Model Scheduling with ADF - Design Proposal

This page includes the design proposal for the model scheduling using Azure Data Factory (ADF).

Table of Contents

Introduction

AB Initio schedules models in on-premise environment. With the implementation in the cloud at Azure, a new cloud oriented way of scheduling is needed. Azure Data Factory (ADF) is proposed for that purpose. The proposed workflow with ADF works in parallel in Data Discovery, QA and Prod environments mentioned in Design for Model Pipelines (MLOps)

Existing Ways of Scheduling

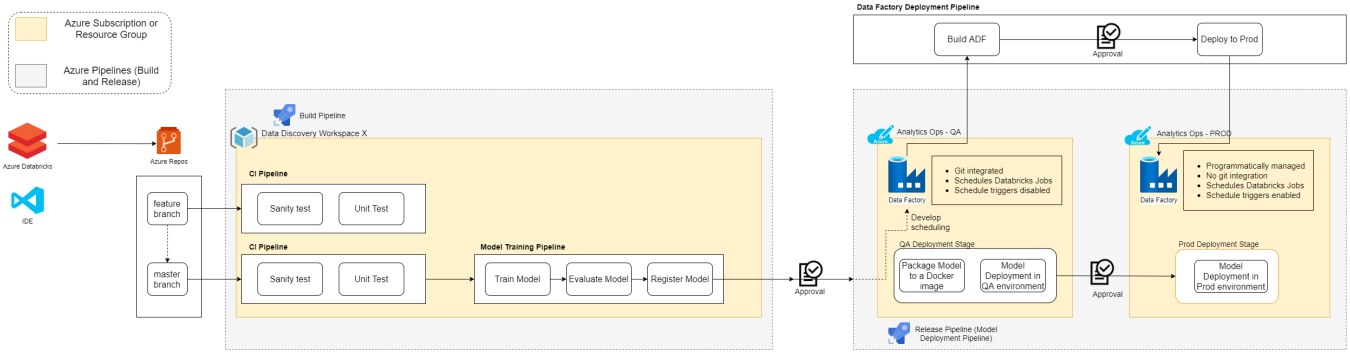
AB Initio is used for scheduling in on premise environment. Models are developed by the data scientists. Afterward, these model are moved QA/prod equivalents on premise. Scheduling is managed by people responsible for these environments and no scheduling is present before that. Before a model runs in its schedule, there are some prerequisite checks such as confirming that a certain dataset exists, as well as passing necessary parameters. The way of scheduling in cloud should not introduce a new flow in terms of user access and should distribute roles in a similar fashion.

Implementation Phases

- Phase 1: This a preliminary phase simplifying Phase 2 based on the meeting on 2021-06-29. It includes a workflow that meets the requirements with an emphasis on keeping the current (on-premise) development & maintenance responsibilities. Data scientists develop & register models with no scheduling responsibility on their part. Scheduling is handled by responsibles of QA and prod environments. ADF is present in those two environments. Intended for a short term solution.
- Phase 2: With this phase, data scientists gains ability to manage scheduling. ADF, the tool responsible for scheduling. The architecture is extended into three different environments providing a full workflow for it. Requiring more development time, this architecture is recommended for the long term. In the short term, it shifts scheduling responsibility to data scientists, which may put additional pressure on them.

Phase 1

Phase 1 - Data Factory Workflow



From a developer / data scientist point of view, no action needs to be taken for scheduling. They develop and register the model as planned.

From a QA / prod responsible person point of view, the following are the steps during development

1. Model is moved to QA environment with an approval
2. ADF pipelines are created for checking the prerequisites of the model execution (presence of a dataset etc) and executing relevant Databricks job (at the ADF)
3. A scheduled trigger is created for the model (at the ADF). Triggers are disabled at QA environment and are run manually for testing purposes.
4. Data Factory Deployment Pipeline is run, which moves the ADF at the QA environment to the production environment. The pipeline also enables the triggers at prod which makes triggers active.

Phase 1 - Azure Resources

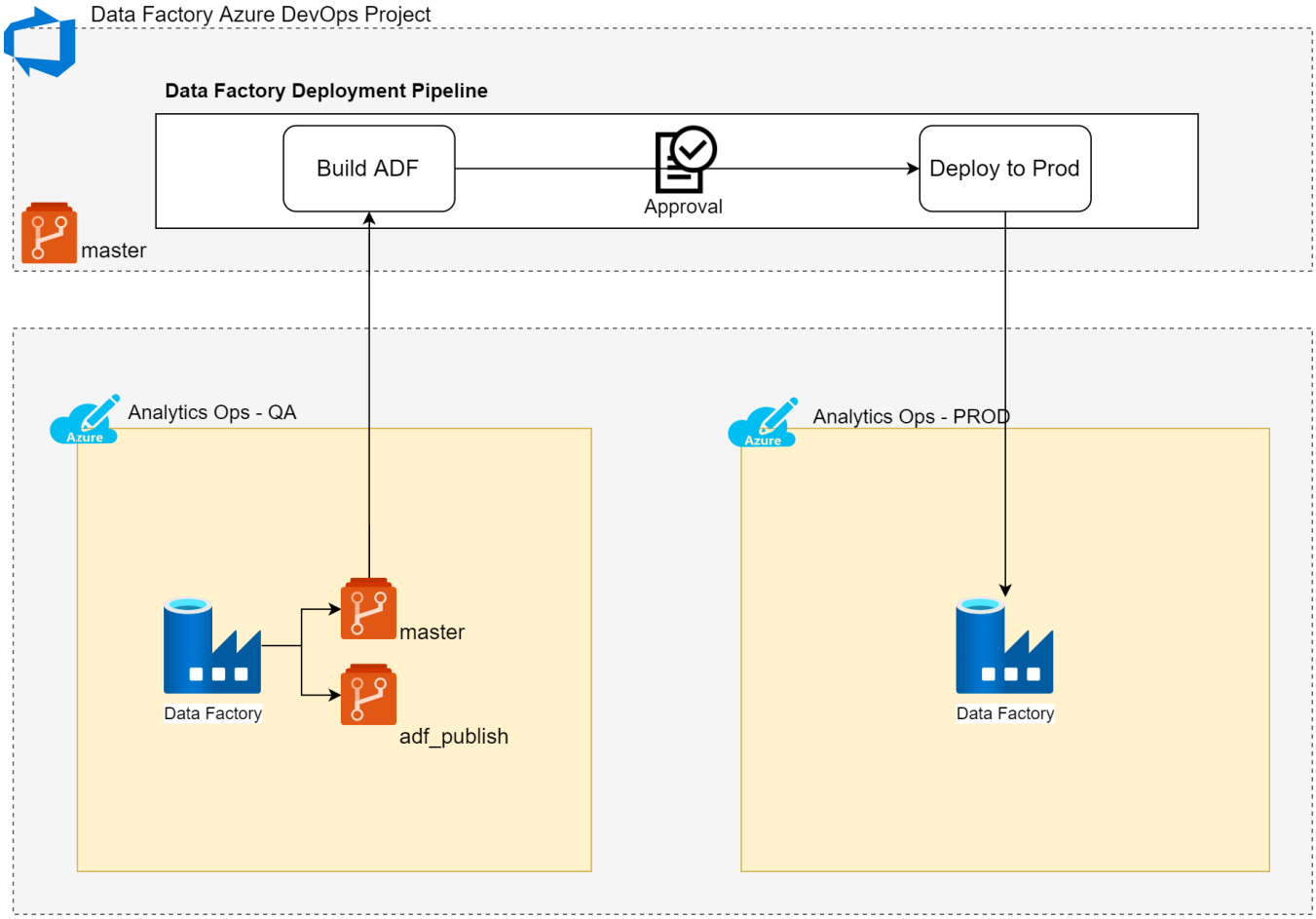
- Deployment of the contents of ADFs are handled separately from the deployment of the resource itself. To be covered under **DevOps** subsection.
- QA and prod environments each have a single data factory resource. QA ADF is managed by responsibles of this environment. Prod ADF is programmatically managed by Data Factory Deployment Pipeline.
- These ADFs authenticate themselves at their corresponding Databricks workspaces with their service principles (SPs). ADF SPs should have contributor rights over relevant Databricks workspaces and this configuration needs to be done manually.

Phase 1 - User Access

- The design focuses on giving access to ADF resources with already existing roles, without creating extra complexity. The reason why access is important is that having access to information about scheduling a model also requires giving information about which notebooks & clusters that model is running with.
- For the people working at data discovery (DD) environments, who are mainly developers and data scientists, no additional access needed.
- People who has access to QA and prod environment also has access to the DevOps Project that contains Data Factory Deployment Pipeline. It is important note that these people will see all the scheduling information of all of the models.

Phase 1 - DevOps

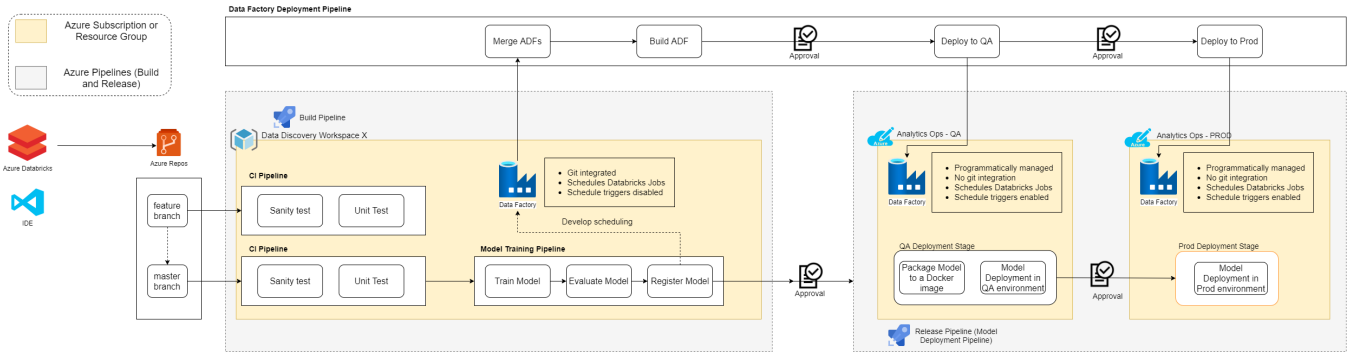
- Data Factory Deployment Pipeline performs the following stages
  - **Build ADF**: As a part of the CI, ADF resource is built in to an ARM template to be used for deployment. This is done through using npm commands on the DevOps agent. The end result is an artifact (contains ARM templates, parameters and scripts) at Azure DevOps. This ensures ease of versioning.
  - **Deploy to prod**: As a part of the CD, the previously built artifact is deployed to the QA environment with an approval. This stage can be automatically triggered after the previous stage. If the need arises (due to a high number of deployment requests), this stage can be configured to trigger manually.



- This DevOps flow is an enhanced version of the ADF CI/CD lifecycle mentioned at Microsoft Documentation. The flow in the link uses automatically generated ARM template located in adf_publish branch to deploy to the next environment (prod in this case), which is not intended for the current use case at it can be seen at the image above. Bypassing this branch allows some steps to be done programmatically and to avoid manual labor.
- The flow in the link implies a life cycle consisting of a single ADF resource in QA environment and moving it to the production environment. The QA ADF resource is git integrated with a master (main) branch. Developers do their work, and publish changes on ADF. These changes later on moved to programmatically generated adf_publish branch which consists of an ARM template to deploy across environments.
- Some benefits of the proposed structure
    - Publishing handled programmatically (no need to publish through the UI)
    - QA ADF is git integrated and the version of the master branch is moved to the next environment. Since master is updated by pull requests, it gives a more control over what is moved to the next environment and what is not.
    - Developers are able to build feature branches and work there. This benefit will have a smaller impact in the early stages where only the scheduling handled by ADF. If in the future requirement changes and ADF performs more tasks, there is a freedom to have a controlled flow.
    - A DevOps artifact is generated after Build ADF stage. This eases versioning.
    - Even though not functional in the prod environment, adf_publish branch is still present in QA ADF. This means that developers are able to publish changes and switch to published version of the ADF at QA workspace, allowing them to test features such as ADF debugging which otherwise is not possible while being git integrated.

Phase 2

Phase 2 - Data Factory Workflow



From a developer / data scientist point of view, the following are the steps during development

1. In the model training pipeline, model is registered
2. Developer / data scientist creates ADF pipelines for checking the prerequisites of the model execution (presence of a dataset etc) and executing relevant Databricks job (at the ADF)
3. Developer / data scientist creates the scheduled trigger for model (at the ADF). At this stage, schedule is present but not activated.

From a QA / prod responsible person point of view, the following are the steps during development

1. With an approval, ADF deployment pipeline brings changes to QA environment. Schedules are enabled in this environment.
2. With an approval and after the scheduling works as expected at QA, ADF deployment pipeline brings changes to prod environment. Schedules are enabled in this environment.

QA / prod responsible people also have access to creating and modifying schedules in an indirect way. They do have access to the ADF that is being deployed to QA and prod, but they do not work directly at DD workspace which is the main place for development for schedules. If the need arises, a direct way for them to work on schedules can be included in the workflow. In that case, the additional development by them would be done at QA environment, and the Data Factory Deployment would deploy to production based on the ADF at QA instead of basing it purely on DD environments.

Phase 2 - Azure Resources

- Deployment of the contents of ADFs are handled separately from the deployment of the resource itself. To be covered under **DevOps** subsection.
- There are multiple data factories located in their corresponding data discovery (DD) workspace. QA and prod environments each have a single data factory resource. DD ADFs are managed by developers / data scientists. QA and Prod ADFs are programmatically managed by Data Factory Deployment Pipeline.
- These ADFs authenticate themselves at their corresponding Databricks workspaces with their service principles (SPs). ADF SPs should have contributor rights over relevant Databricks workspaces and this configuration needs to be done manually.

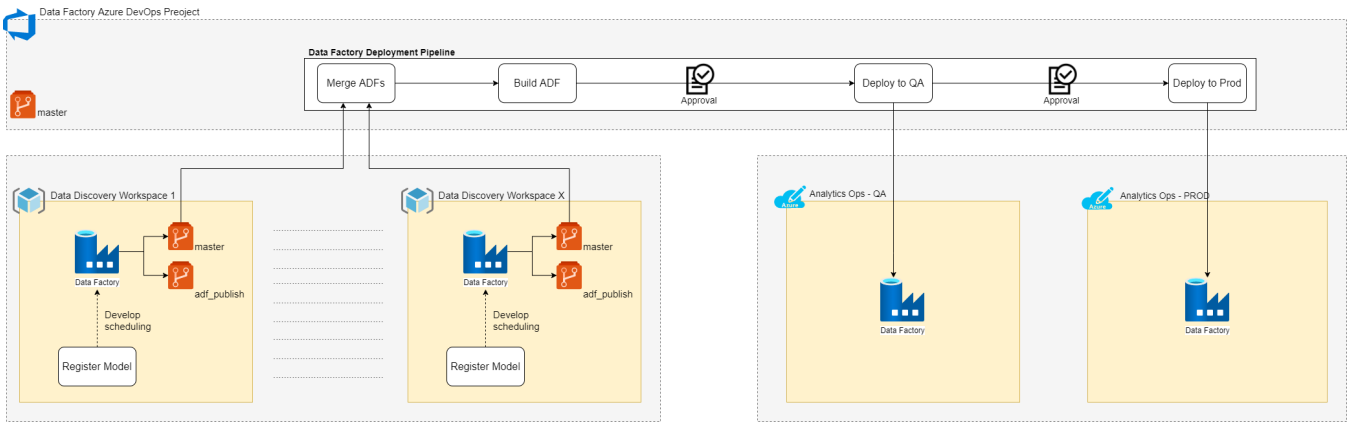
Phase 2 - User Access

- The design focuses on giving access to ADF resources with already existing roles, without creating extra complexity. The reason why access is important is that having access to information about scheduling a model also requires giving information about which notebooks & clusters that model is running with.
- Developers and data scientists have access to the data discovery workspace they are working with including the data factory resource and git access for that that workspace's corresponding repo. They only have access to scheduling information of models they are working with.
- People who has access to QA and prod environment also has access to the DevOps Project that contains Data Factory Deployment Pipeline. It is important note that these people will see all the scheduling information of all of the models.

Phase 2 - DevOps

- The DevOps structure in phase 2 builds on top of phase 1. See the grey lines for the additions in this phase. The main difference is that this phase includes multiple ADFs in DD workspaces for development which requires an additional stage at the DevOps pipeline to merge these ADFs into one.
- Data Factory Deployment Pipeline performs the following stages
    - **Merge ADFs**: Merge multiple ADFs present in DD workspaces into a single one. The pipeline connects to git repos that contain different ADFs, fetches JSON files that includes ADF definition (ADF pipelines, schedule info, connections) and brings them together. Additional checks and tests need to developed here to avoid collision.
    - **Build ADF**: As a part of the CI, the previously merged ADF resource is built in to an ARM template to be used for deployment. This is done through using npm commands on the DevOps agent. The end result is an artifact at Azure DevOps. This ensures ease of versioning.
    - **Deploy to QA**: As a part of the CD, the previously built artifact is deployed to the QA environment with an approval. This stage can be automatically triggered after the previous stage. If the need arises (due to a high number of deployment requests), this stage can be configured to trigger manually.

- **Deploy to prod**: Similar to the previous step, ADF is deployed to prod environment.



- This DevOps flow is an enhanced version of the ADF CI/CD lifecycle mentioned at Microsoft Documentation.
- The life cycle in the link consists of a single ADF resource and moving it across different environments. The development ADF resource is git integrated with a master (main) branch. Developers do their work, and publish changes on ADF. These changes later on moved to programmatically generated adf_publish branch which consists of an ARM template to deploy across environments.
- A similar approach cannot be used due to the fact that there are multiple ADFs present at multiple DD workspaces that later on need work as a single resource in QA and prod.
- Some benefits of the proposed structure
    - Publishing handled programmatically (no need to publish through the UI)
    - Each DD ADF are git integrated and the version of the master branch is moved to the next environment. Since master is updated by pull requests, it gives a more control over what is moved to the next environment and what is not.
    - Developers are able to build feature branches and work there. This benefit will have a smaller impact in the early stages where only the scheduling handled by ADF. If in the future requirement changes and ADF performs more tasks, there is a freedom to have a controlled flow.
    - A DevOps artifact is generated after Build ADF stage. This eases versioning.
    - Even though not functional in the QA and prod environment, adf_publish branch is still present in DD ADFs. This means that developers are able to publish changes and switch to published version of the ADF at DD workspaces, allowing them to test features such as ADF debugging which otherwise is not possible while being git integrated.

Scheduling Databricks Notebooks / Jobs

Interaction between ADF and Databricks workspace is a common concern for both Phase 1 and Phase 2 which can be handled in a similar fashion. This subsection investigates how Data Factory should work with Databricks jobs. There are two main ways of scheduling Databricks jobs which has not been decided yet.

- Option 1 (recommended): Schedule a Databricks notebook from ADF.
    - With this workflow, ADF creates a temporary job at Databricks during an execution of a notebook.
    - On the ADF side, developers specificy Databricks notebook directory. ADF can either create a new cluster or use an existing one with specifying a cluster ID.
    - ADF authenticates itself through its SP
    - Job execution is fully traceable/visible from ADF but not from Databricks.
- Microsoft Documentation: ADF Databricks Notebook activity
- Development guide: Model Scheduling Development with ADF

- Option 2: Perform an API call from ADF to a Databricks job
    - With this workflow, ADF performs an API call to already existing Databricks Jobs.
    - These jobs need to be created beforehand either by hand or programmatically. Data Factory performs an API call to invoke these jobs through ADF Web Activity.
    - This will make job execution tracable/visible from Databricks but not ADF.
    - Requires more development time compared to option 1, and is not recommended due to the fact that this approach moves handling of jobs and scheduling in separate locations (If a job is changed at Databricks, then it needs to be changed at ADF as well), which could lead to diverging resources in the long term.
- Microsoft Documentation: Run Databricks Jobs with Service Principals, Databricks Jobs API, ADF Web Activity

	Option 1	Option 2
Principle	Running temporarily generated Databricks jobs from ADF via internal ADF Run Databricks Notebook activity	Running already existing Databricks jobs ADF via API calls to Databricks
Scheduling and Prerequisite Check	Developed and visible at ADF	Developed and visible at ADF
Existing Databricks Jobs	Existing Databricks jobs cannot be used. Temporary jobs should be defined at ADF.	Existing Databricks jobs can and should be used. ADF can run but not modify a job. Jobs maintained at Databricks.
Job Run Details and Job Logs	Visible at ADF via a link to Databricks. Not visible at Databricks via its UI.	Visible at Databricks but not ADF. No direct link provided between ADF and Databricks.
Model Logs	Custom model logging to an external location. No difference from option 2	Custom model logging to an external location. No difference from option 1
Error Tracking	On failure, ADF is informed about the status, opening up options to automatically manage it.	API call response is not as informative. It informs if the job is started but not the job details. Additional checkpoints needed.
DevOps	ADF content is managed in git. No difference from option 2	ADF content is managed in git. No difference from option 1
Microsoft Support	ADF manages the run with ADF Run Databricks Notebook activity, responsibility on ADF (Microsoft)	API call is done to run a Databricks job. The call should be maintained internally.
Development Time	A faster time to market	Requires 3-4 times more development time compared to option 1. Not only during initial development but also when new scheduling info

During meeting on 2021-07-07 it is discussed that the initial implementation will follow Option 1 and there will be a shift toward Option 2 if the need occurs.

Meeting Video 2021-07-02