



CI-CD en Azure Data Factory

Por Santos Nicanor Oliva Escobar

Santa Rosa 320 Piso 5 A

Teléfono +54(351)5693075

Email: info@piconsulting.com.ar

www.piconsulting.com.ar



Gold Data Analytics
Gold Data Platform





Que es CI-CD

Se utilizan las siglas CI-CD para hacer referencia a Integración Continua, Entrega Continua.

- La integración continua es el procedimiento de probar cada cambio realizado en el código base automáticamente y tan pronto como sea posible.
- La entrega continua sigue las pruebas realizadas durante la integración continua y envía los cambios a un sistema de ensayo o producción.

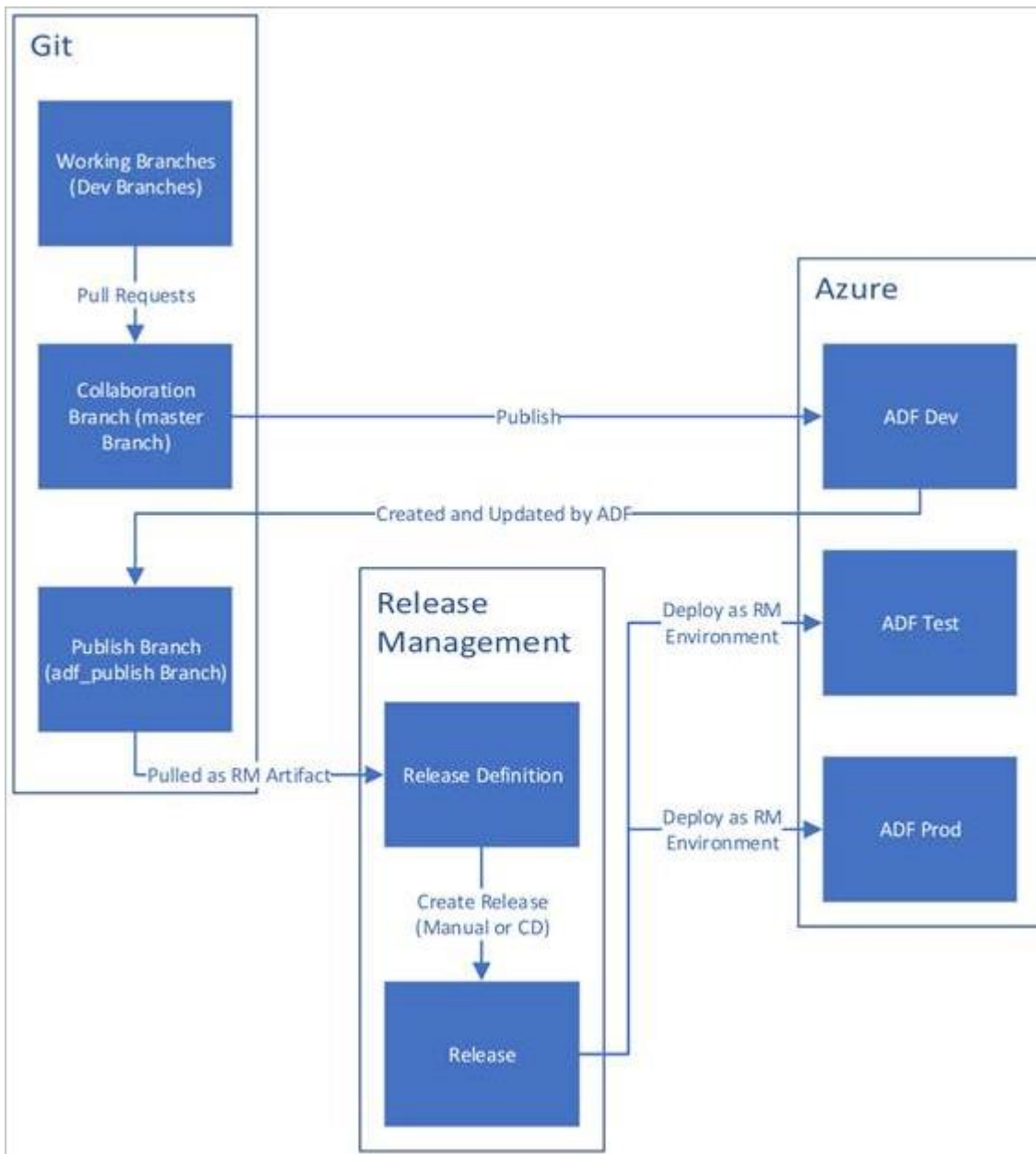
En Azure Data Factory, la integración y la entrega continuas (CI/CD) implican el traslado de pipelines de Data Factory de un entorno (desarrollo, prueba o producción) a otro.

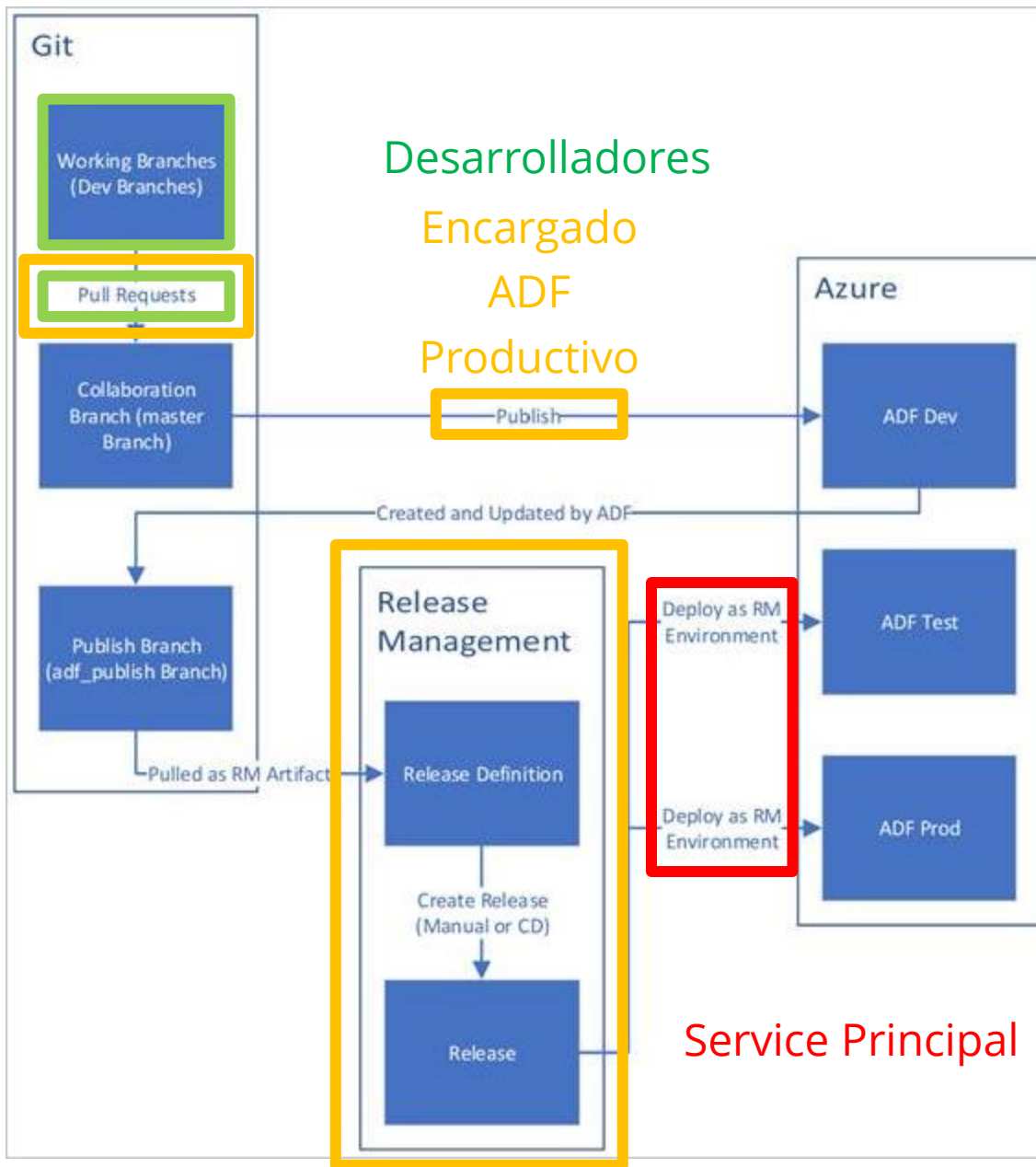
Azure Data Factory usa las plantillas de Azure Resource Manager para almacenar la configuración de las distintas entidades de ADF (canalizaciones, conjuntos de datos, flujos de datos, etc.).



Ciclo de vida

Para trabajar integración continua en Azure DataFactory primero es necesario entender el ciclo de vida







Acá podemos observar:

- 3 ramas de GIT
 - o Dev branches
 - o Master branch
 - o Adf-publish branch
- 3 Recursos ADF
 - o Dev
 - o Test
 - o Prod
- La herramienta de gestión de Releases
 - o Los cuales se pueden gestionar manual o automáticamente

Antes de explicar el paso a paso es importante comprender la lógica con la cual se trabajará:

- En el recurso **ADF Dev** vamos a trabajar con Git, trabajando sobre distintas ramas,
 - o La rama **Master branch** será la única que podrá publicar en nuestro recurso ADF y no se podrán hacer cambios en ella sino a través de pull request.
 - o Para realizar cambios se puede trabajar creando distintas ramas de master e ir eliminándolas a medida que se realiza merge o realizar todos los cambios en una rama **Dev branch**.
 - o **Adf-publish branch** es una rama creada automáticamente que guarda todo lo publicado en nuestro recurso ADF-dev.
- Una vez realizado el Publish en ADF-Dev, Devops se encargará de hacer un deploy de lo publicado en los entornos ADF-Test y ADF-Prod.
- Solo el Service Principal definido podrá hacer modificaciones en el ADF productivo, a través de los sucesivos deploys



Santa Rosa 320 Piso 5 A

Teléfono +54(351)5693075

Email: info@piconsulting.com.ar

www.piconsulting.com.ar



Gold Data Analytics
Gold Data Platform





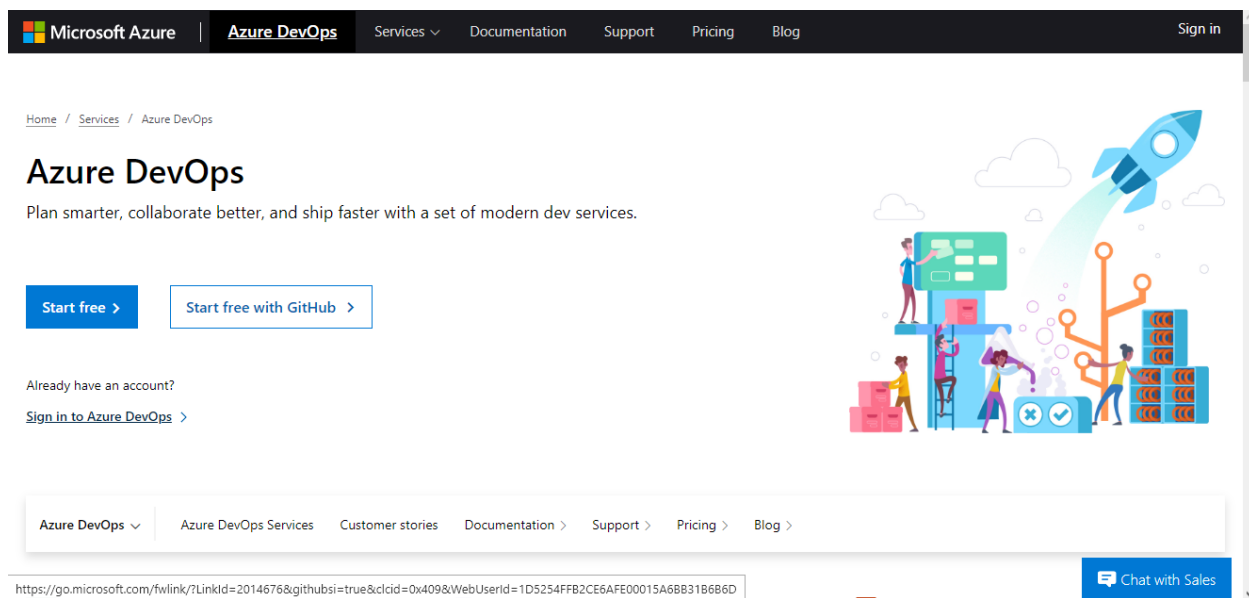
Repositorios

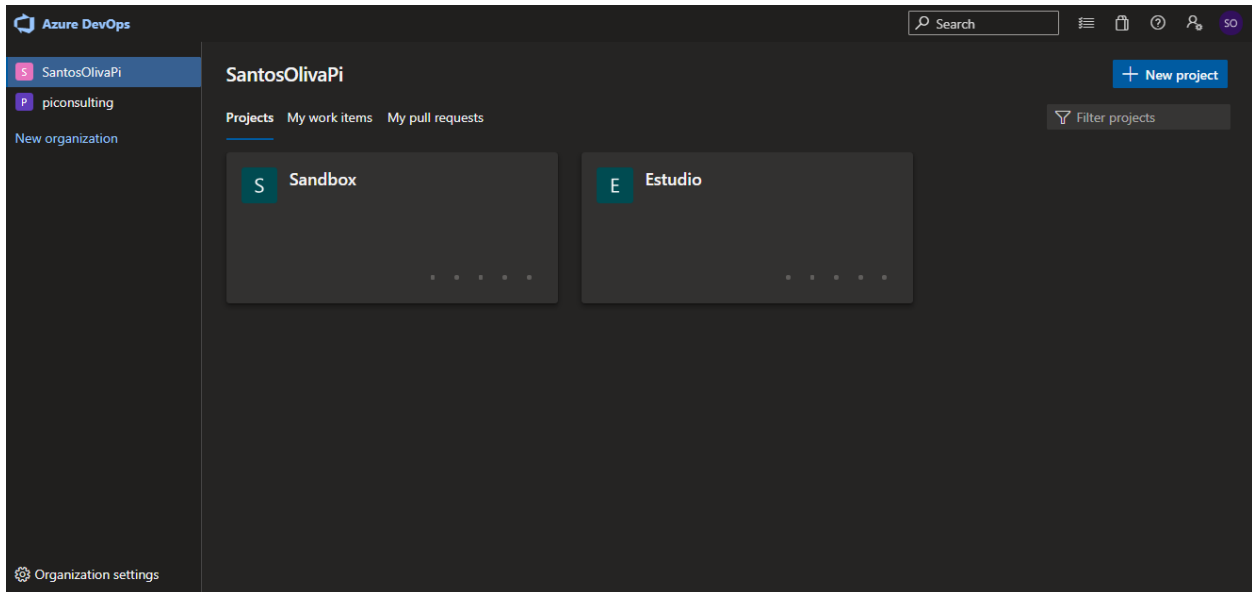
Teniendo en cuenta esto, y ya teniendo los distintos recursos ADF trabajando, crearemos un repositorio GIT, tanto para la integración continua como para control de versión.

Esto se puede trabajar utilizando un repo de GITHUB, pero teniendo en cuenta la existencia de Azure Repos GIT que permite que el código sea privado de manera gratuita, utilizaremos dicha herramienta.

Creación DevOps

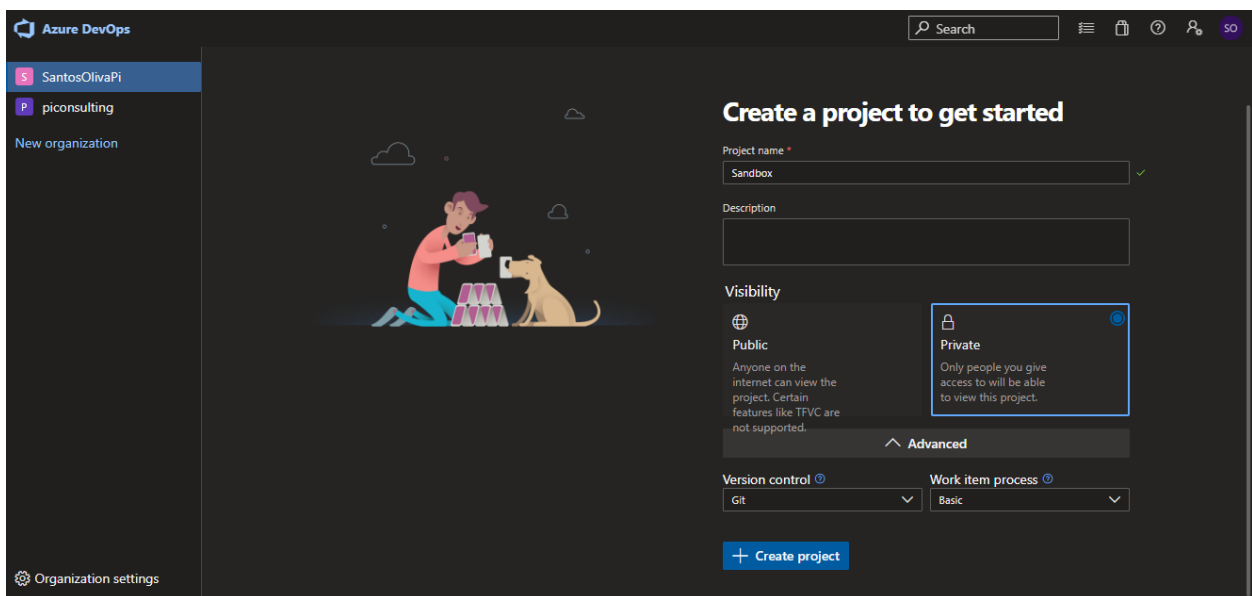
Entramos a dev.azure.com y logueamos con nuestra cuenta





A la izquierda podemos seleccionar la organización, en mi caso, cree una organización propia con motivos de práctica.

En el cuadro central se observan los proyectos, y la opción Nuevo proyecto, donde se crea un proyecto nuevo, seleccionando su visibilidad.

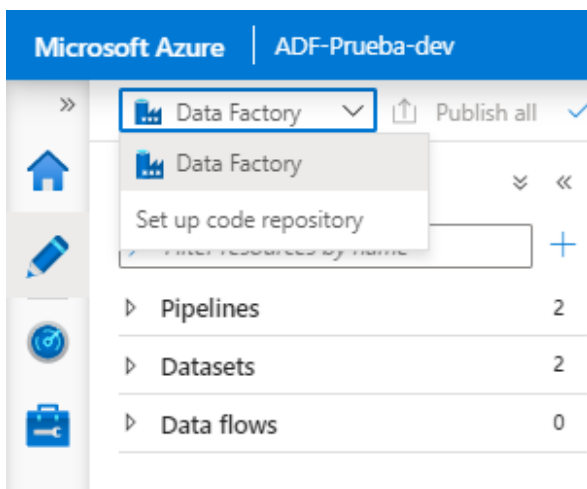




Configuración ADF

El siguiente paso se trabaja desde Azure Data Factory.

Una vez dentro de la sección de edición. Se hace click en el combo box "Data Factory" y se selecciona Set up code repository.





Aquí, configurando el repositorio, seleccionaremos como tipo Azure DevOps Git, seleccionamos el nombre del proyecto, en nuestro caso "Sandbox" y el repositorio Git.

Nosotros crearemos un repositorio desde aquí, dándole nombre y seleccionando la rama de colaboración, esta suele ser master. Esta rama será la única con posibilidad de realizar Publish en nuestro recurso ADF.

También seleccionaremos la opción de guardar lo trabajado en nuestro ADF en nuestro Git, en nuestro caso guardaremos en la rama de colaboración.

Repository settings

Repository type * ?
Azure DevOps Git ▼

Select a different directory ☐

Azure DevOps Account * ?
SantosOlivaPi ▼

Project name * ?
Sandbox ▼

Git repository name * ☒ Create new ☐ Use existing ?
ADF-Prueba

Collaboration branch * ?
master

Root folder * ?
/ADF

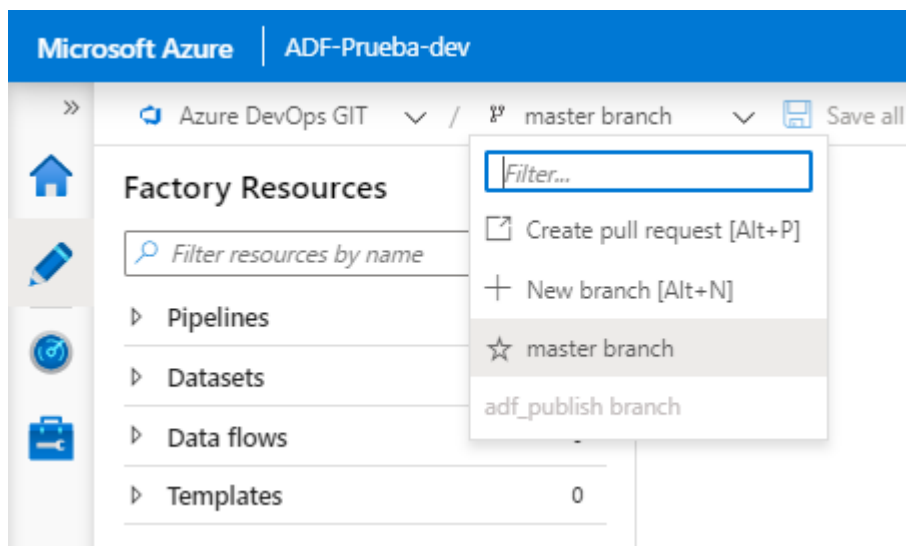
☒ Import existing Data Factory resources to repository

Branch to import resources into * ?
☒ Use Collaboration ☐ Create new ☐ Use existing

Apply Cancel



Una vez aplicadas estas configuraciones, haremos click en la rama master, en la cual estamos trabajando y seleccionaremos crear una nueva rama



Ahí creamos una rama Dev, donde realizaremos todos los cambios del ADF

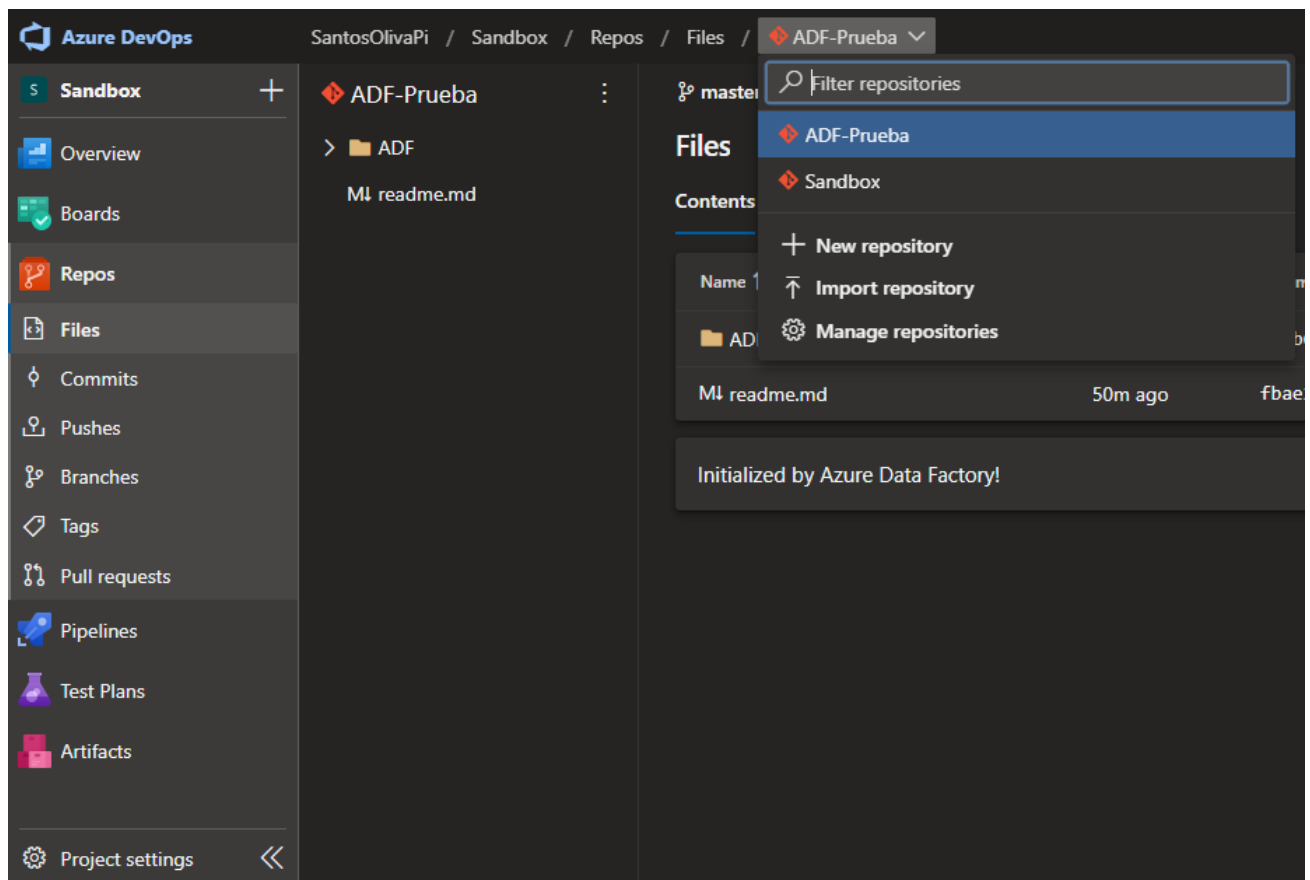


Gestión de repositorios en Azure DevOps

Volviendo al entorno Azure DevOps, posicionándonos en

Repos – Files

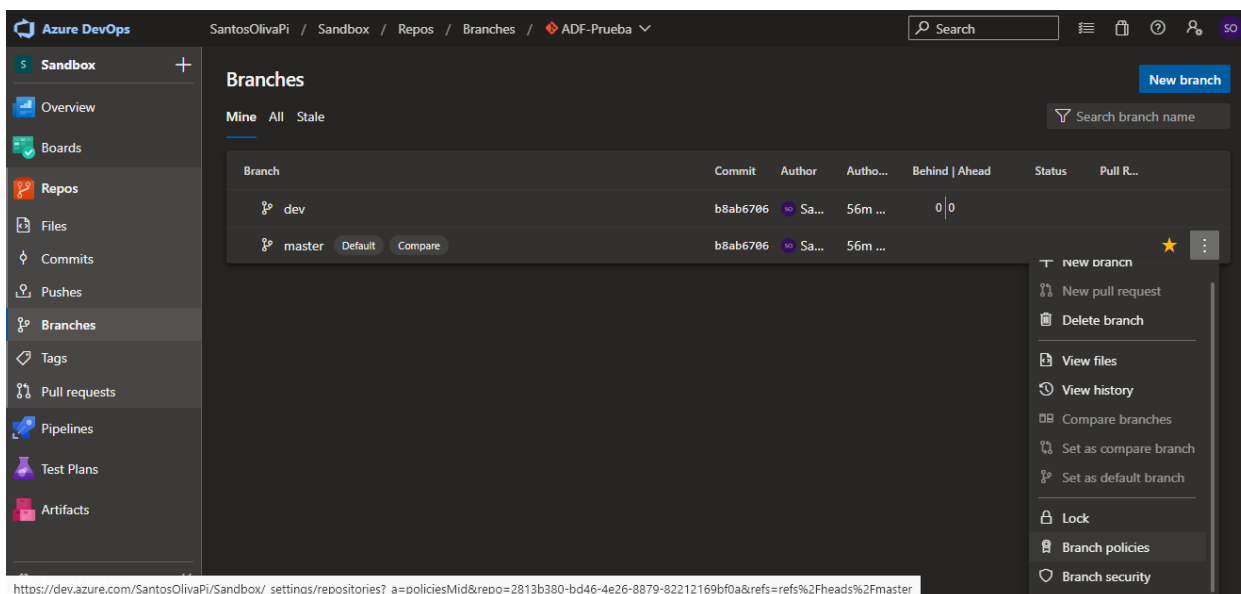
Y luego haciendo click en el lugar que se indica en la imagen veremos el path donde guardamos nuestro código, como en nuestro caso llamamos al root ADF-Prueba por el recurso utilizado, toda nuestra información se encontrara allí, en lugar que en Sandbox



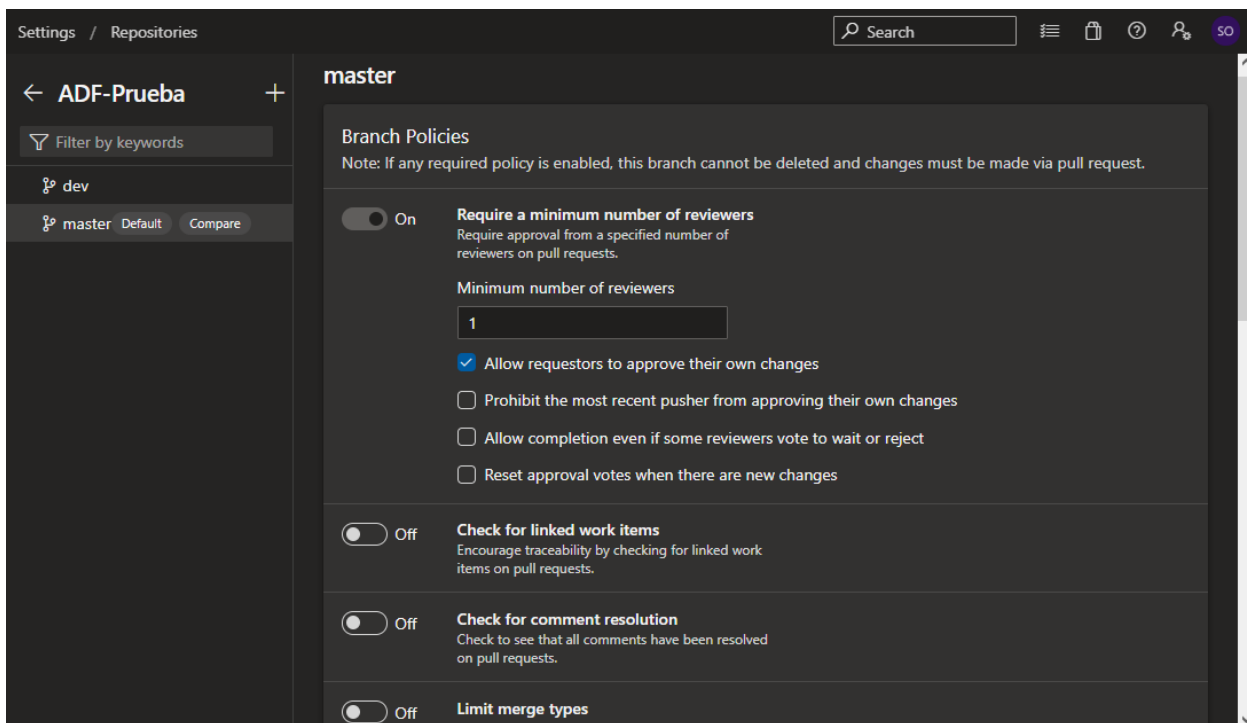


Haciendo click en la opción Branches, podremos gestionarla.

Como no deseamos que se realicen cambios directamente sobre la rama master sino por pull requests, modificaremos sus políticas haciendo click en los 3 puntos situados a la derecha del mismo y seleccionando "Branch policies"

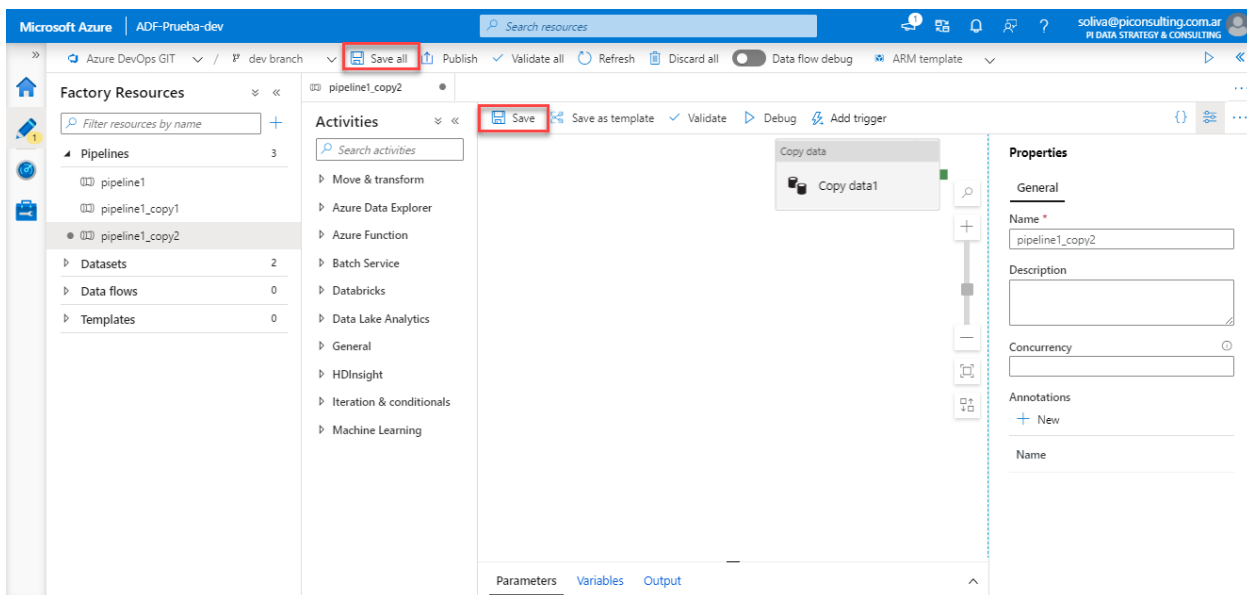


Aquí haremos las configuraciones correspondientes a las políticas deseadas, como este es un caso de ejemplo, seleccione las que se muestran a continuación.



Con esto seteado, probamos como es el flujo desde la Branch dev hasta la adf_publish

Realizando cualquier cambio en ADF, en nuestro caso clonamos un pipeline, se guarda haciendo click en SAVE o SAVE ALL opciones no disponibles previamente





Si quisiéramos publicar estos cambios, encontraríamos que no esta disponible tal opción, nos aparece esta notificación

Publish is only allowed from collaboration ('master') branch.

[Merge the changes to 'master'.](#) 

OK



Haciendo clic en Merge, podremos observar la siguiente ventana, una vez seleccionadas las opciones deseadas, por ejemplo los Reviewers necesarios y haremos click en Create

Azure DevOps SantosOlivaPi / Sandbox / Repos / Pull requests / ADF-Prueba

Sandbox +

- Overview
- Boards
- Repos
- Files
- Commits
- Pushes
- Branches
- Tags
- Pull requests**
- Pipelines
- Test Plans
- Artifacts

New pull request

dev into master

Overview Files 1 Commits 1

Title

Adding pipeline: pipeline1_copy2

Description

Adding pipeline: pipeline1_copy2

Markdown supported.

Adding pipeline: pipeline1_copy2

Reviewers Add required reviewers

Santos Oliva X +

Work items to link

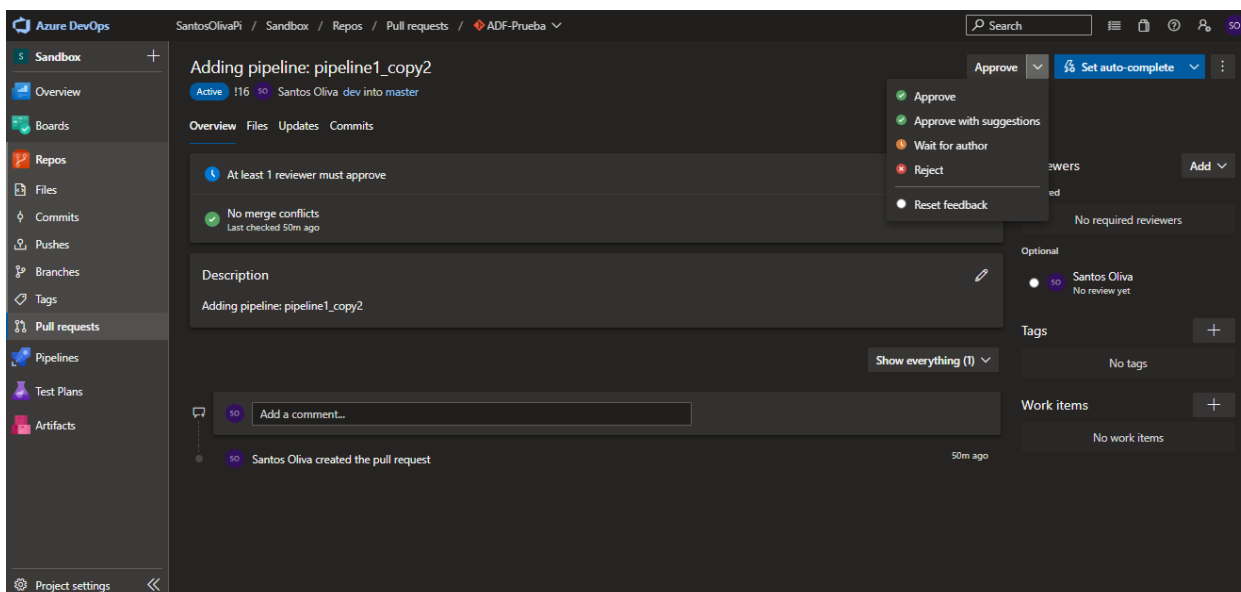
Search work items by ID or title

Tags

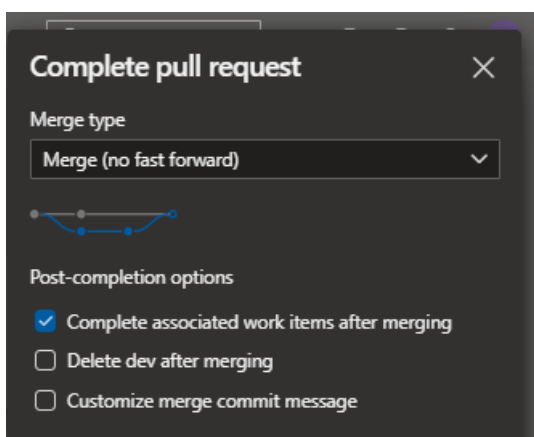
Create



Aquí quien deba revisar, dará su feedback o aprobará el cambio, y hará click en complete



Se gestionara las opciones deseadas en cuanto a las ramas de Git, y luego Completar Merge



Una vez realizado este paso, los cambios se podrán ver en la rama Master, que ahora podremos observar desde nuestro ADF, con los cambios implementados en la rama Dev.

Aquí, podremos publicar los cambios trabajados, que impactaran en la rama adf_publish, que representa lo publicado en el recurso ADF correspondiente.



1

2

?

soliva@piconsulting.com.ar
PI DATA STRATEGY & CONSULTING

Pending changes

Publish branch

adf_publish

▲ Pipelines

pipeline1_copy2	(New)	-
-----------------	-------	---

▲ Datasets

▲ Data flows

▲ Integration runtimes

▲ Linked services

▲ Triggers

OK

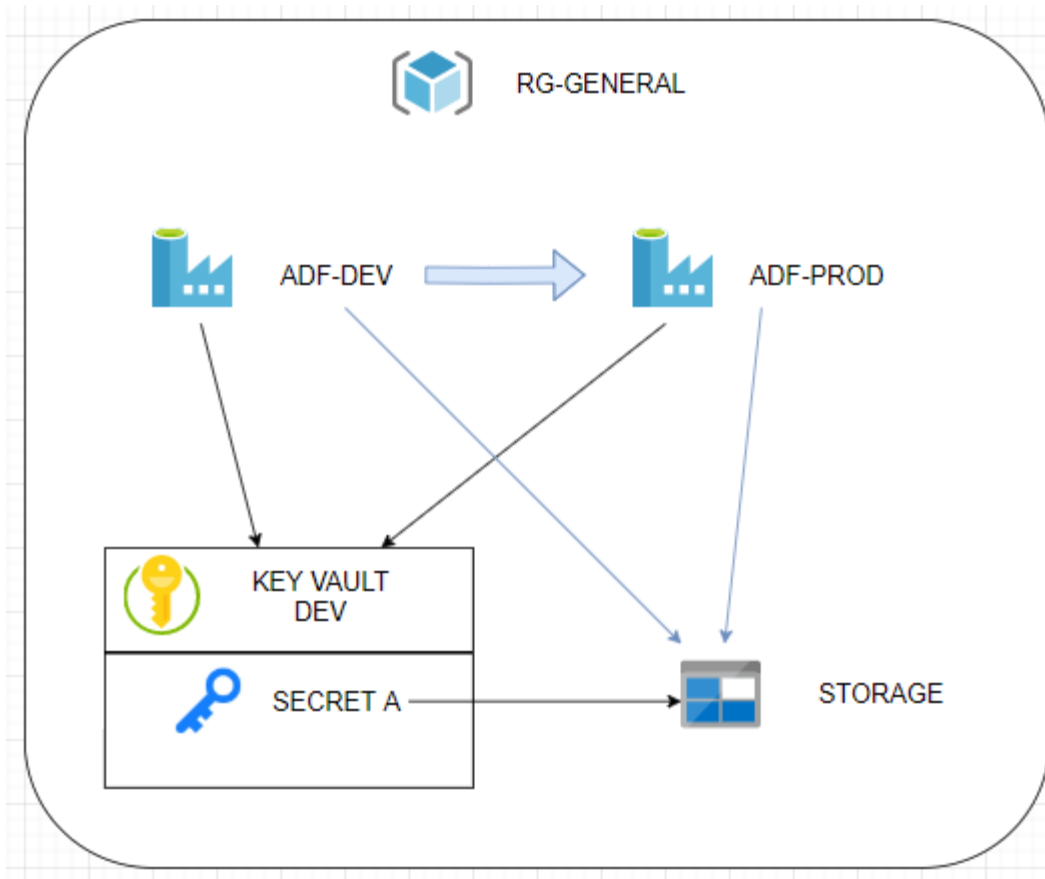
Cancel

Con esto, ya terminamos nuestro trabajo en ADF-Prueba-dev

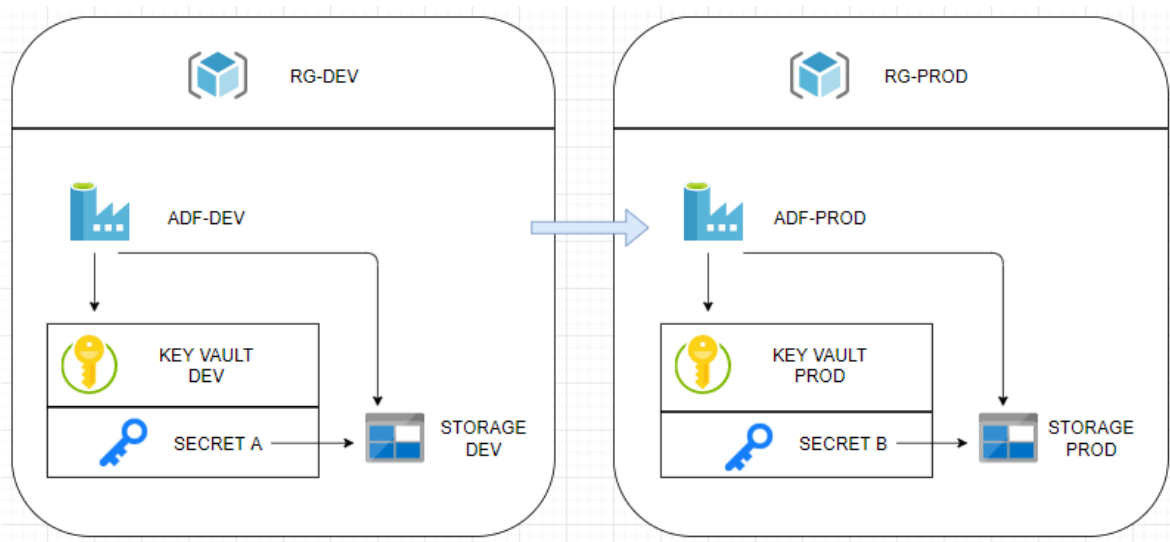


Arquitectura

La arquitectura de datos utilizada en nuestro caso fue la siguiente, por lo tanto, lo único a cambiar en Override Parameters, era el nombre del Data Factory.



A modo de ejemplo, este es otro caso que puede suceder, donde cambien cuestiones de arquitectura según el ambiente



Donde:

- Los grupos de recursos son diferentes
 - o Que se gestiona en cual se realizara en la operación de Deploy
- No solo el nombre del ADF es diferente, sino también el de los recursos con los que se conecta, en este ejemplo,
 - o el Key Vault y el nombre de su secreto
 - o el storage donde se guarda información.

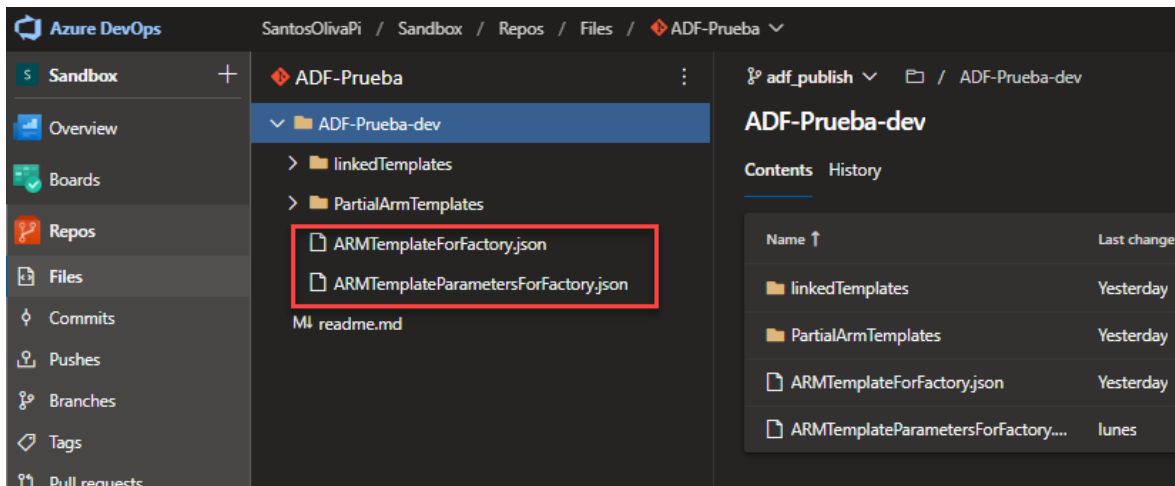
Este problema de arquitectura se estudia analizando los servicios que corresponde a cada ADF, y reemplazándolos via parametrización.



Parametrización de ARM Templates

En nuestro proyecto de DevOps, Repos-Files, y posicionándonos dentro de la rama `adf_publish` encontraremos los archivos:

- `ARMTemplateForFactory.json`
 - o Que tiene toda la información del Data Factory
- `ARMTemplateParametersForFactory.json`
 - o Que contiene parámetros utilizados en el template



Utilizaremos estos archivos para cubrir las diferencias entre ambos ADF y que se pueda realizar el paso de Dev a Prod de manera correcta.



Haciendo clic en cada uno de los archivos, tendremos acceso a su contenido, y también la posibilidad de editarlos

Acá se puede observar como se relacionan ambos archivos

```
1 {
2   "$schema": "https://schema.management.azure.com/schemas/2015-01-01/deploymentParameters.json#",
3   "contentVersion": "1.0.0.0",
4   "parameters": {
5     "FactoryName": {
6       "value": "ADF-Prueba-dev"
7     },
8     "kv_Sandbox_properties_typeProperties_baseUrl": {
9       "value": "https://kv-sandbox1.vault.azure.net/"
10    },
11    "st_sandbox_properties_typeProperties_url": {
12      "value": "https://stestudio.dfs.core.windows.net"
13    },
14    "st_sandbox_properties_typeProperties_tenant": {
15      "value": "ca1d08b0-9543-4bd9-8718-42becddc7786"
16    },
17    "st_sandbox_properties_typeProperties_servicePrincipalId": {
18      "value": "226958b0-7eba-49a4-9d49-2387b482d03a"
19    }
20  }
21 }
```

Así es como se consumen dichos parametros

```
10 "kv_Sandbox_properties_typeProperties_baseUrl": {
11   "type": "string",
12   "defaultValue": "https://kv-sandbox1.vault.azure.net/"
13 },
14 "st_sandbox_properties_typeProperties_url": {
15   "type": "string",
16   "defaultValue": "https://stestudio.dfs.core.windows.net"
17 },
18 "st_sandbox_properties_typeProperties_tenant": {
19   "type": "string",
20   "defaultValue": "ca1d08b0-9543-4bd9-8718-42becddc7786"
21 },
22 "st_sandbox_properties_typeProperties_servicePrincipalId": {
23   "type": "string",
24   "defaultValue": "226958b0-7eba-49a4-9d49-2387b482d03a"
25 }
26 },
27 "variables": {
28   "factoryId": "[concat('Microsoft.DataFactory/factories/', parameters('factoryName'))]"
29 },
30 "resources": [
31   {
32     "name": "[concat(parameters('factoryName'), '/pipeline1')]",
33     "type": "Microsoft.DataFactory/factories/pipelines",
34     "apiVersion": "2018-06-01",
```



Teniendo en cuenta esta información, parametrizaremos los datos que se modificarán cuando se pase del entorno de desarrollo al de producción (tema trabajado en Releases).

En nuestro caso de arquitectura alternativo, observando los parámetros presentes en `ARMTemplateParametersForFactory.json` nos faltaría parametrizar el “secret name”.

Para utilizar parámetros personalizados de manera correcta, se deberá trabajar desde la rama que corresponda, entrando a “Manage” y luego a “Parametrization template”. Editando este .json, se establecerán parámetros que serán tenidos en cuenta al momento de realizar el Deploy.



Plantilla ARMTemplateParametersForFactory

Sintaxis de los parámetros parametrizados

Trabajaremos la plantilla de la siguiente manera:

Se utilizará la nomenclatura <acción>:<nombre>:<tipoDato>

- Siendo las acciones
 - o “=” si se desea mantener el valor actual como valor por defecto
 - o “-” si no se desea mantener el valor actual como valor por defecto
 - o “|” como un caso especial para Connection strings o keys.
- En <nombre> se colocará el nombre que se desea utilizar como nombre del parámetro, por defecto tomara el nombre de la propiedad.
- <tipoDato> es el tipo del parametro:
 - o Por defecto será “string”, si no se completa
 - o Puede tomar los valores “string”, “bool”, “number”, “object” y “secureString”

Es importante tener en cuenta que una definición no puede ser específica de una instancia de recurso. Cualquier definición se aplica a todos los recursos de ese tipo.



Dentro de cada parte del factory se especificará las propiedades que se quieran parametrizar, utilizando * para indicar que se quieren parametrizar todas las propiedades internas en un primer nivel

Por ejemplo, aquí se quieren parametrizar las propiedades *"waitTimeInSeconds"* y *"headers"* de los pipelines, ubicadas en *properties-Activities-typeProperties*

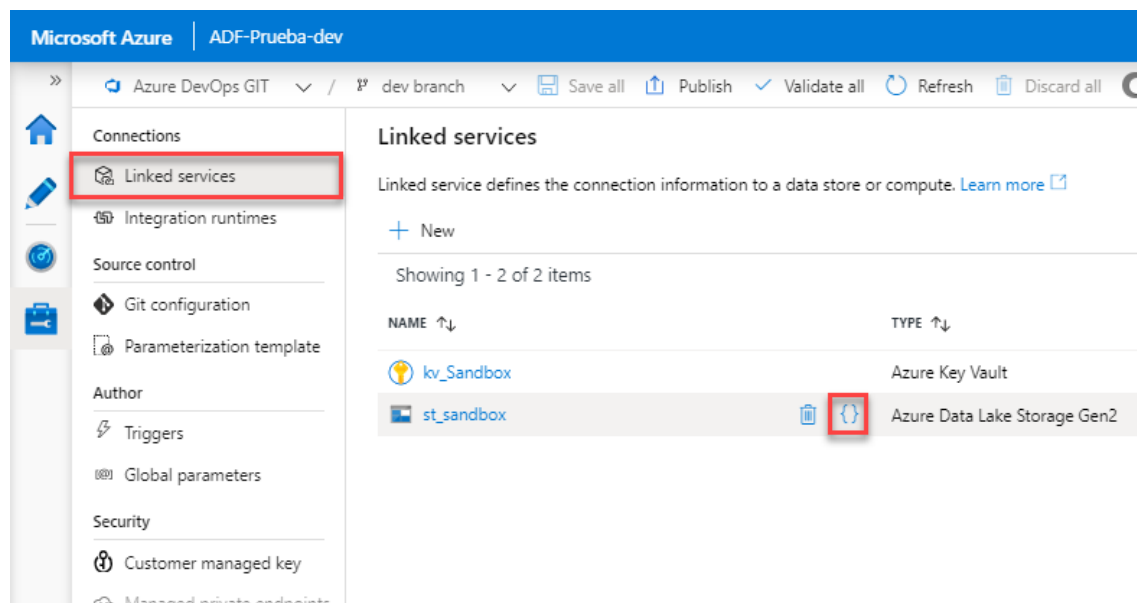
```
{
  "Microsoft.DataFactory/factories/pipelines": {
    "properties": {
      "activities": [{
        "typeProperties": {
          "waitTimeInSeconds": "-:::number",
          "headers": "=:object"
        }
      }]
    }
  },
```

Mientras que aquí, trabajaríamos todas las propiedades dentro de *"parameters"*

```
"Microsoft.DataFactory/factories/triggers": {
  "properties": {
    "pipelines": [{
      "parameters": {
        "*": "="
      }
    }]
  }
}
```



Considerando esta información, trabajaremos nuestro ADF, primero buscaremos la ubicación de lo que deseamos parametrizar, investigando su código json,

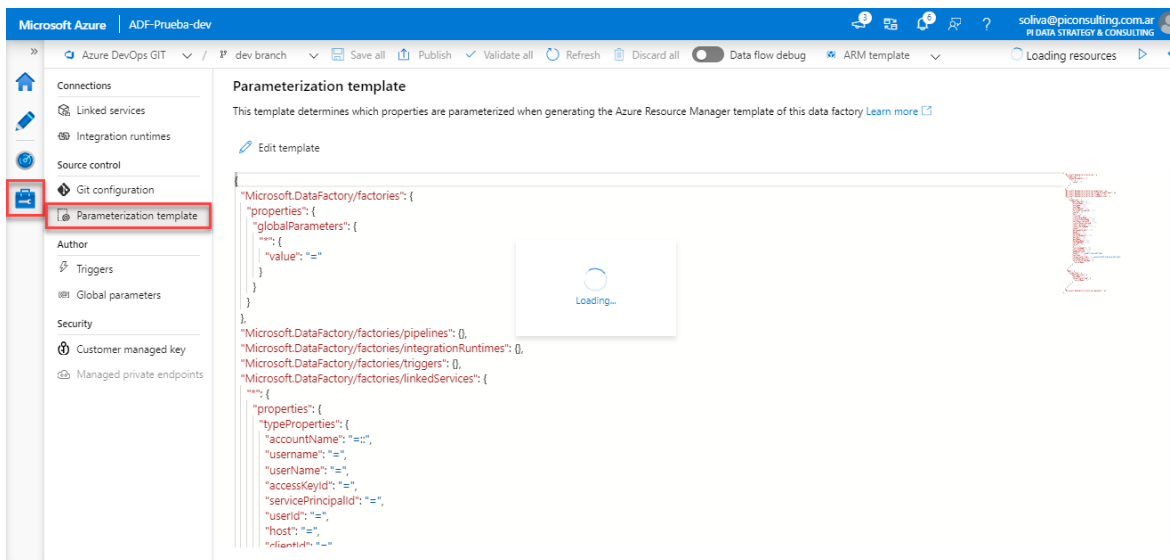


Aquí buscaremos la ubicación de lo que buscamos parametrizar, recordando el path del mismo

```
1 {
2   "name": "st_sandbox",
3   "type": "Microsoft.DataFactory/factories/linkedservices",
4   "properties": {
5     "annotations": [],
6     "type": "AzureBlobFS",
7     "typeProperties": {
8       "url": "https://stestudio.dfs.core.windows.net",
9       "tenant": "ca1d08b0-9543-4bd9-8718-42becddc7786",
10      "servicePrincipalId": "226958b0-7eba-49a4-9d49-2387b482d03a",
11      "servicePrincipalKey": {
12        "type": "AzureKeyVaultSecret",
13        "store": {
14          "referenceName": "kv_Sandbox",
15          "type": "LinkedServiceReference"
16        },
17        "secretName": "SP-sandbox-soliva"
18      }
19    },
20    "connectVia": {
21      "referenceName": "integrationRuntime2",
22      "type": "IntegrationRuntimeReference"
23    }
24  }
25 }
```

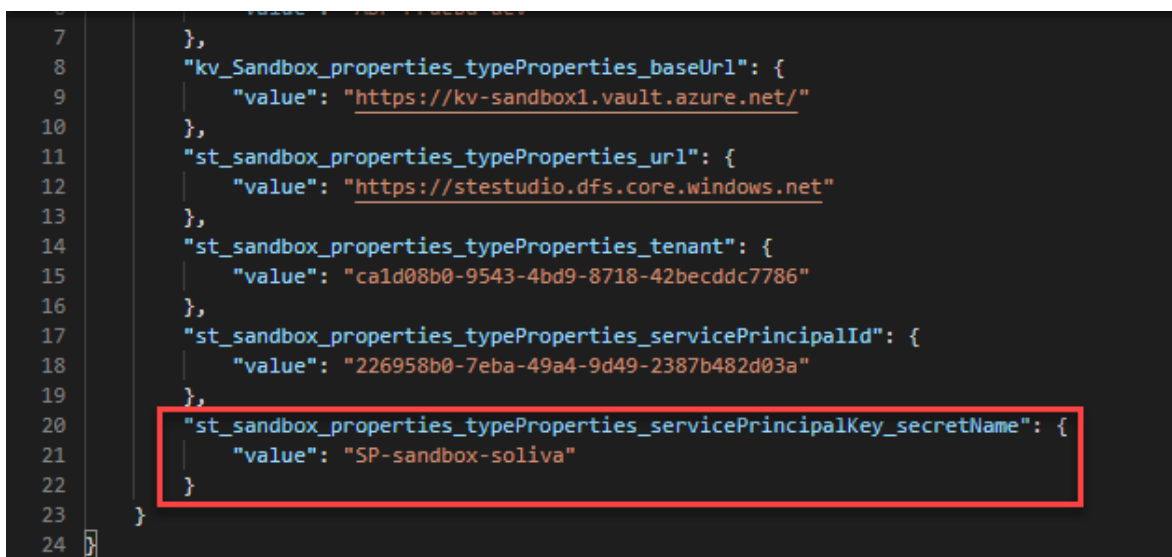
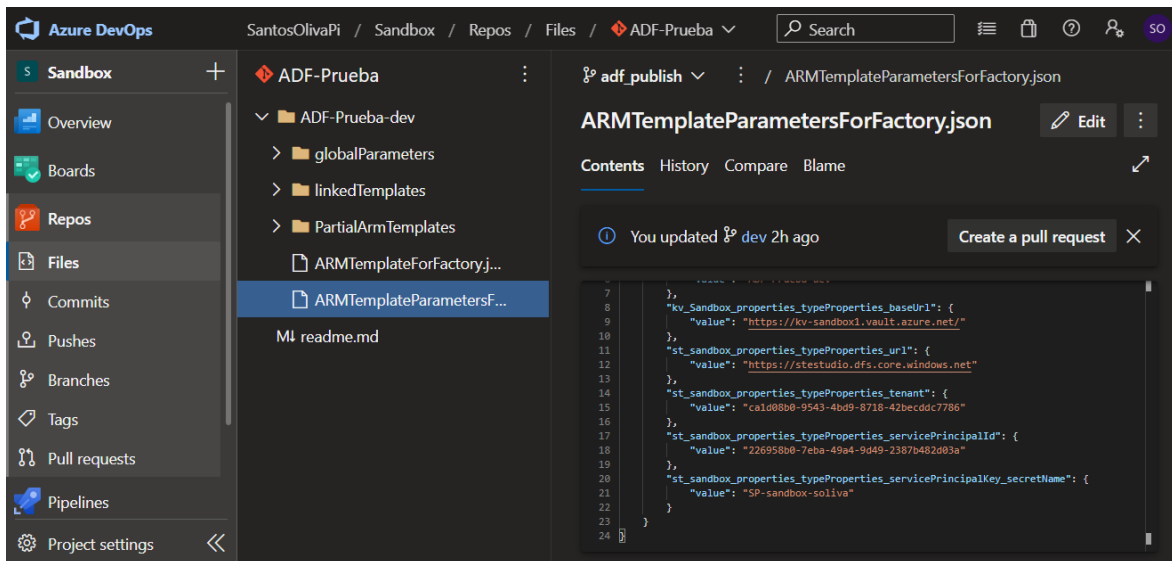


Luego modificaremos la plantilla de parametrización desde nuestro ADF



```
"Microsoft.DataFactory/factories/linkedServices": {  
  "properties": {  
    "typeProperties": {  
      "servicePrincipalKey": {  
        "secretName": "=",  
      },  
      "accountName": "=",  
      "username": "=",  
      "userName": "=",  
      "accessKeyId": "=",  
      "servicePrincipalId": "=",  
      "userId": "=",  
      "host": "=",  
      "clientId": "=",  
      "clusterUserName": "=",  
      "clusterSshUserName": "=",  
      "hostSubscriptionId": "=",  
      "clusterResourceGroup": "=",  
    },  
  },  
}
```

Con esto modificado, luego de publicar veremos los cambios desde DevOps





adf_publish / ADF-Prueba-dev / ARMTemplateForFactory.json

ARMTemplateForFactory.json

Commit Cancel

Contents Highlight changes

You updated dev 2h ago Create a pull request

```
565 name: "[concat(parameters('factoryName'), '/st_sandbox')]",
566 type: "Microsoft.DataFactory/factories/linkedServices",
567 apiVersion: "2018-06-01",
568 properties: {
569   "annotations": [],
570   "type": "AzureBlobFS",
571   "typeProperties": {
572     "url": "[parameters('st_sandbox_properties_typeProperties_url')]",
573     "tenant": "[parameters('st_sandbox_properties_typeProperties_tenant')]",
574     "servicePrincipalId": "[parameters('st_sandbox_properties_typeProperties_servicePrincipalId')]",
575     "servicePrincipalKey": {
576       "type": "AzureKeyVaultSecret",
577       "store": {
578         "referenceName": "kv_Sandbox",
579         "type": "LinkedServiceReference"
580       },
581       "secretName": "[parameters('st_sandbox_properties_typeProperties_servicePrincipalKey_secretName')]"
582     },
583   },
584 }
```

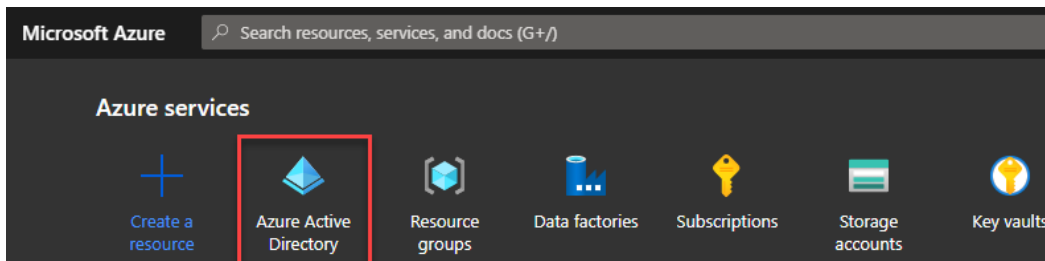


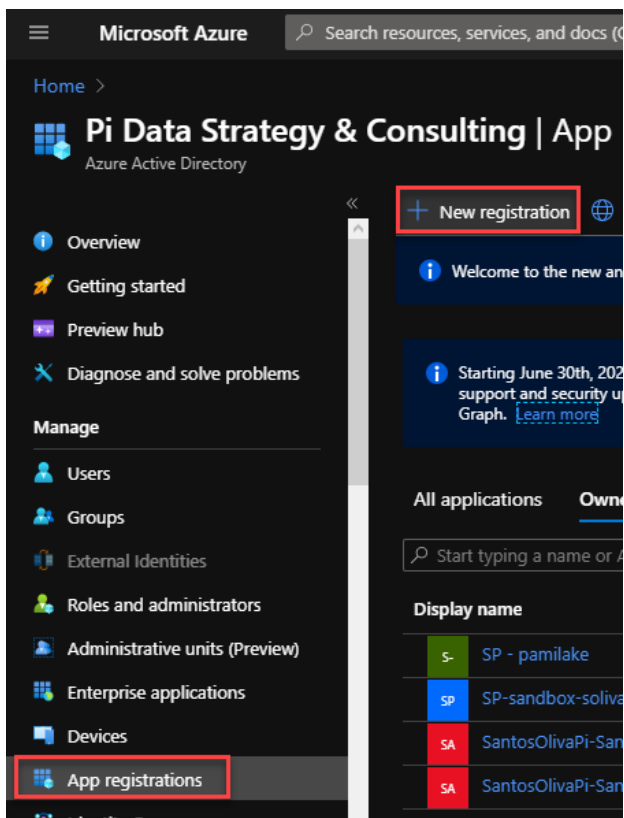
Service Principal

Un Service Principal [SP] es una identidad creada para su uso con aplicaciones, servicios hospedados y herramientas automatizadas que acceden a los recursos de Azure.

Este acceso está restringido por los roles asignados a la entidad de servicio, lo que permite controlar a qué recursos pueden tener acceso y en qué nivel. Por motivos de seguridad, se recomienda usar siempre entidades de servicio con las herramientas automatizadas, en lugar de permitirles iniciar sesión con una identidad de usuario.

Creación de un SP







[Home](#) > [Pi Data Strategy & Consulting | App registrations](#) >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

SP-sandbox-soliva



Supported account types

Who can use this application or access this API?

- ☒ Accounts in this organizational directory only (Pi Data Strategy & Consulting only - Single tenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- ☐ Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- ☐ Personal Microsoft accounts only

[Help me choose...](#)

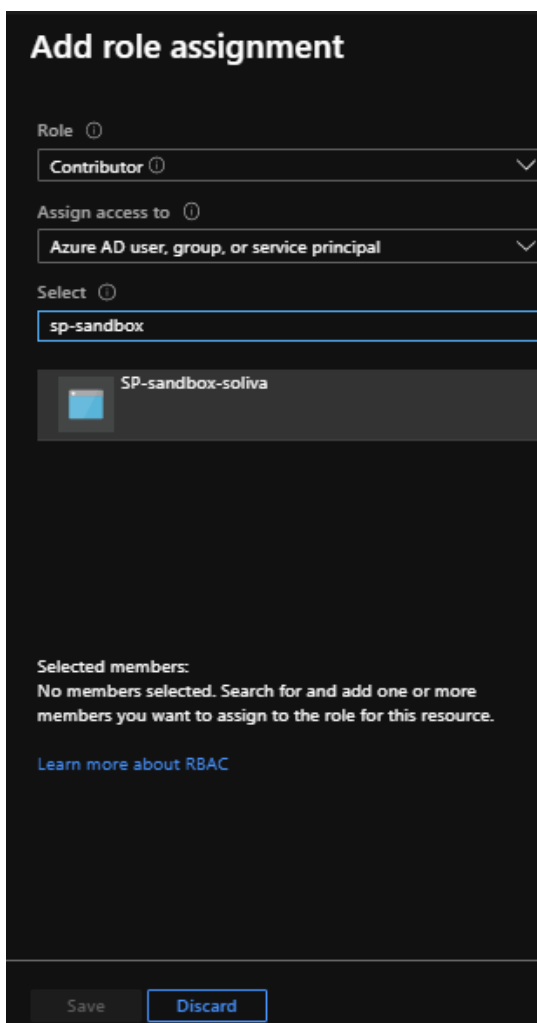
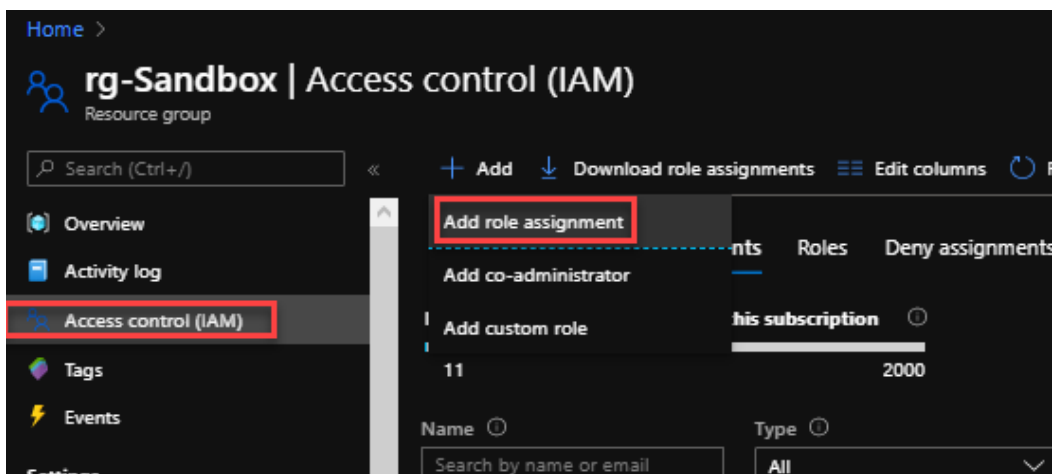
By proceeding, you agree to the [Microsoft Platform Policies](#)

Register



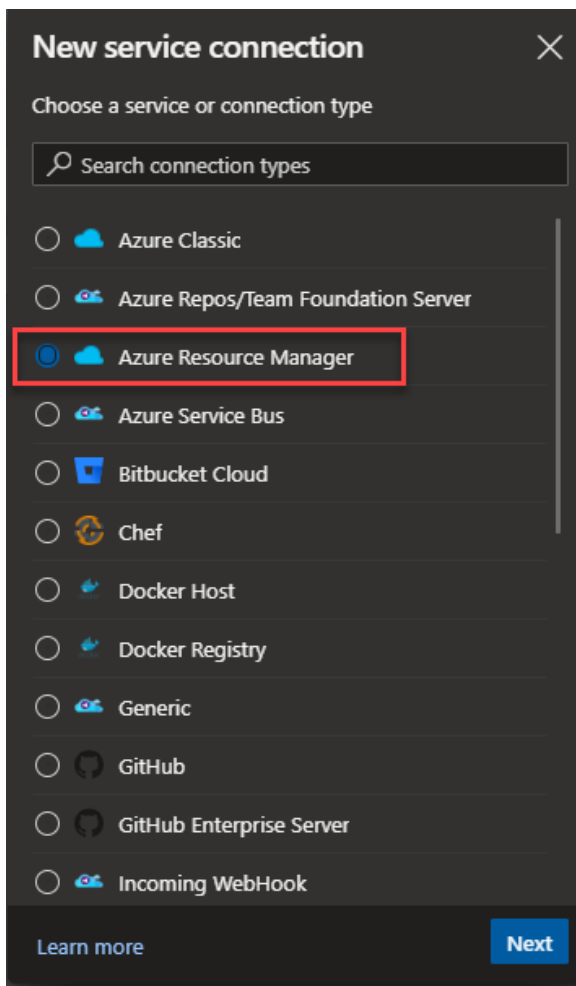
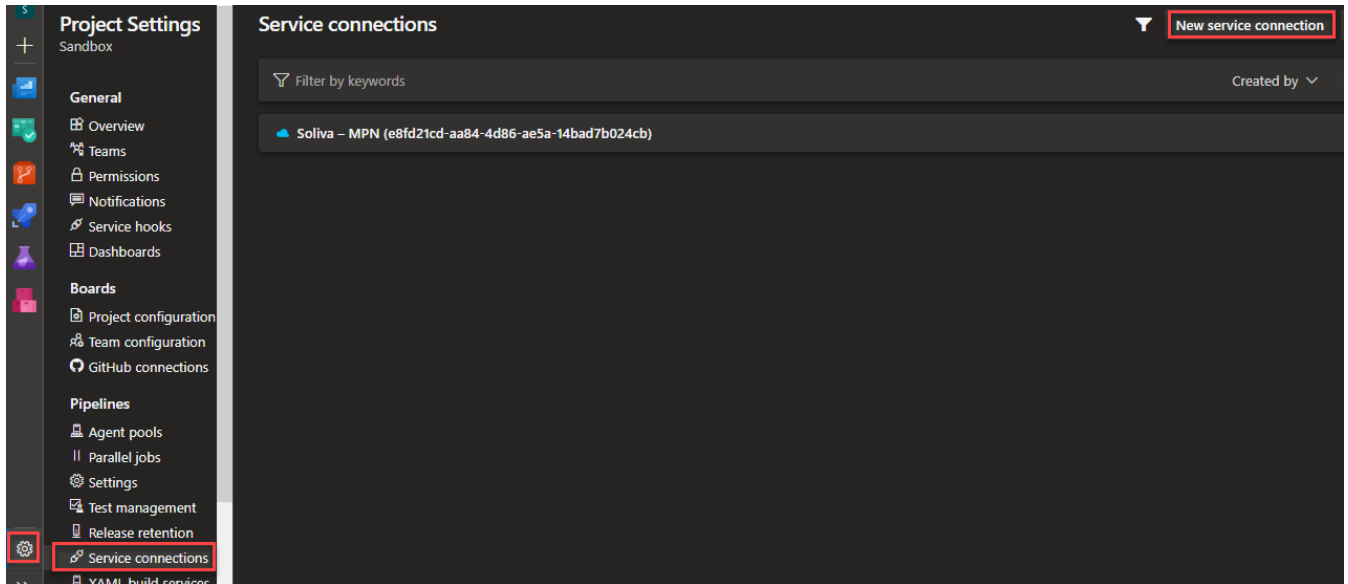
Acceso al grupo de recursos

Accediendo al grupo de recursos desde el portal de Azure





Acceso a DevOps





New Azure service connection

Azure Resource Manager

Authentication method

- ☒ Service principal (automatic) **Recommended**
- ☐ Service principal (manual)
- ☐ Managed identity
- ☐ Publish Profile

Need help choosing a connection type?

[Back](#) [Next](#)

New Azure service connection

Azure Resource Manager using service principal (automatic)

Scope level

- ☒ Subscription
- ☐ Management Group
- ☐ Machine Learning Workspace

Subscription

Soliva – MPN (e8fd21cd-aa84-4d86-ae5a-14bad... ▼

Resource group

rg-Sandbox ▼

Details

Service connection name

SP-sandbox-soliva

Description (optional)

Security

☒ Grant access permission to all pipelines

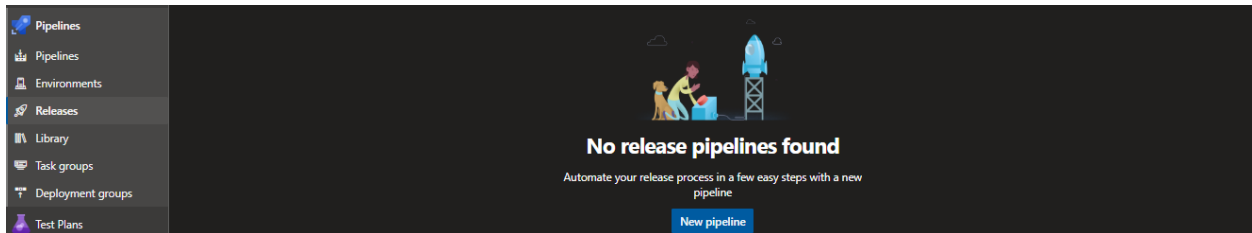
[Learn more](#) [Troubleshoot](#) [Back](#) [Save](#)



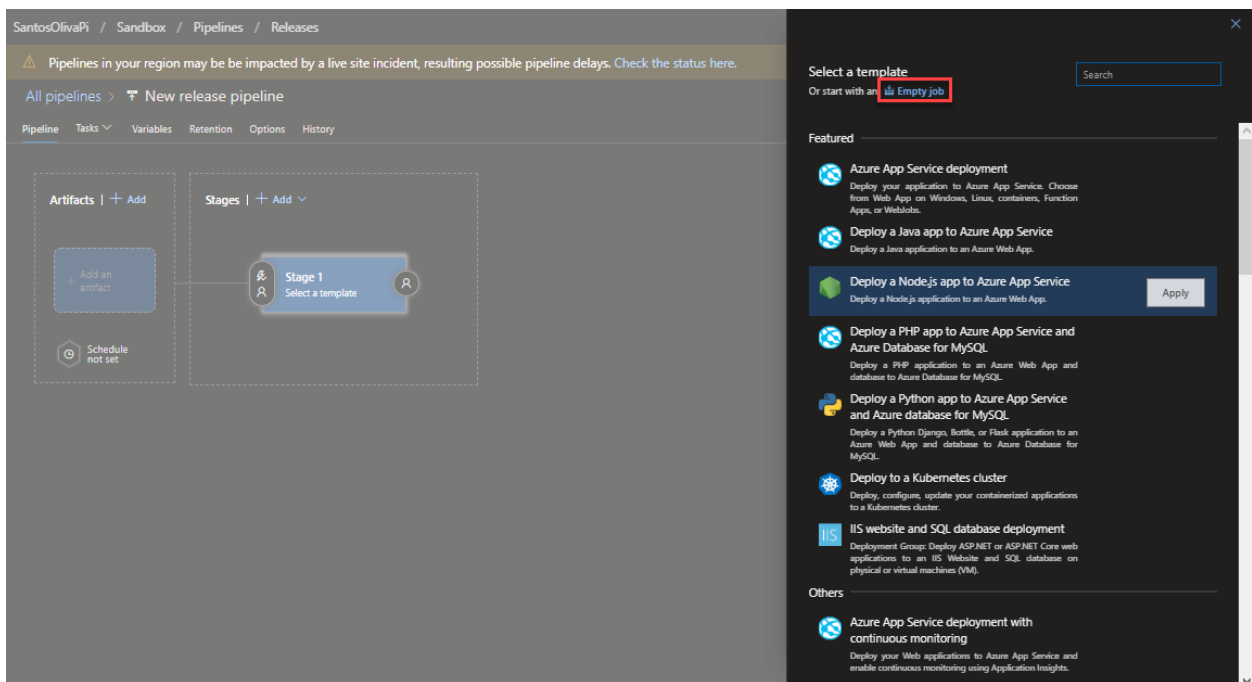
Releases

Ahora gestionaremos los releases desde DevOps.

Nos posicionamos en Pipelines-Releases y crearemos un nuevo pipeline

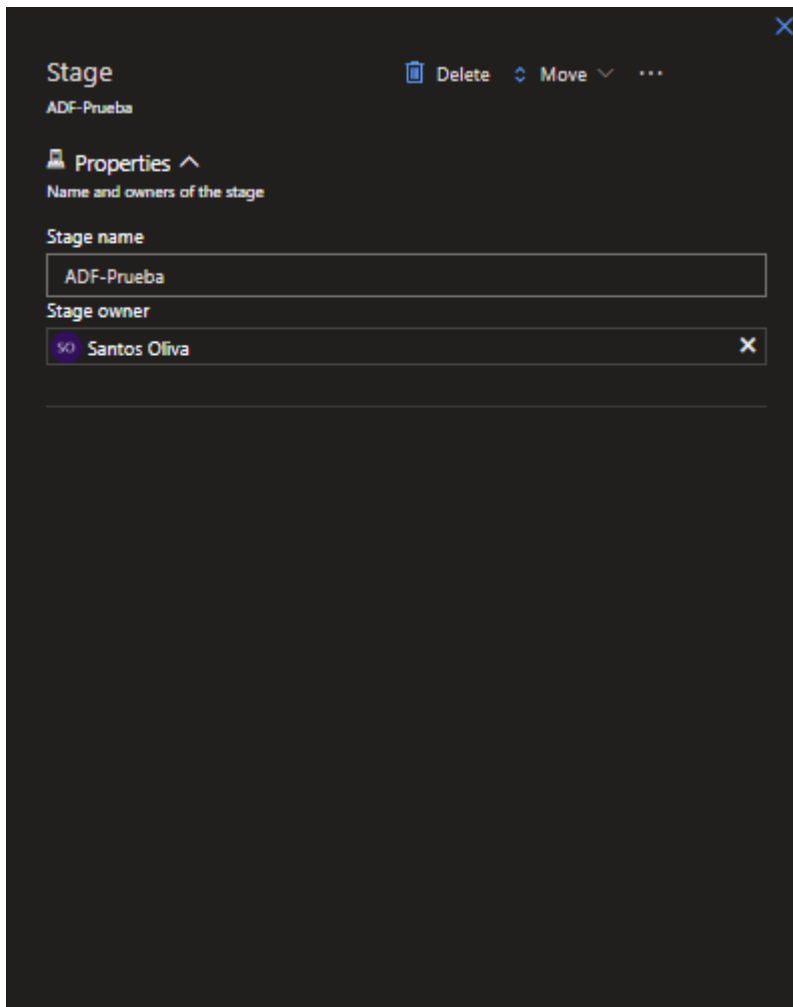


Y haremos clic en Empty Job



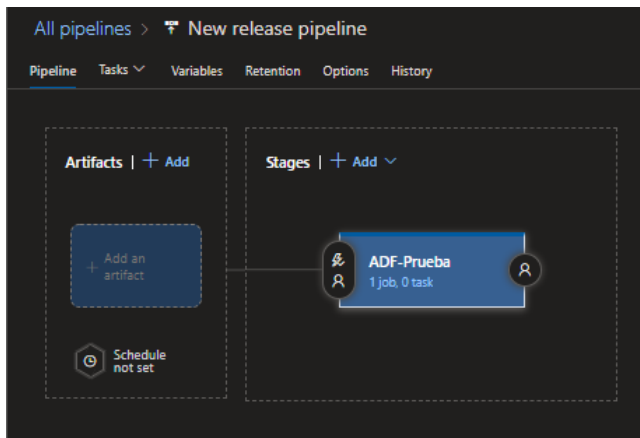


Aquí lo nombraremos y definiremos quien es el owner





Aquí iremos a Add an artifact y completaremos con los datos correspondientes



Utilizaremos Azure Repos como fuente, seleccionamos nuestro proyecto y respectivo repositorio.


Es importante que la rama por default seleccionada aquí sea `adf_publish`, en caso de no encontrarse disponible es porque no se ha realizado ningún Publish en el recurso ADF-prueba-dev.


Finalmente haremos clic en Add.





Add an artifact

Source type

 Build

 ✓ Azure Rep...

 GitHub

 TPVC

5 more artifact types ▾

Project * ⓘ

Sandbox ▾

Source (repository) * ⓘ

ADF-Prueba ▾

Default branch * ⓘ

adf_publish ▾

Default version * ⓘ

Latest from the default branch ▾

☐ Checkout submodules ⓘ

☐ Checkout files from LFS ⓘ

Shallow fetch depth ⓘ

Source alias * ⓘ

_ADF-Prueba

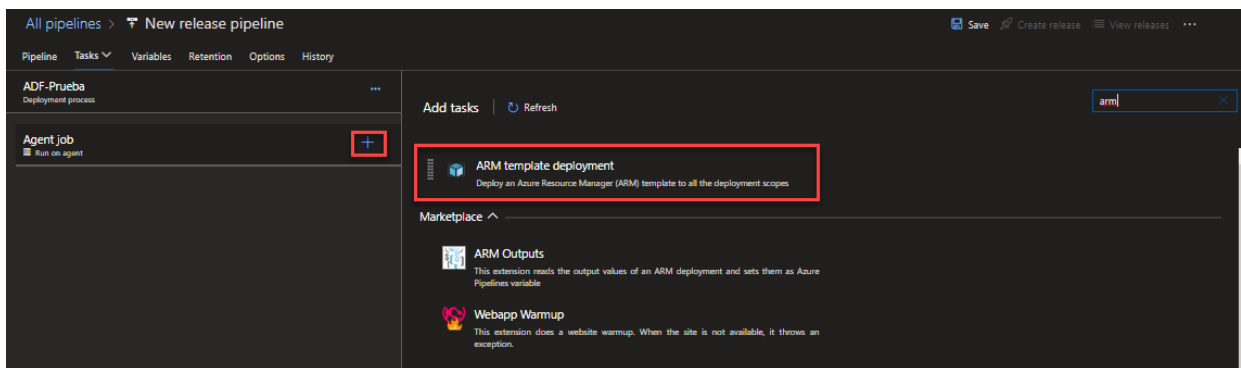
Add



Ahora haremos clic en 1 job, 0 task



Aquí se abrirá el Agent Job y agregamos la siguiente actividad.





Configuro el deployment de la siguiente manera.

La acción utilizada será la creación o update del grupo de recursos destino, este puede llegar a ser el mismo que el grupo de recursos origen, pero de ser así, el nombre del ADF será distinto.

La localización utilizada deberá ser la misma en la que se alojan los diferentes recursos.

En Override parameters se sobreescriben parámetros que aplican en el recurso fuente, en nuestro caso ADF-Prueba-dev, pero no en el de destino (ADF-Prueba-prod).

ARM template deployment ⓘ View YAML Remove

Task version 3.* ▾

Display name *
ARM Template deployment: Resource Group scope

Azure Details ^

Deployment scope * ⓘ
Resource Group ▾

Azure Resource Manager connection * ⓘ | Manage [Manage](#)
SP-sandbox-soliva ▾ ↻
 ⓘ Scoped to resource group 'rg-Sandbox'

Subscription * ⓘ
Soliva - MPN (e8fd21cd-aa84-4d86-ae5a-14bad7b024cb) ▾ ↻

Action * ⓘ
Create or update resource group ▾

Resource group * ⓘ
rg-Sandbox ▾ ↻

Location * ⓘ
East US 2 ▾ ↻



ba Save Create release View

ptions History

Create or update resource group

Resource group * ⓘ

rg-Sandbox

Location * ⓘ

East US 2

Template ^

Template location *

Linked artifact

Template * ⓘ

\$(System.DefaultWorkingDirectory)/_ADF-Prueba/ADF-Prueba-dev/ARMTemplateForFactory.json

Template parameters ⓘ

\$(System.DefaultWorkingDirectory)/_ADF-Prueba/ADF-Prueba-dev/ARMTemplateParametersForFactory.json

Override template parameters ⓘ

-factoryName "ADF-Prueba-prod" -st_sandbox_properties_typeProperties_servicePrincipalKey_secretName "secret-sp-sandbox"

Deployment mode * ⓘ

Incremental

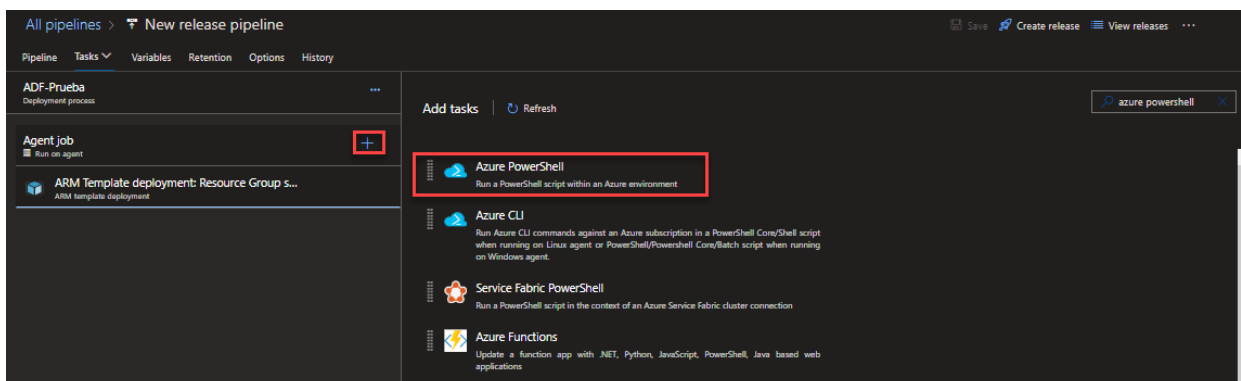
Advanced ▾

Control Options ▾

Output Variables ▾



De manera opcional, también se puede gestionar el pausar y reanudar los Triggers de ADF, via powershell



Esto lo realizaremos 2 veces, una para pausar, otra para reanudar.

Los Scripts utilizados serán los siguientes, reemplazando lo resaltado por los nombres que correspondan al entorno trabajado:

En nuestro caso, DataFactoryName será ADF-Prueba-prod y nuestro ResourceGroupName será rg-Sandbox

PAUSAR

```
$triggersADF = Get-AzDataFactoryV2Trigger -DataFactoryName $(DataFactoryName) -ResourceGroupName $(ResourceGroupName)

$triggersADF | ForEach-Object { Stop-AzDataFactoryV2Trigger -ResourceGroupName $(ResourceGroupName) -DataFactoryName $(DataFactoryName) -Name $_.name -Force }
```

REANUDAR

```
$triggersADF = Get-AzDataFactoryV2Trigger -DataFactoryName $(DataFactoryName) -ResourceGroupName $(ResourceGroupName)

$triggersADF | ForEach-Object { Start-AzDataFactoryV2Trigger -ResourceGroupName $(ResourceGroupName) -DataFactoryName $(DataFactoryName) -Name $_.name -Force }
```



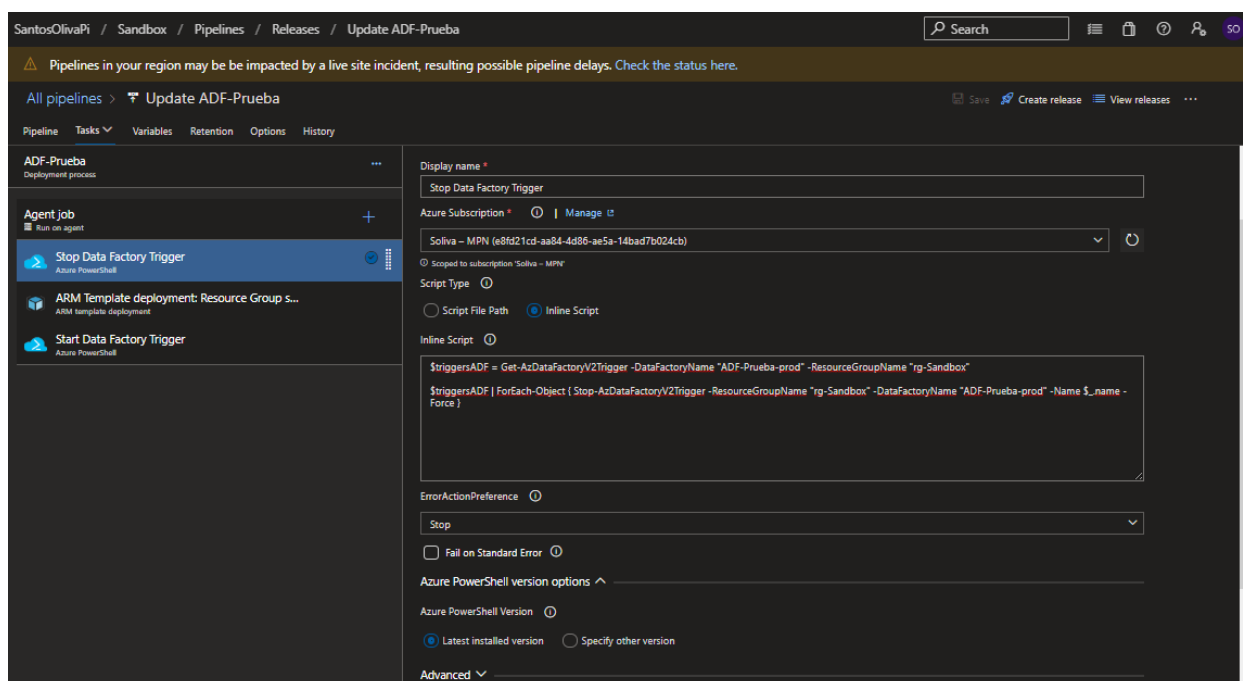
Como ejemplos terminados...

```
$triggersADF = Get-AzDataFactoryV2Trigger -DataFactoryName "ADF-Prueba-prod"-ResourceGroupName "rg-Sandbox"
```

```
$triggersADF | ForEach-Object { Stop-AzDataFactoryV2Trigger -ResourceGroupName "rg-Sandbox" -DataFactoryName "ADF-Prueba-prod"-Name $_.name -Force }
```

```
$triggersADF = Get-AzDataFactoryV2Trigger -DataFactoryName "ADF-Prueba-prod"-ResourceGroupName "rg-Sandbox"
```

```
$triggersADF | ForEach-Object { Start-AzDataFactoryV2Trigger -ResourceGroupName "rg-Sandbox" -DataFactoryName "ADF-Prueba-prod" -Name $_.name -Force }
```





Una vez gestionadas las tareas a realizar por el Agent Job, se hará clic en Create Release y haremos el Deploy

All pipelines > Update ADF-Prueba Save Create release

Create a new release

Update ADF-Prueba

Pipeline ^
Click on a stage to change its trigger from automated to manual.

ADF-Prueba

Stages for a trigger change from automated to manual. ⓘ

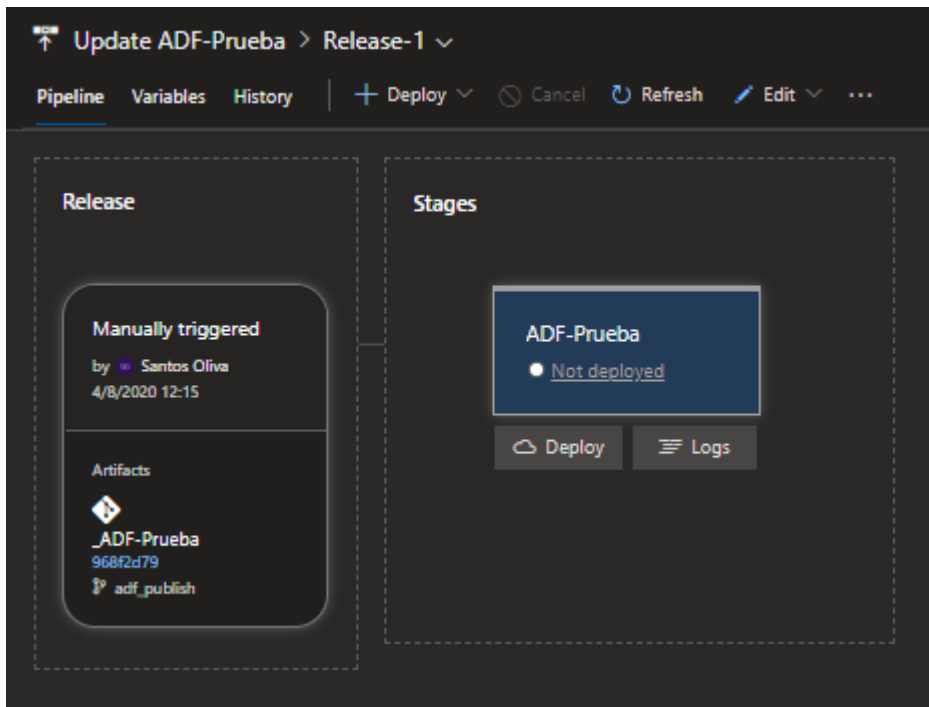
✓ ADF-Prueba

Artifacts ^
Select the version for the artifact sources for this release

Source alias	Version
_ADF-Prueba	968f2d79 (ARM template and parameters...)

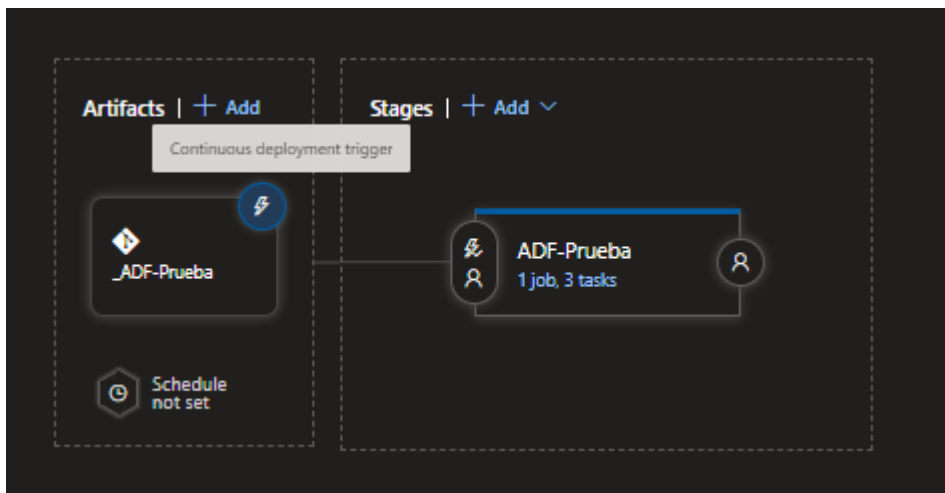
Release description

Create Cancel



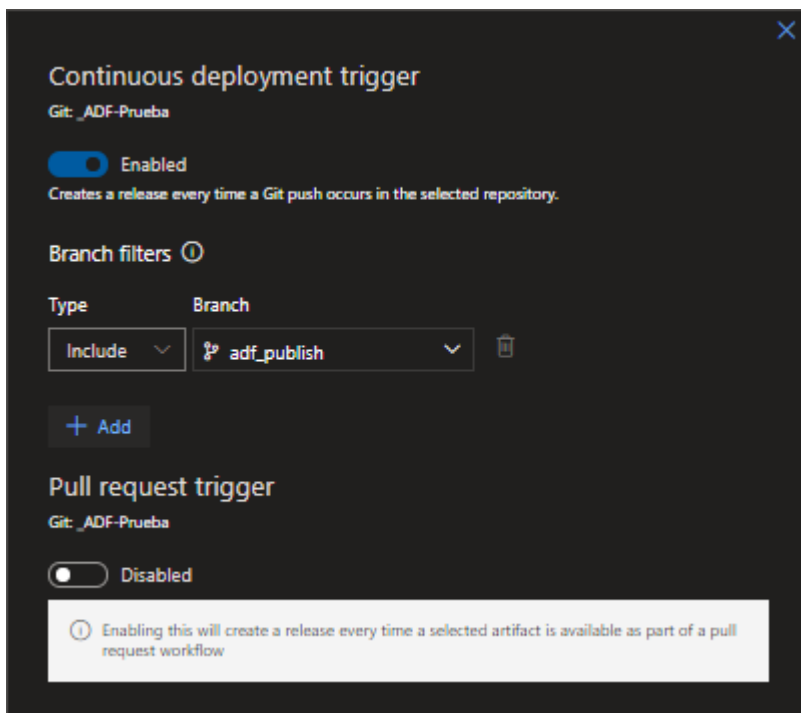
Con eso realizado, se realizará el primer deploy, en este caso manual y se verificara que las tareas programadas se ejecutan correctamente.

Editando el release automático, haremos clic en el rayo para gestionar el deploy continuo.



Aquí seleccionaremos las opciones deseadas,

- CD trigger
 - o para que se realice un deploy cada vez que sucede un push en Git (cuando confirman el pull request)
- Pull request trigger
 - o Se realiza el deploy cada vez que se realiza un pull request.





En nuestro caso habilitaremos el CD trigger, y finalmente haremos clic en Save.



Para mas información

<https://docs.microsoft.com/en-us/azure/data-factory/continuous-integration-deployment>