Activity 05: CI/CD with Synapse Workspaces

Intructor-led, interactive demo (30 minutes)

1. Configure a Git repository for the workspace
    1. Explain the collaboration and publishing branches
2. Switch between Synapse Live and the Git-backed workspace
    1. Explain how Synapse artifacts are handled in the two versions of the workspace
    2. Observe that publishing is not allowed in Synapse Live
    3. Using feature branches (create from Synapse Studio)
3. The lifetime of a Synapse artifact
    1. Create
    2. Commit/publish
    3. Update
    4. Commit/publish
4. CI/CD pipelines with GitHub Actions(see instructions to set up a release in GitHub Actions)
    1. Deploy to a production Synapse workspace

Best practices for Git integration

Time permitting, go through the following Git integration best practices:

- **Permissions**. After you have a git repository connected to your workspace, anyone who can access to your git repo with any role in your workspace will be able to update artifacts, like sql script, notebook,spark job definition, dataset, dataflow and pipeline in git mode. Typically you don't want every team member to have permissions to update workspace. Only grant git repository permission to Synapse workspace artifact authors.
- **Collaboration**. It's recommended to not allow direct check-ins to the collaboration branch. This restriction can help prevent bugs as every check-in will go through a pull request review process described in Creating feature branches.
- **Synapse live mode**. After publishing in git mode, all changes will be reflected in Synapse live mode. In Synapse live mode, publishing is disabled. And you can view, run artifacts in live mode if you have been granted the right permission.
- **Edit artifacts in Studio**. Synapse studio is the only place you can enable workspace source control and sync changes to git automatically. Any change via SDK, PowerShell, will not be synced to git. We recommend you always edit artifact in Studio when git is enabled.

Best practices for CI/CD

Time permitting, go through the following Git integration best practices:

If you're using Git integration with your Azure Synapse workspace and you have a CI/CD pipeline that moves your changes from development to test, and then to production, we recommend these best practices:

- **Integrate only the development workspace with Git**. If you use Git integration, integrate only your development Azure Synapse workspace with Git. Changes to test and production workspaces are deployed via CI/CD and don't need Git integration.

- **Prepare pools before you migrate artifacts**. If you have a SQL script or notebook attached to pools in the development workspace, use the same name for pools in different environments.
- **Sync versioning in infrastructure as code scenarios**. To manage infrastructure (networks, virtual machines, load balancers, and connection topology) in a descriptive model, use the same versioning that the DevOps team uses for source code.
- **Review Azure Data Factory best practices**. If you use Data Factory, see the best practices for Data Factory artifacts.

Best practices for Synapse Pipelines artifacts

Time permitting, go through the following Git integration best practices:

If you're using Git integration with your data factory and have a CI/CD pipeline that moves your changes from development into test and then to production, we recommend these best practices:

- **Git integration**. Configure only your development data factory with Git integration. Changes to test and production are deployed via CI/CD and don't need Git integration.

- **Pre- and post-deployment script**. Before the Resource Manager deployment step in CI/CD, you need to complete certain tasks, like stopping and restarting triggers and performing cleanup. We recommend that you use PowerShell scripts before and after the deployment task. For more information, see Update active triggers. The data factory team has provided a script to use located at the bottom of this page.

- **Integration runtimes and sharing**. Integration runtimes don't change often and are similar across all stages in your CI/CD. So Data Factory expects you to have the same name and type of integration runtime across all stages of CI/CD. If you want to share integration runtimes across all stages, consider using a ternary factory just to contain the shared integration runtimes. You can use this shared factory in all of your environments as a linked integration runtime type.

- **Managed private endpoint deployment**. If a private endpoint already exists in a factory and you try to deploy an ARM template that contains a private endpoint with the same name but with modified properties, the deployment will fail. In other words, you can successfully deploy a private endpoint as long as it has the same properties as the one that already exists in the factory. If any property is different between environments, you can override it by parameterizing that property and providing the respective value during deployment.

- **Key Vault**. When you use linked services whose connection information is stored in Azure Key Vault, it is recommended to keep separate key vaults for different environments. You can also configure separate permission levels for each key vault. For example, you might not want your team members to have permissions to production secrets. If you follow this approach, we recommend that you to keep the same secret names across all stages. If you keep the same secret names, you don't need to parameterize each connection string across CI/CD environments because the only thing that changes is the key vault name, which is a separate parameter.

- **Resource naming**. Due to ARM template constraints, issues in deployment may arise if your resources contain spaces in the name. The Azure Data Factory team recommends using '_' or '-' characters instead of spaces for resources. For example, 'Pipeline_1' would be a preferable name over 'Pipeline 1'.

- **Exposure control and feature flags**. When working on a team, there are instances where you may merge changes, but don't want them to be run in elevated environments such as PROD and QA. To handle this scenario, the ADF team recommends the DevOps concept of using feature flags. In ADF, you can combine global parameters and the if condition activity to hide sets of logic based upon these environment flags.

Resources

Source control in Synapse Studio

Continuous integration and delivery for an Azure Synapse Analytics workspace

Continuous integration and delivery in Azure Data Factory