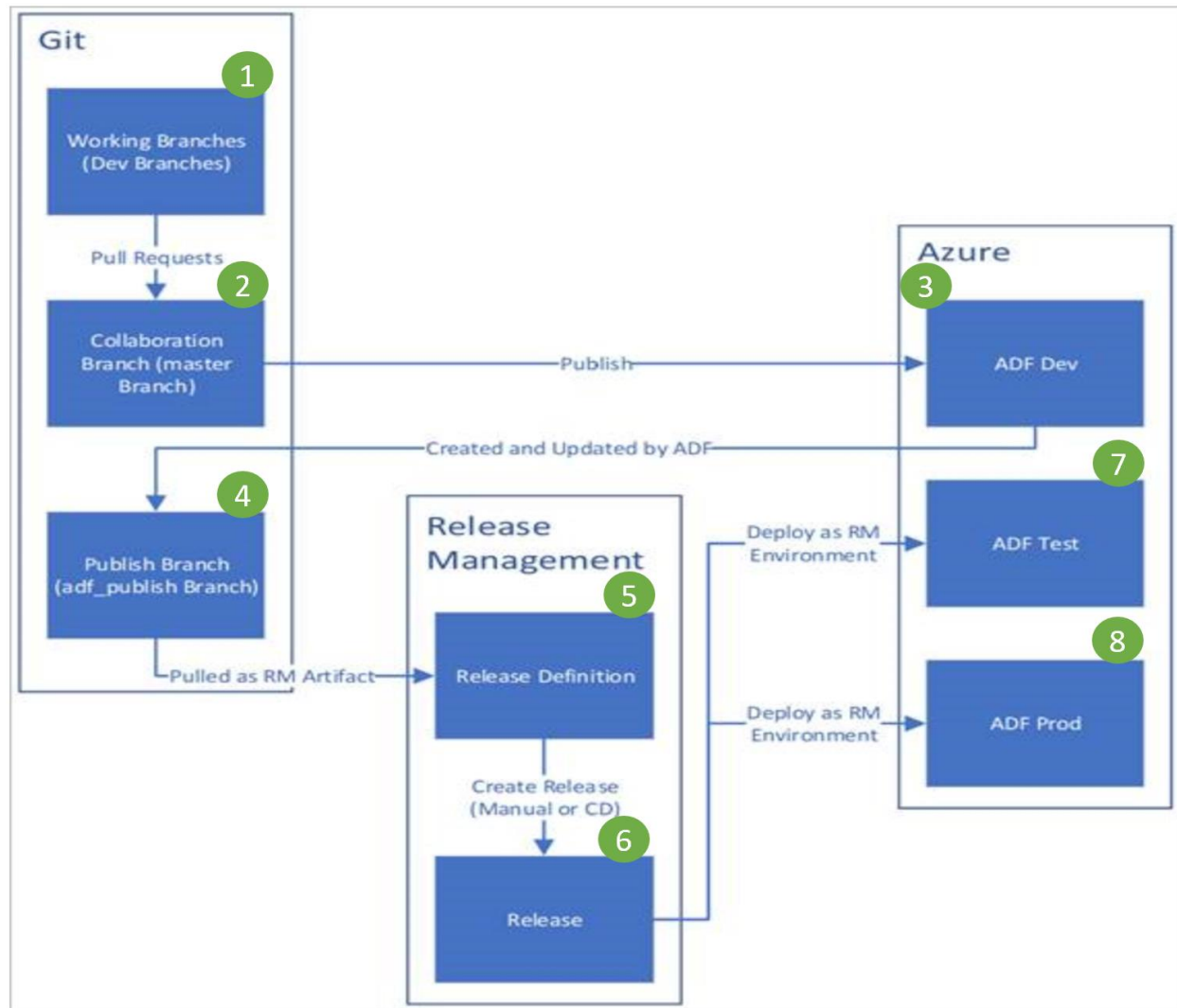
Working branch ☒ Create new ☐ Use Existing

DEV

Each developer must create feature branch to create, update and save their pipelines and activities. Once development is complete for a specific feature, the developer will save his/her changes and submit a pull request from master branch to merge the changes into the master branch. While submitting the pull request, the developer must provide title and detailed description of the changes. The developer can link the pull request to the associated work items.

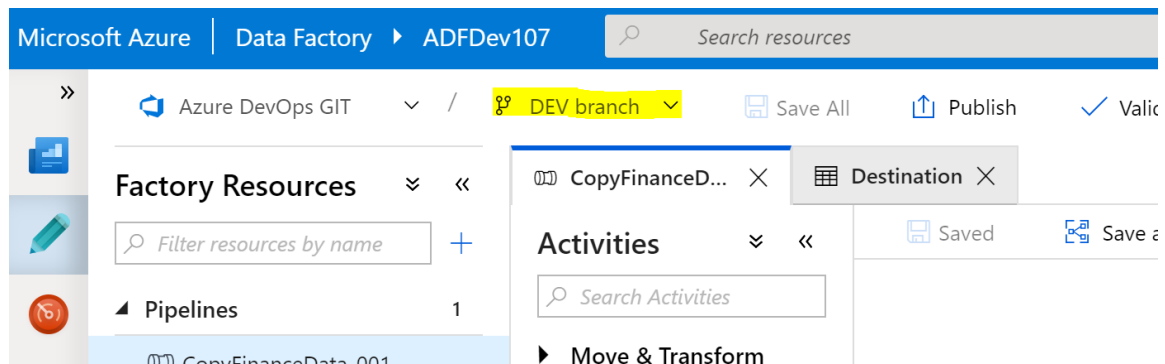
The reviewer, usually the Dev lead, will review all changes and can approve or reject the request. The reviewer can see all the files that have been changed as part of this release/sprint by doing a file comparison before deciding whether to approve or reject.

If the pull request is approved and the changes merged into the main/master branch, the dev lead can publish the main branch. This action will trigger create or update of the adf_publish branch. At this point, the dev team create a release or automatically deploy an existing release to the higher-level environments. Let us go through these steps with the help of this image-



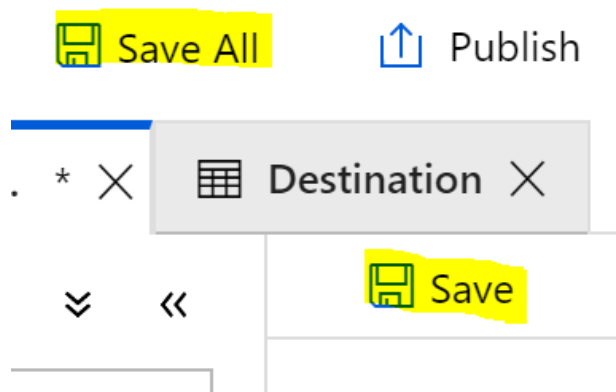
1

Each developer creates a feature branch (generically named DEV in this example) as the working branch. This is the branch that is linked to feature you are developing for Azure data factory.

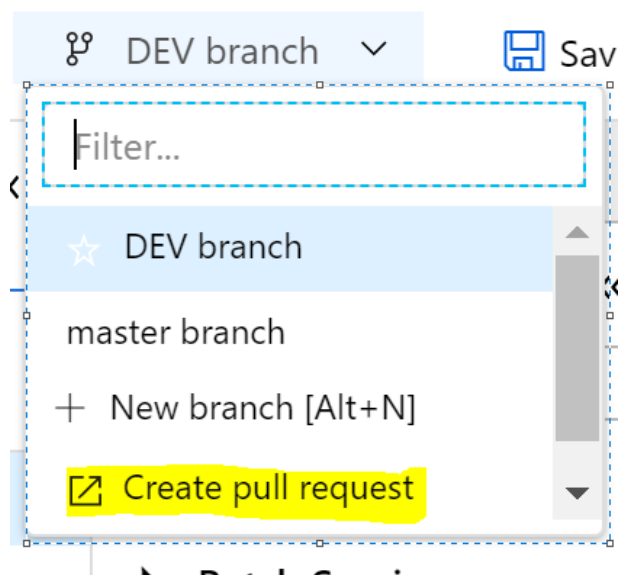


2

Once all the changes have been saved on the dev branch, this can be done by clicking the save all/save button, you will have to make a pull request.



The pull request is made as follows



This will open a new tab, make sure that the pull request is from the dev branch to the master branch.

New Pull Request

DEV into master

Title *

Updating pipeline: CopyFinanceData_001

Add label

Description

Updating pipeline: CopyFinanceData_001

Markdown supported

Updating pipeline: CopyFinanceData_001

Reviewers

Search users and groups to add as reviewers

Work Items

Search work items by ID or title

Create

Click create, and it will bring up a screen to approve and complete the pull request.

Approve

Complete

...

Click Approve if you are happy with the pull request and finally click complete. You can delete the DEV branch given that the feature development is complete.

Complete pull request

Merge commit comment

Merged PR 3: Updating pipeline: pipeline1

Updating pipeline: pipeline1

Merge type

Merge (no fast-forward)

Post-completion options

☐ Complete linked work items after merging

☒ Delete dev after merging

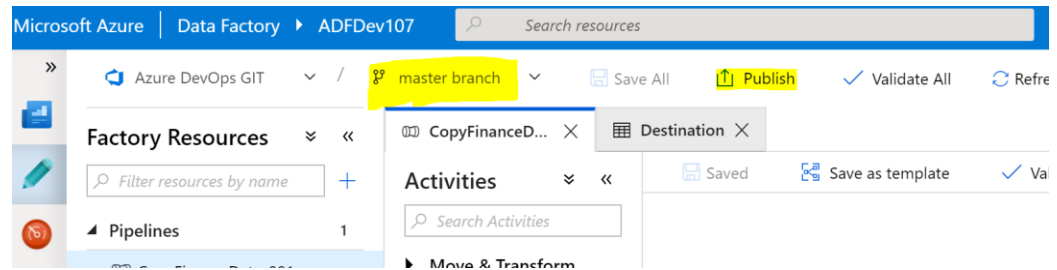
Complete merge

Cancel

3

Once all the above steps are complete, the ADF can be published from the master branch into the development ADF.

4

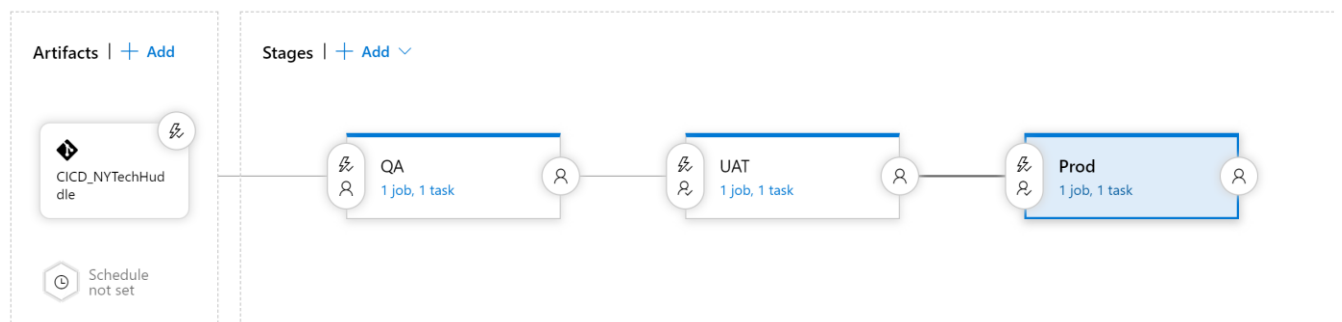


This will update the Dev ADF and publish changes to the **adf_publish** branch.

Release Management and continuous deployment

We recommend one repository for the entire project. Every environment (QA, UAT, PROD) will be defined as a stage. We'll define the dependencies (pre-deployment) between deployments to make sure all environments are deployed at the same time. For example, a new release deployed in UAT must not only be successfully deployed in QA, but also pass all the tests. To ensure UAT is deployed after a successful QA deployment, we'll set up a pre-deployment approval. You can set up the time out for the approval to the estimated QA validation time.

Below is a sample of what will try to build



The first step is to create a release pipeline. To do this, go to the pipeline section, click on release and then "New Pipeline". Select an empty job template.

Select a template

Or start with an **Empty job**

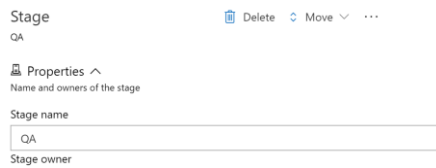
Featured



Azure App Service deployment

Deploy your application to Azure App Service. Choose from Web App on Windows, Linux, containers, Function Apps, or WebJobs.

Provide the stage name, example QA for a deployment to the QA environment and save.



Stage

QA

Delete Move ...

Properties ^

Name and owners of the stage

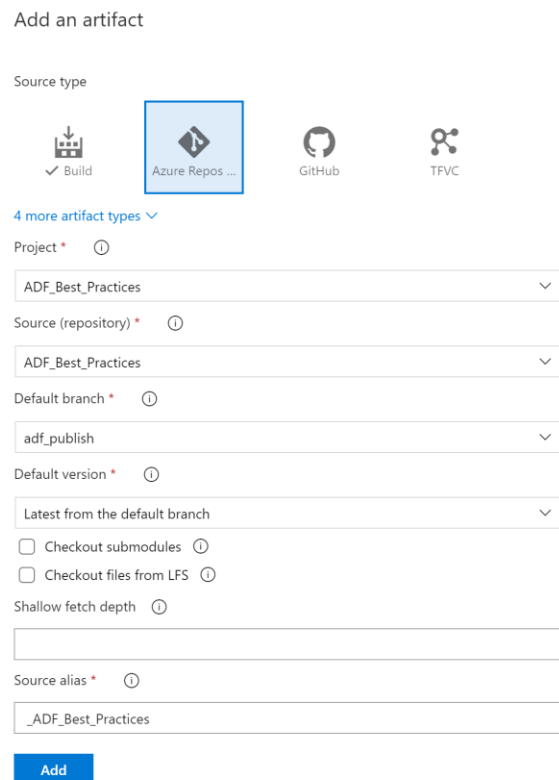
Stage name

QA

Stage owner

After the creation of the stage, we need to add the artifacts. In the artifact section, click add artifact and select Azure repository Git as source type. Select the project name from the dropdown list and choose the source repository.

In the default branch, `adf_publish`. Note that the `adf_publish` branch will not appear in the dropdown list until after you deploy the collaboration branch in the DEV data Factory at least once. Select the default version of the branch you want to deploy. Often, it's the latest version. Click add



Add an artifact

Source type

✓ Build Azure Repos ... GitHub TFVC

4 more artifact types v

Project * ⓘ

ADF_Best_Practices v

Source (repository) * ⓘ

ADF_Best_Practices v

Default branch * ⓘ

adf_publish v

Default version * ⓘ

Latest from the default branch v

☐ Checkout submodules ⓘ

☐ Checkout files from LFS ⓘ

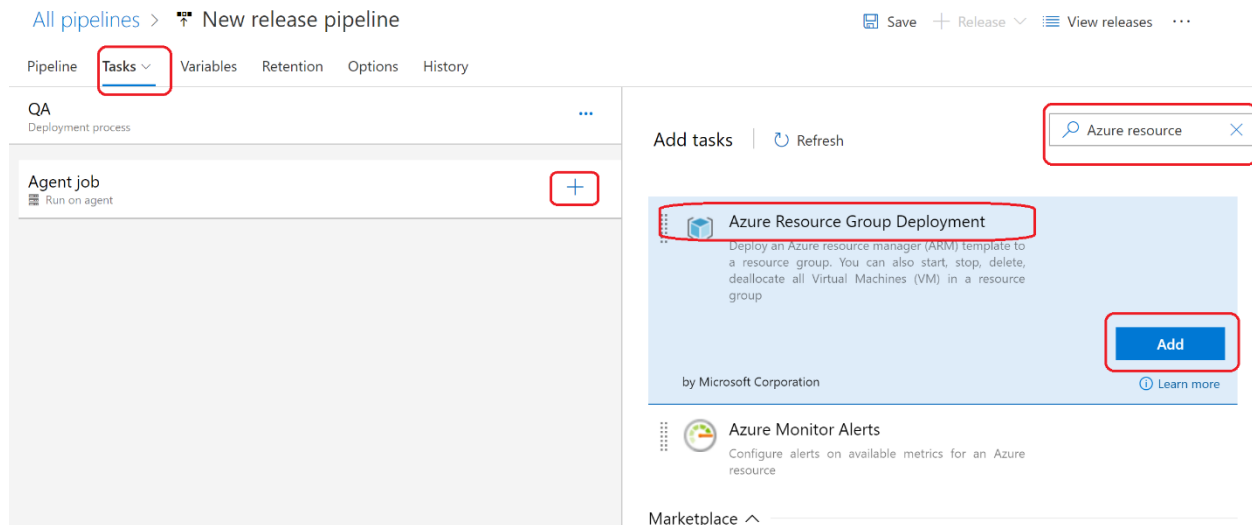
Shallow fetch depth ⓘ

Source alias * ⓘ

_ADF_Best_Practices

Add

Now, you need to add an Azure Resource Group Deployment task. In the task menu, select agent job and click on the + sign. In the search box, enter “Azure resource” and select Azure Resource Group Deployment. Click Add.



Select the added task and enter the different parameters. Note that after you enter the subscription id, you will need to authenticate (click authorize) to access the resource group and other resources in the subscription.

Display name *

Azure Deployment:Create Or Update Resource Group action on

Azure Details ^

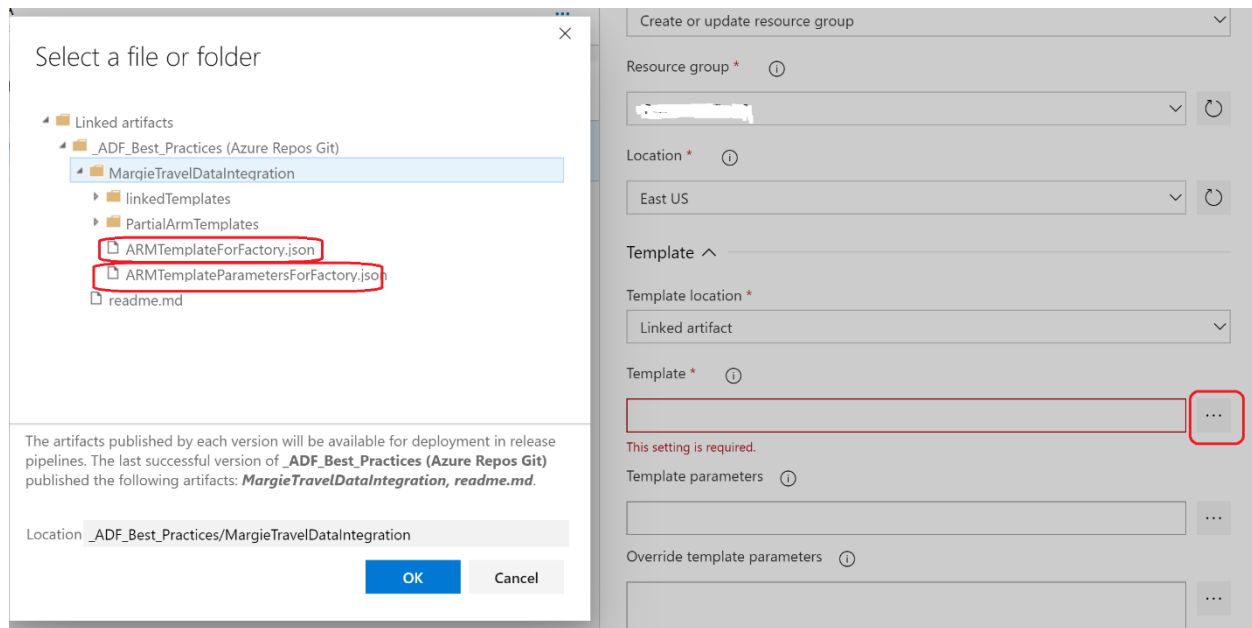
Azure subscription * ⓘ | [Manage](#)

ⓘ Click Authorize to configure an Azure service connection

Action * ⓘ

In the template section, you will need to specify the ARM template for the pipeline and the ARM template for the parameter file.

Note that you can chose “Linked artifacts” and browse the linked artifact folder structure to the ARM template files.



Alternatively, you can generate these templates manually and store them in blob or another storage. If that's your choice, select "URL to the file" in the template location section and enter the url of the ARM template files in the template link text box.

Create or update resource group

Resource group *

Location *

East US

Template ^

Template location *

URL of the file

Template link *

https://blob.core.windows.net/adfcicd/arm_template.json

Template parameters link

https://blob.core.windows.net/adfcicd/arm_template_parameters.json

The override template parameter section allows to define the parameters in the target environment. We recommend having one parameter template file per environment.

While you can override the parameter values manually in each environment deployment stage, we also recommend to use KeyVault secrets to avoid having sensitive information (passwords, connection strings,...) in plain text. Go to the following link for more details on how to [set and retrieve a secret from Azure Key Vault using the Azure portal](#).

Click save after you enter all the required parameter values.

Action * ⓘ

Create or update resource group

Resource group * ⓘ

IB_ADF_CICD_RG

Location * ⓘ

East US

Template ^

Template location *

Linked artifact

Template * ⓘ

\$(System.DefaultWorkingDirectory)/_ADF_Best_Practices/MargieTravelDataIntegration/ARMTemplateForFactory.json

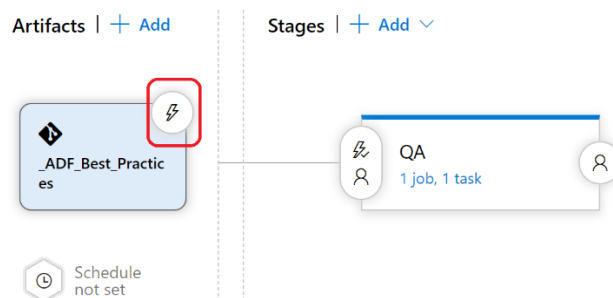
Template parameters ⓘ

\$(System.DefaultWorkingDirectory)/_ADF_Best_Practices/MargieTravelDataIntegration/ARMTemplateParametersForFactory.json

Override template parameters ⓘ

-factoryName "MargieTravelDataIntegration" -
LS_AzureBlobStorage_properties_typeProperties_connectionString_secretName
"ModernDWStorageAccount" -

In the next section we'll define what triggers a continuous deployment. Select the continuous deployment trigger as indicated in the figure below.



Enable the “Continuous deployment trigger. In the example below, we selected `adf_publish` as filter branch. This mean, a deployment will be triggered whenever there is commit on the `adf_publish` branch.

Continuous deployment trigger

Git: `_ADF_Best_Practices`



Creates a release every time a Git push occurs in the selected repository.

Branch filters ⓘ

Type

Branch



+ Add

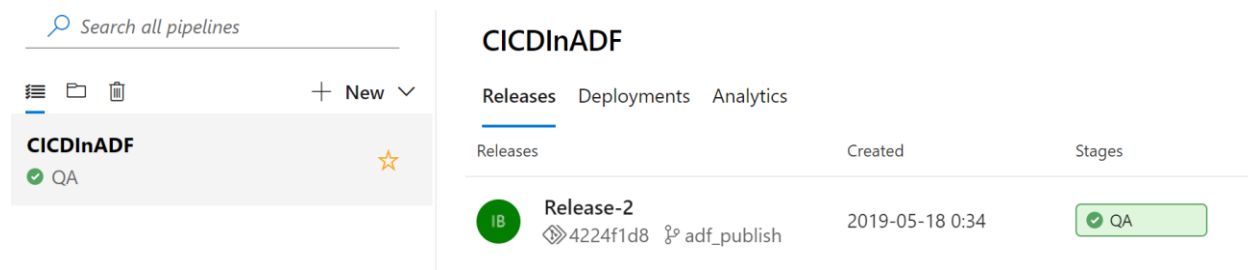
Pull request trigger

Git: `_ADF_Best_Practices`

☐ Disabled

ⓘ Enabling this will create a release every time a selected artifact is available as part of a pull request workflow

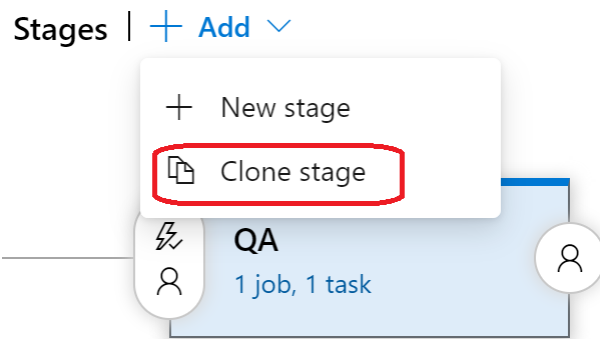
At this point, you can make some changes to you Data factory in the DEV branch and create a pull request. Once the reviewer approves the pull request and completes the merge in the collaboration branch, you can publish the change to the Dev environment in Azure Data Factory. This will automatically trigger a deployment to your QA environment.

The screenshot shows the Azure Data Factory interface. On the left, a sidebar displays a search bar and a list of pipelines, with "CICDInADF" selected and marked with a star. The main area shows the "CICDInADF" pipeline details, with tabs for "Releases", "Deployments", and "Analytics". The "Releases" tab is active, showing a table with columns "Releases", "Created", and "Stages". A single release is listed: "Release-2" with a commit hash "4224f1d8", branch "adf_publish", and a creation time of "2019-05-18 0:34". The "Stages" column shows a green box with a checkmark and the label "QA".

Releases	Created	Stages
Release-2 4224f1d8 adf_publish	2019-05-18 0:34	✓ QA

Deploy to UAT and specify the dependencies between environment as well as the constraints

You can create a new stage for UAT from scratch or clone the QA stage and make the necessary changes. To clone the QA stage, select it and, in the stages area, click on the *add* dropdown list and chose *clone stage*



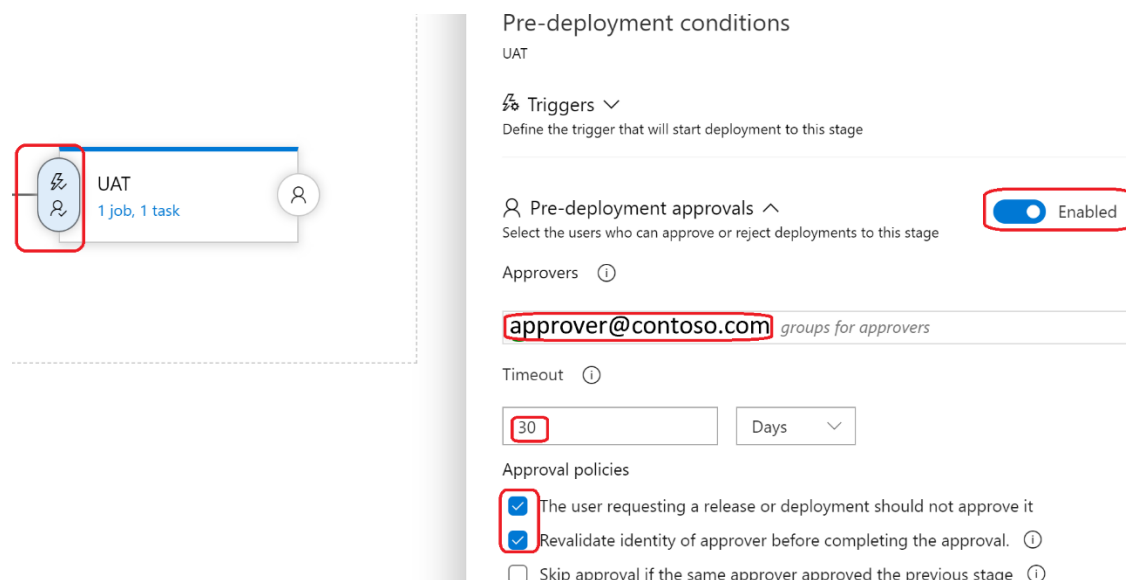
Rename the cloned stage to UAT and save.

The next step is to update the task with UAT values. Also, because the UAT deployment is triggered right after a successful deployment in QA, we'll add an approval condition. This will allow us to wait until QA passes all the validation tests before we deploy to UAT.

In the Task dropdown list, select UAT and update all the relevant parameters to the UAT environment (resource group, templates...) and save



select the pre-deployment conditions and enable pre-deployment approvals. Enter the name/email of the approver.

In the timeout textbox, you can put the estimated duration of the QA validation phase. Optionally, you can put additional constrains like to avoid the user who submitted the deployment request to also approve it. Save the changes.



Repeat the UAT steps to create a (pre) production environment.

Useful links

-  [Implementing DevOps Development Processes](#)
-  [Define your multi-stage continuous deployment \(CD\) pipeline](#)