How to manage Microsoft Fabric as GitOps in the CNS Solution

Executive summary

We are pleased to submit our proposal for the Cognitive Software solution (CNS) based on Hyper Scale Cloud. We are confident that we can meet the requirements and expectations of Ericsson Cognitive Network Solutions (CNS) and provide the best platform and services for their Software as a Service (SaaS) offering of Cognitive Software.

Our solution for Cognitive Software is a cloud-native software based on Microsoft Fabric, which aims to optimize the Total Cost of Ownership (TCO) for customers by providing network diagnostics, root cause analysis, and network optimization actions for the Radio Access Network (RAN) domain. Cognitive Software is planned to be a SaaS offering hosted on our public Azure cloud. Cognitive Software will leverage our serverless capabilities, PaaS/SaaS services, MLOps, and AI platform to achieve faster time to market, scalability, and reliability.

One key component of our solution is Microsoft Fabric (here after called Fabric) and this document is intended to clarify how to work with Fabric from a GitOps perspective in order to drive automation and keep cost of operations to a minimum.

Introduction to how to use GitOps to manage Fabric in CNS

GitOps in Fabric is designed to streamline the development and deployment process, ensuring that changes are integrated and delivered efficiently. This document provides an overview of how to work with GitOps in Microsoft Fabric

- One-click deployment with no manual steps except approval gates.
- Reuse of existing Infrastructure as Code (IaC) deployment mechanisms such as Terraform, Azure Resource Manager or similar.
- Understanding how to deal with Infrastructure as a Service (IaaS) and Platform as a Service (PaaS), while Software as a Service (SaaS) for data services is relatively new.
- Awareness of various options in Fabric for GitOps.

Git integration in Microsoft Fabric enables developers to integrate their development processes, tools, and best practices straight into the Fabric platform. It allows developers who are developing in Fabric to:

- Backup and version their work
- Revert to previous stages as needed
- Collaborate with others or work alone using Git branches
- Maintain artifacts for deployment as code
- Connect / disconnect a workspace to a git branch
- Commit to /update from git
- Get item status
- Deploy all workspace content
- Deploy selected items
- Create a pipeline
- Assign a workspace to a pipeline stage
- Manage user access to a pipeline

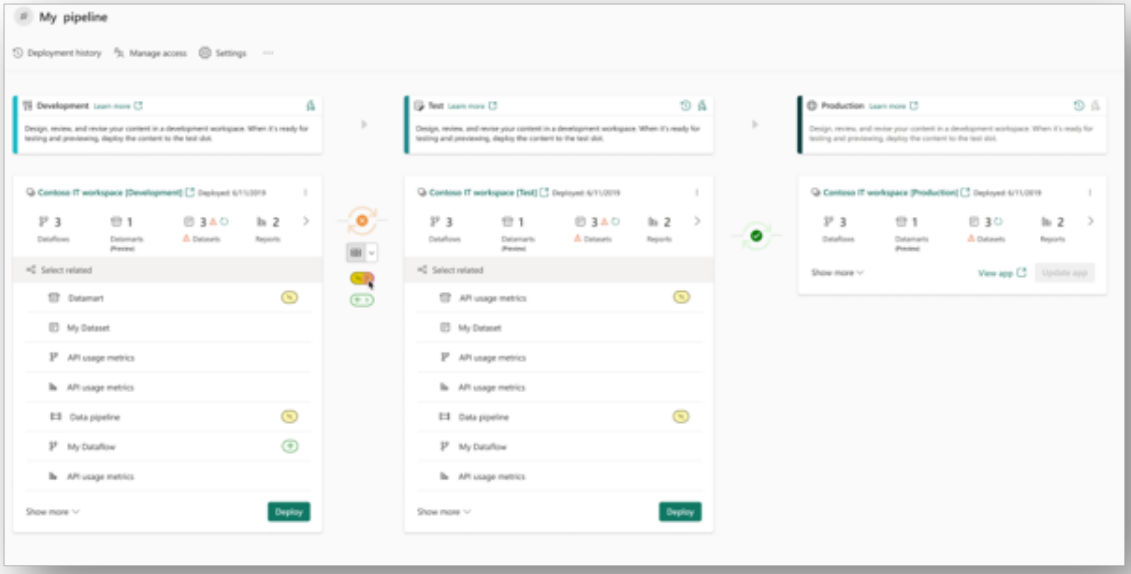
- Design and run unittests

Apply the capabilities of familiar source control tools to manage Fabric items

The integration with source control is at a workspace level. Developers can version items they develop within a workspace in a single process, with full control and visibility across all their artifacts.

Fabric GitOps Platform

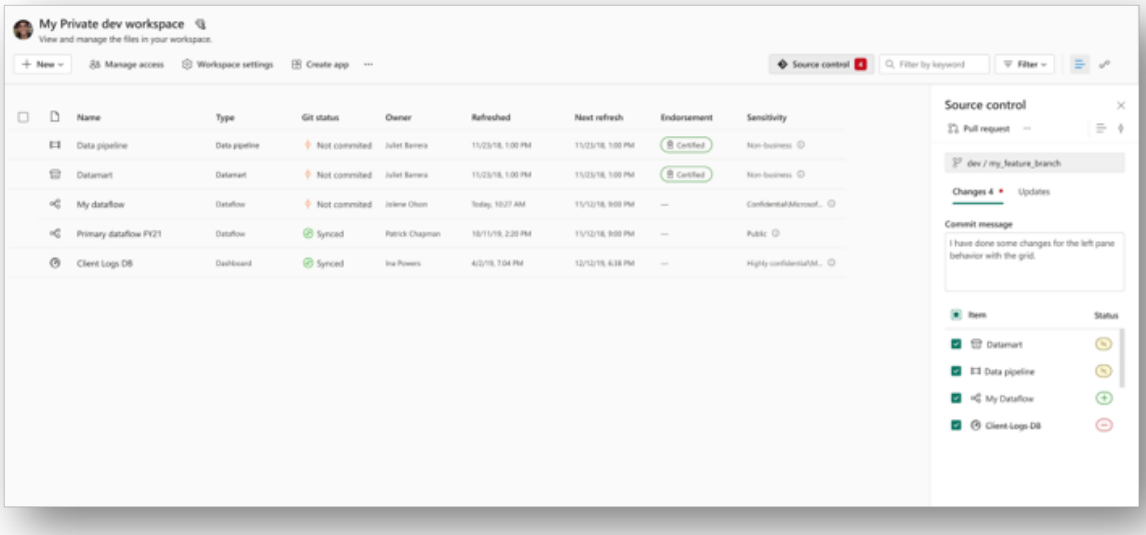
Deployment Pipelines



Microsoft Fabric provides deployment pipelines that support:

- Git integration, allowing synchronization of a workspace to a Git branch and back from inside the Fabric Ui or through the Fabric Git Rest Api.
- Deployment of items across workspaces with configurable rules.
- Comparison of changes at the code level.
- Automation of pipeline creation and deployments using GitHub Actions and workflows.

Git Integration



Key concepts of Git integration in Fabric include:

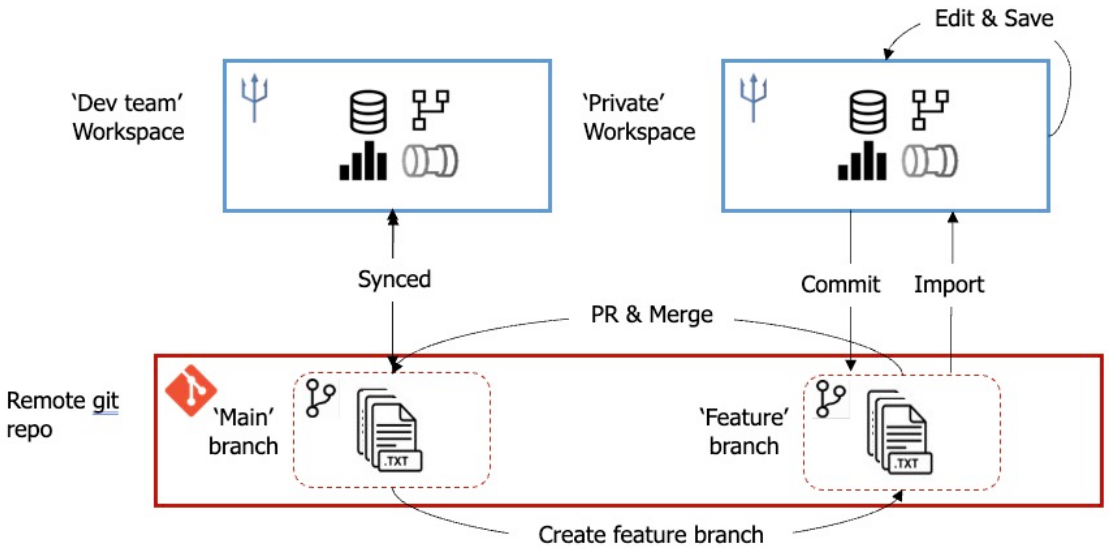
- Workspaces in Fabric act as lightweight containers for items and shared authoring/consumption areas based on permissions.
- Each workspace can be connected to a single Git branch at a time.
- Authoring can be done separately in Fabric and Integrated Development Environments (IDEs) like Visual Studio Code and others.

In the CNS solution we will use Workspaces as the concept for tenant management and separation of concerns. Each Workspace inside Fabric can have independent relations to a Git branch enabling things like canary deployments.

Two alternatives for developing and maintaining the code in CNS

There are 2 options for authoring workflows for. Fabric, through the Fabric UI or through an IDE.

Authoring through Fabric workspace UI



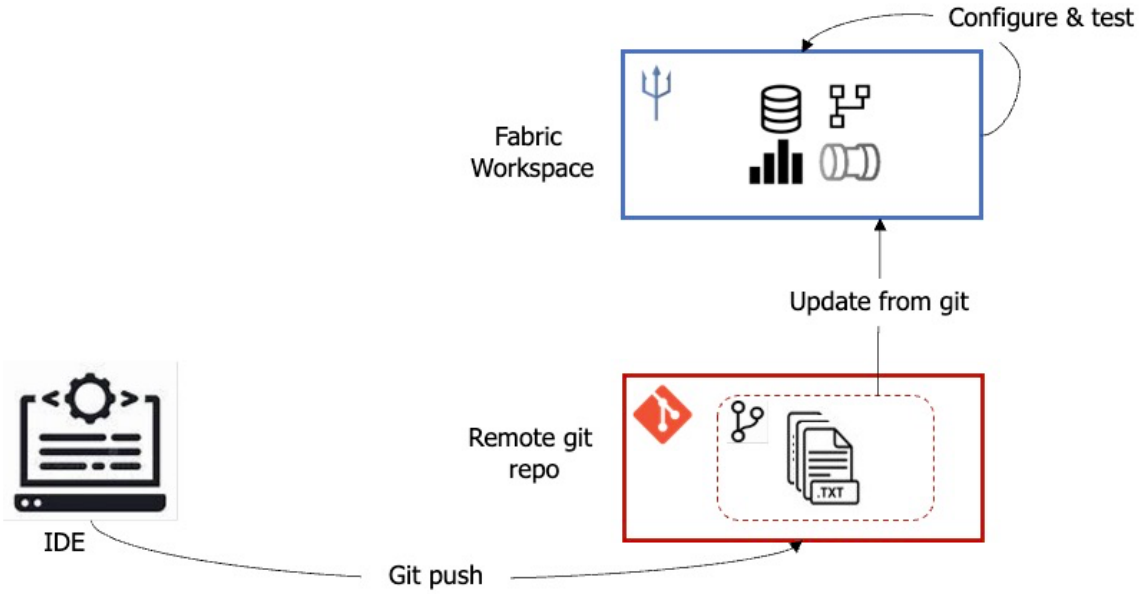
Maintaining Fabric in the Fabric UI involves saving the changes in the UI to the main branch of a Git repository, which acts as the source of truth for all updates. To ensure that these updates are effectively distributed across different workspaces, one can follow a systematic approach.

First, any modifications made within the Fabric UI should be committed to the main branch. This ensures that the latest changes are centralized and can be accessed by all relevant parties. Once the changes are committed to the main branch, a feature branch can be created or updated. This feature branch will act as a temporary holder for the new updates, allowing for thorough testing and validation (including automatic canary deployments of specific branches if so desired) before the changes are propagated to other environments.

The next step involves merging the changes from the main branch into the feature branch. This can be done by checking out the feature branch and pulling the latest updates from the main branch. By doing this, the feature branch will incorporate the latest changes, ensuring that it is up to date.

After merging, the updates in the feature branch can be pushed to other workspaces as needed. This is achieved through deployment pipelines, which automate the process of distributing changes from the development environment to other environments, such as UAT and Production (what Microsoft usually refers to a “ring” model, taking newer code through several rings until it reaches production customers). The deployment pipelines can validate the updates, retain connections, and perform an audit to ensure everything is in order.

Authoring through IDE



When working with Microsoft Fabric artifacts through an Integrated Development Environment (IDE) such as Visual Studio Code, developers can follow a systematic approach to ensure that their changes are efficiently propagated to consuming workspaces.

Developing Microsoft Fabric Artifacts in an IDE

1. Setting up the Environment:

- Begin by cloning the Git repository associated with the Fabric workspace into your local development environment. This ensures you have access to the latest version of the codebase and configurations.

- Open the cloned repository in your preferred IDE, such as Visual Studio Code, which offers robust support for Git operations and support for Fabric artifact development

2. Authoring Changes:

- Develop and modify the Fabric artifacts within the IDE. This can include creating new datasets, reports, or other necessary components.

- Utilize the IDE's extensions and plugins tailored for Microsoft Fabric development to enhance productivity, such as syntax highlighting, code completion, debugging tools and AI code generation for boilerplate or unit testing.

3. Committing Changes to Git:

- After making the necessary modifications, stage the changes and commit them with a descriptive message that outlines the updates. This practice maintains a clear history of changes for future reference and collaboration.

- Push the committed changes to the remote main branch of the Git repository. This centralizes the updates, making them accessible to all relevant team members and environments.

4. Creating and Merging Feature Branches:

- To ensure the stability and integrity of the main branch, create a feature branch from the main branch. This feature branch will serve as a workspace for testing and validating the new updates.

- Merge the changes from the main branch into the feature branch, ensuring that it incorporates the latest updates. This step is crucial for maintaining consistency across all branches.

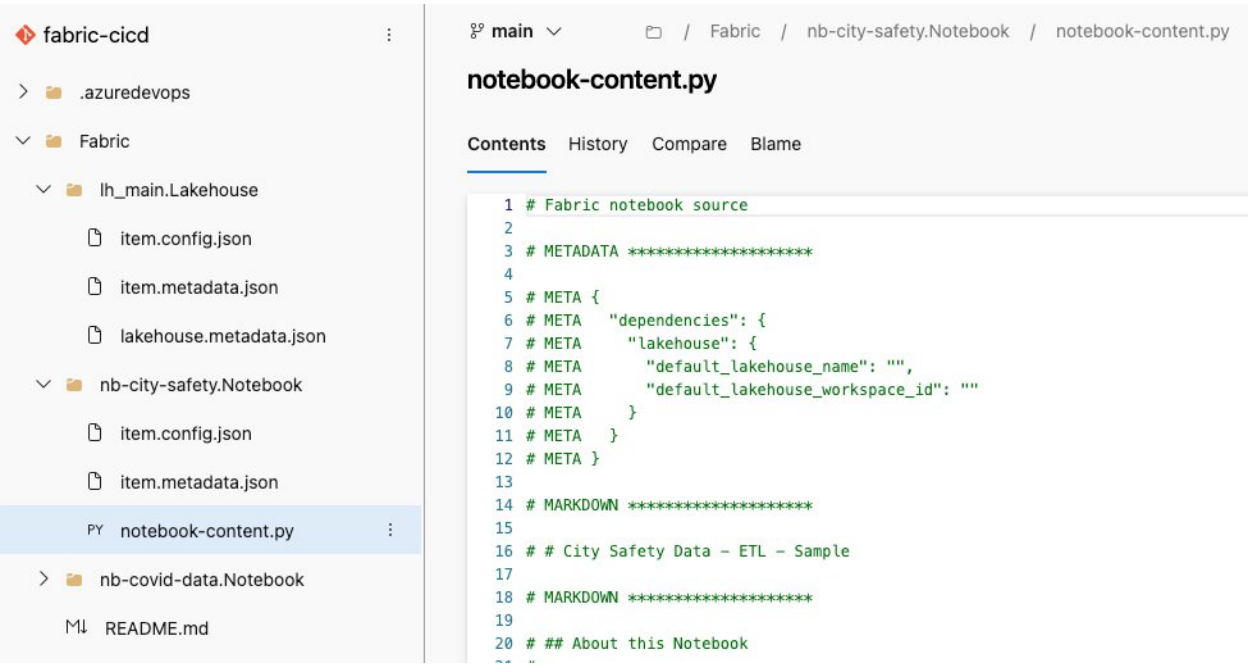
5. Pushing Updates to Consuming Workspaces:

- Once the changes in the feature branch have been thoroughly tested and validated, they can be pushed to other workspaces as needed. This is achieved through deployment pipelines that automate the distribution process.

- The deployment pipelines will validate the updates, retain connections, and perform an audit to ensure that everything is in order. This automation minimizes manual intervention and potential errors.

By following these steps, developers can seamlessly use an IDE to develop Microsoft Fabric artifacts and push them to the Git main branch, allowing for efficient propagation of changes to consuming workspaces.

Microsoft Fabric Git structure



Workspace Organization

A Microsoft Fabric Workspace is designed to facilitate collaboration among data engineers, data scientists, analysts, and business users. It provides a structured environment where you can:

- Manage Artifacts: Store and organize various data artifacts such as datasets, reports, notebooks, and pipelines.
- Deploy items across Workspaces and apply rules on configurations
- Control Access: Define user roles and permissions to secure your data and resources.
- Collaborate: Enable team members to work together on shared projects and resources.
- Monitor and Govern: Keep track of activities, usage, and compliance within the workspace.

Supported Artifacts in Microsoft Fabric Workspace

Within a Microsoft Fabric Workspace, you can create and manage a variety of artifacts that cater to different aspects of data analytics and processing. The following are the supported artifacts available

1. Data Pipelines

Orchestrate and automate data workflows using a visual interface. Data pipelines allow you to integrate data from various sources, perform transformations, and load it into destinations.

Example use cases: ETL/ELT processes, data migration, and workflow automation.

2. Lakehouse

Combines the benefits of data lakes and data warehouses into a single platform. Lakehouse provides a unified storage layer that supports both structured and unstructured data

Example use cases: Big data analytics, machine learning, and real-time data processing.

3. Notebooks

Interactive computational documents that allow you to write and execute code in languages like Python, Spark SQL, or Scala. Notebooks are ideal for data exploration, visualization, and experimentation.

Example use cases: Data analysis, prototyping models, and collaborative research. In the CNS deployment we will be using Notebooks for parsing of the incoming XML files.

4. Paginated Reports

Pixel-perfect reports optimized for printing or PDF export. Paginated reports display all the data in a table, regardless of its length, across multiple pages.

Example use cases: Operational reporting, invoices, and regulatory documents.

5. Reports

Interactive reports created using Power BI tools, enabling rich visualizations and data exploration.

Example use cases: Business intelligence dashboards, KPI monitoring, and ad-hoc analysis.

6. Semantic Models

Data models that define the relationships, calculations, and structures of your data, enabling consistent and reusable data definitions. Excludes support for push datasets, live connections to Analysis Services, and model v1.

Example use cases: Data standardization, complex calculations, and data governance.

7. Spark Job Definitions

Predefined configurations for running Spark jobs without the need to write extensive code. Spark job definitions simplify the process of submitting and managing Spark workloads.

Example use cases: Batch processing, big data transformations, and scheduled analytics tasks. Ther is an option to use Spark jobs instead of Notebooks to define and work with the parsing of the incoming files in CNS.

8. Spark Environment

Provides a managed Spark cluster environment within the workspace, enabling scalable data processing and analytics using Apache Spark.

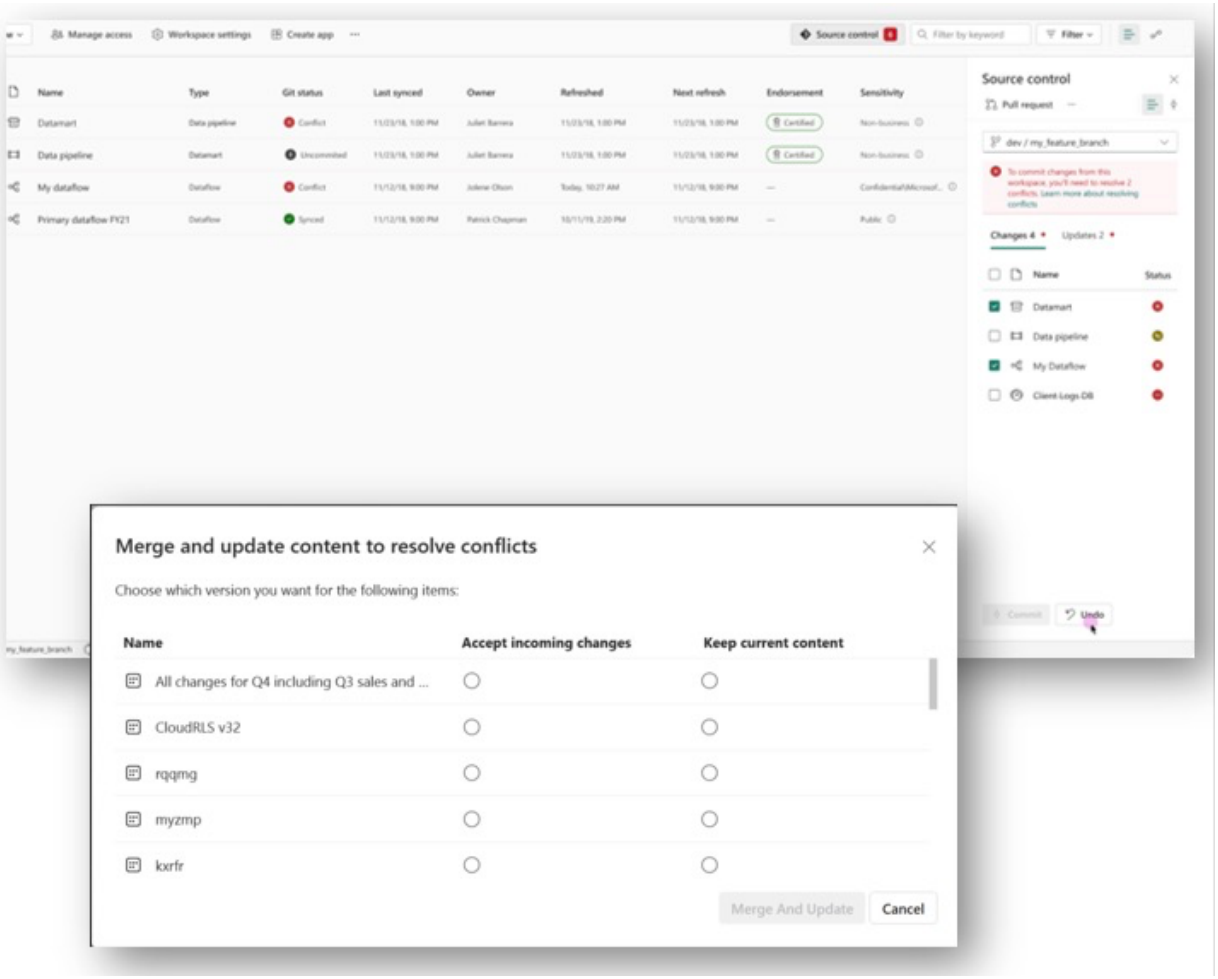
Example use cases: Distributed computing, machine learning algorithms, and real-time analytics.

9. Warehouses

Fully managed SQL-based data warehouses optimized for large-scale analytics. Warehouses in Microsoft Fabric offer high performance and concurrency for analytical queries.

Example use cases: Enterprise data warehousing, complex query processing, and large-scale reporting.

Managing Git Conflicts in Microsoft Fabric



When managing Git version conflicts in the Fabric UI, any changes made to the same item will be clearly marked as ‘Conflict’. Resolving these conflicts efficiently is crucial to maintaining a smooth workflow. Users have several options to address these conflicts:

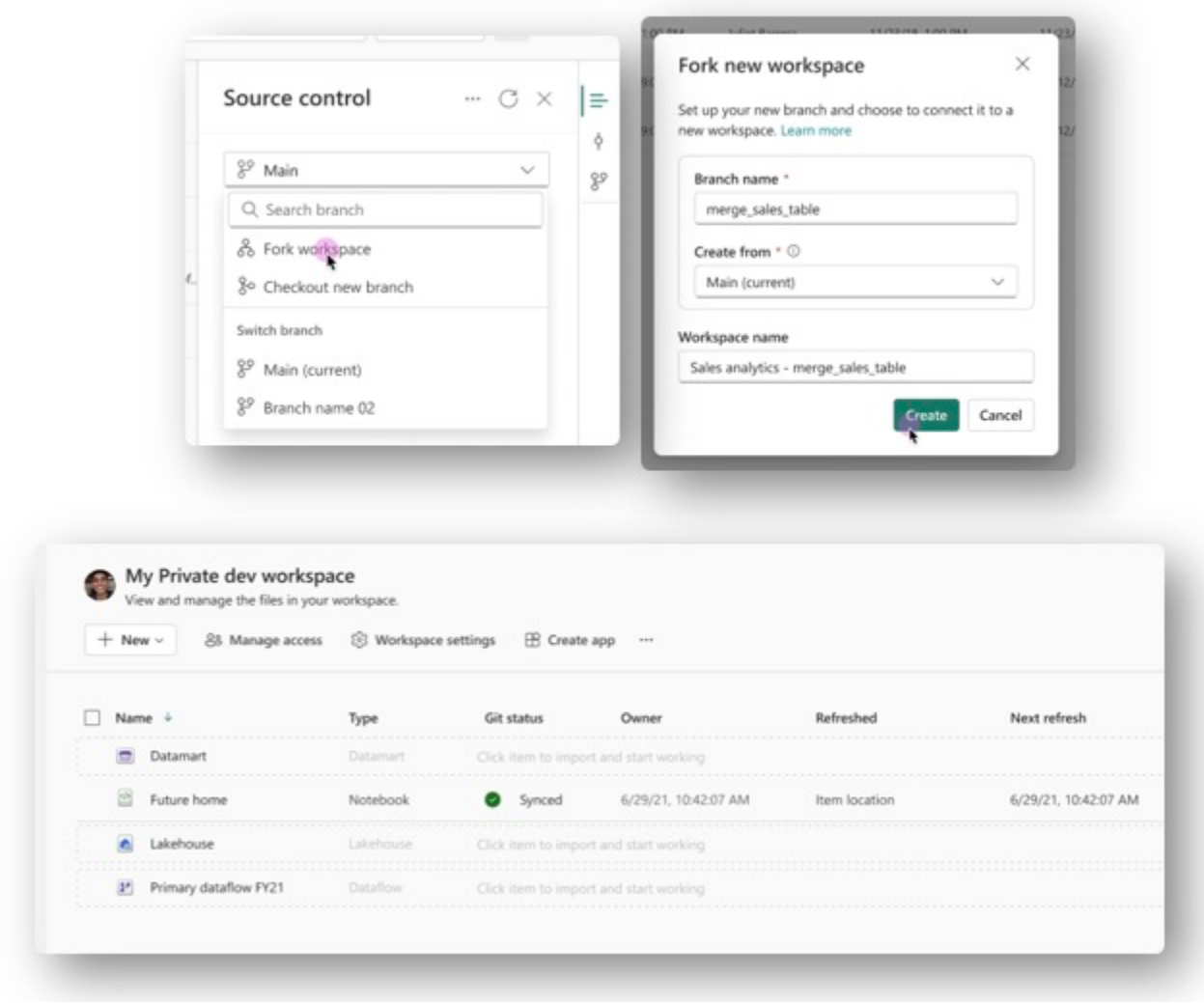
1. ‘Undo’ + ‘Update’: This method allows users to revert changes to a previous state, effectively removing the conflict. Once undone, the user can then update the item to ensure it reflects the latest, conflict-free version.

2. Resolve in the UI: Users can resolve conflicts directly within the Fabric UI by selecting which content they prefer to keep. This allows for a straightforward resolution process without needing to leave the interface.

3. ‘Checkout branch’ + resolve in GitHub: For those who prefer to handle conflicts using Git, users can checkout the relevant branch and resolve the conflicts using the tools available in GitHub.

Each method provides a flexible approach to conflict resolution, ensuring that users can select the option that best fits their workflow and preferences.

Mapping Fabric workspaces to Git Branches



Mapping Fabric workspaces to Git Branches involves a streamlined and efficient process that allows users to 'branch out' to new feature workspaces and branches for specific tasks effectively. This system supports:

1. Easy ‘Branching Out’: Users can effortlessly create new feature workspaces and corresponding Git branches for specific functionalities, promoting a structured and organized development environment.

2. Seamless Navigation: The integration facilitates smooth transitions between related workspaces, all while maintaining shared connections and distinct branches. This ensures that the movement between different parts of the project remains intuitive and fluid.

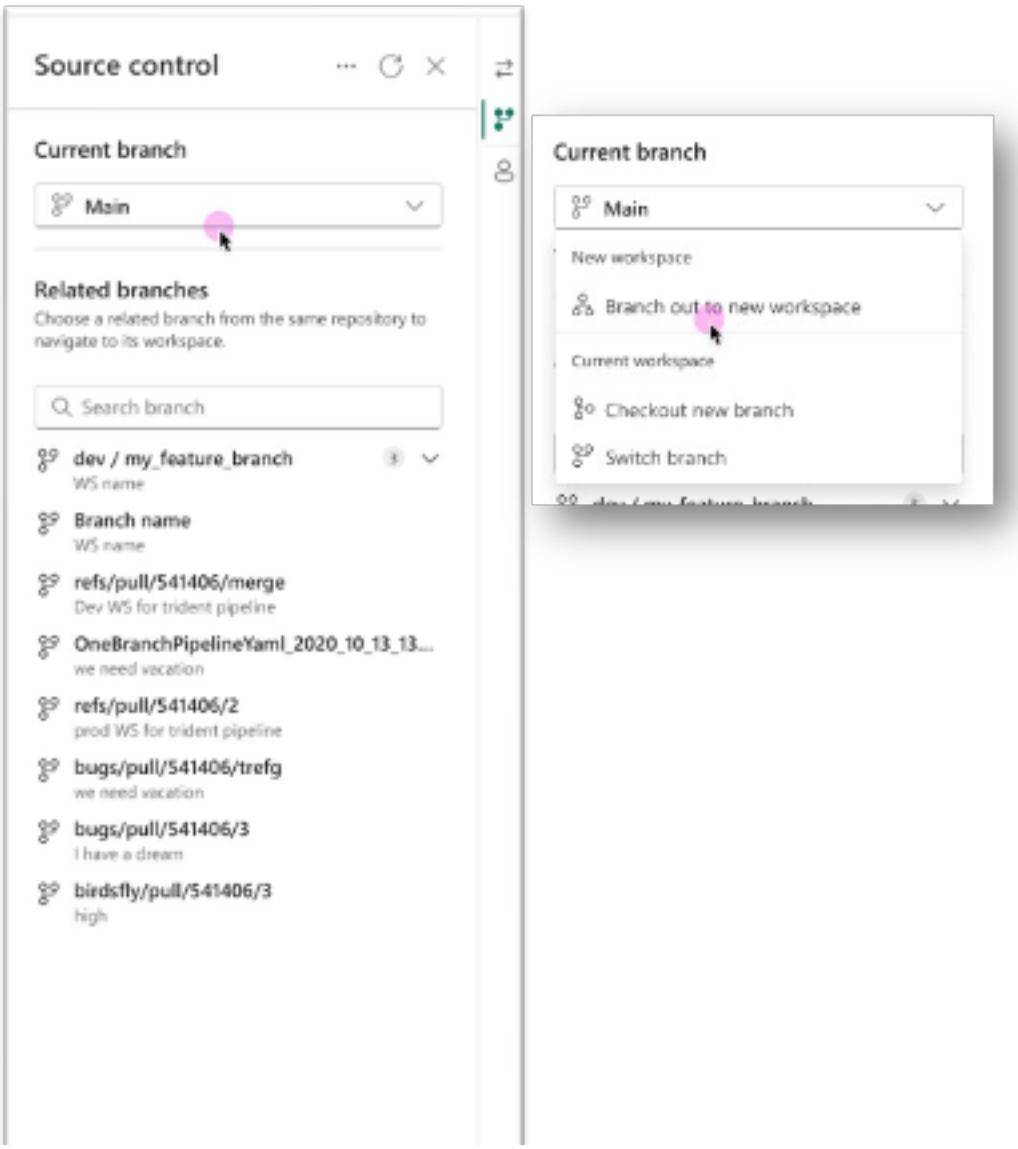
3. Minimizing Friction: The creation of new workspaces and branches is designed to be frictionless, reducing any potential barriers that might hinder the development process. This efficiency helps in quick setup and immediate commencement of work.

4. Reducing Setup Time: The time required to start working in a new environment is minimized, allowing developers to focus more on their tasks rather than the setup process. This leads to increased productivity and faster project progress.

5. Managing Workspaces and Branches: Integrated tools are provided to manage the workspaces and branches created with Fabric’s Git integration. These tools offer comprehensive management capabilities, making it easier for users to handle various aspects of their project within the Fabric environment.

By adopting these practices, teams can ensure a smooth and efficient workflow, leveraging Git branches and Fabric workspaces to maximize their development potential.

Contextual switching workspace connection to a Git branch



Contextual navigation between workspaces and branches is a critical aspect of maintaining an agile and efficient development environment. The 'Branches' tab plays a pivotal role in this process by providing a centralized interface to manage and navigate between different Git branches and their corresponding Fabric workspaces. This functionality is designed to enhance the developer experience by offering:

Branch Overview

The 'Branches' tab offers a comprehensive overview of all active branches within the project. This includes details such as branch names, commit histories, and the status of each branch, allowing users to quickly identify the branch they need to work on.

Workspace Synchronization

Users can easily link their current workspace to a specific Git branch using the 'Branches' tab. This ensures that all changes made within the workspace are reflected in the corresponding branch, maintaining consistency across the project.

Branch Switching

The 'Branches' tab simplifies the process of switching between branches. With just a few clicks, users can transition their workspace to a different branch, enabling them to work on multiple features or bug fixes simultaneously without losing context.

Conflict Resolution

The 'Branches' tab also provides tools for resolving merge conflicts that may arise during development. Users can view conflicts, make necessary adjustments, and merge changes directly from this interface, streamlining the conflict resolution process.

Branch Creation and Deletion

Creating new branches for feature development or deleting obsolete branches is made straightforward through the 'Branches' tab. This capability encourages a clean and organized branch structure, facilitating better project management.

Branch Insights

The 'Branches' tab includes analytics and insights into branch activity, such as the number of commits, contributors, and the time since the last update. These insights help teams monitor branch usage and identify any branches that may require attention.

By leveraging the 'Branches' tab, development teams can navigate their projects more effectively, ensuring smooth transitions between different parts of the codebase and

maintaining an organized and efficient workflow. This centralized approach to branch management not only improves productivity but also enhances collaboration among team members.

Leverage Fabric API for Git Automation

To further streamline development workflows, Fabric exposes a set of public APIs designed to automate Git operations, enabling users to integrate GitOps practices seamlessly into their CI/CD pipelines. This comprehensive API suite includes endpoints for managing connections, synchronization, and status checks, thus providing developers with the tools they need to automate and monitor their workflows effectively.

Connection Public APIs

These APIs allow users to connect or disconnect a workspace to or from a remote branch. Automating these operations ensures that developers can manage their repository connections without manual intervention, facilitating a more efficient workflow.

Sync Public APIs

The sync APIs enable users to commit changes from their workspace to the remote branch and update their workspace with changes from the remote branch. This bidirectional synchronization is crucial for maintaining consistency and ensuring that all team members are working with the latest codebase.

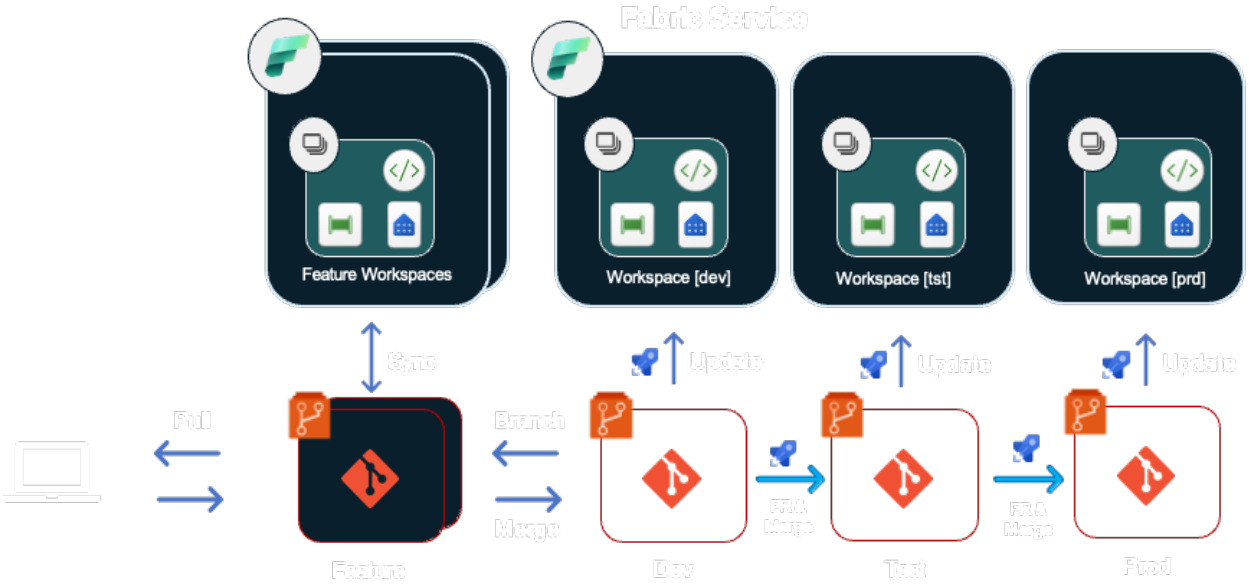
Status Public APIs

To keep track of the state of their workspaces, developers can use status APIs to check the connection status of a workspace and the Git status of all supported items within it. This visibility is essential for maintaining oversight of development activities and ensuring that any issues are promptly identified and addressed.

By leveraging these Fabric APIs, development teams can automate their GitOps operations, reducing manual effort and enhancing the reliability and speed of their deployments.

Scenarios for GitOPs

Scenario 1: GitOps based deployments from branches



Scenario 1: Git-based Deployments Using Fabric API

In this scenario, Git serves as the 'single source of truth', ensuring that all deployments originate from the repository, thus maintaining consistency and traceability throughout the development lifecycle. Deployments are streamlined by using primary branches for each stage of the CI/CD pipeline, such as development (DEV), user acceptance testing (UAT), and production.

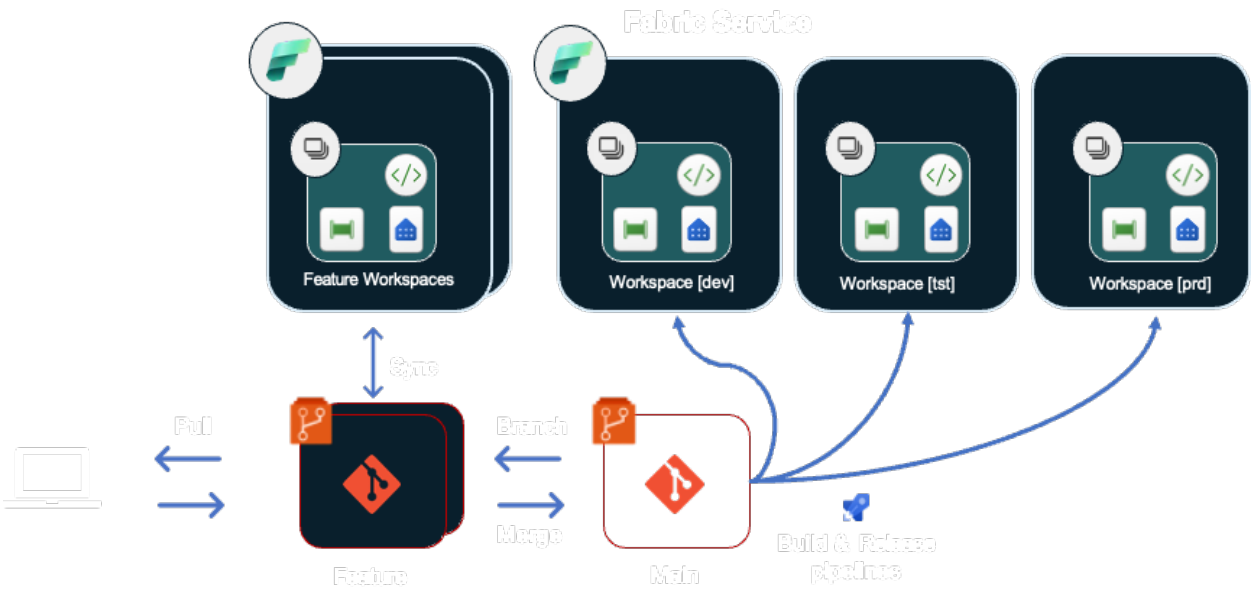
Fabric Git APIs play a pivotal role in updating workspaces at each stage. These APIs allow for seamless synchronization and status checks, ensuring that each workspace reflects the current state of the codebase. This integration reduces manual intervention, thereby minimizing errors and enhancing the reliability of the deployment process.

Key steps include:

- Mapping the development environment to the Git repository.
- Using Fabric Git APIs to sync changes between the workspace and the repository.
- Applying automated deployment pipelines to transition code from DEV to UAT and finally to Production.

By leveraging Fabric Git APIs, development teams can automate and monitor their workflows effectively, ensuring a smooth and efficient deployment process.

Scenario 2: Git & Build Environments



Workflow begins with the creation of a feature branch. This allows developers to work on new features or bug fixes in isolation without affecting the main development branch. Once the development in the feature branch is complete and has passed all necessary tests, it is merged into the dev branch. Here's a detailed breakdown of the process:

Step 1: Create a Feature Branch

- Developers create a feature branch from the main branch to implement new features or fixes.
- All changes are committed to this branch, allowing for isolated development.

Step 2: Push Feature Branch to Dev Branch

- Once development is complete, the feature branch is subject to code reviews and automated tests.
- After passing the reviews and tests, the feature branch is merged into the dev branch.

Step 3: Continuous Integration in Dev Branch

- The dev branch is connected to the CI/CD pipeline, triggering automated builds and tests.
- Using Fabric Git APIs, the updates are synchronized with the development environment, ensuring the workspace reflects the latest state of the codebase.

Step 4: Push to Prod Branch

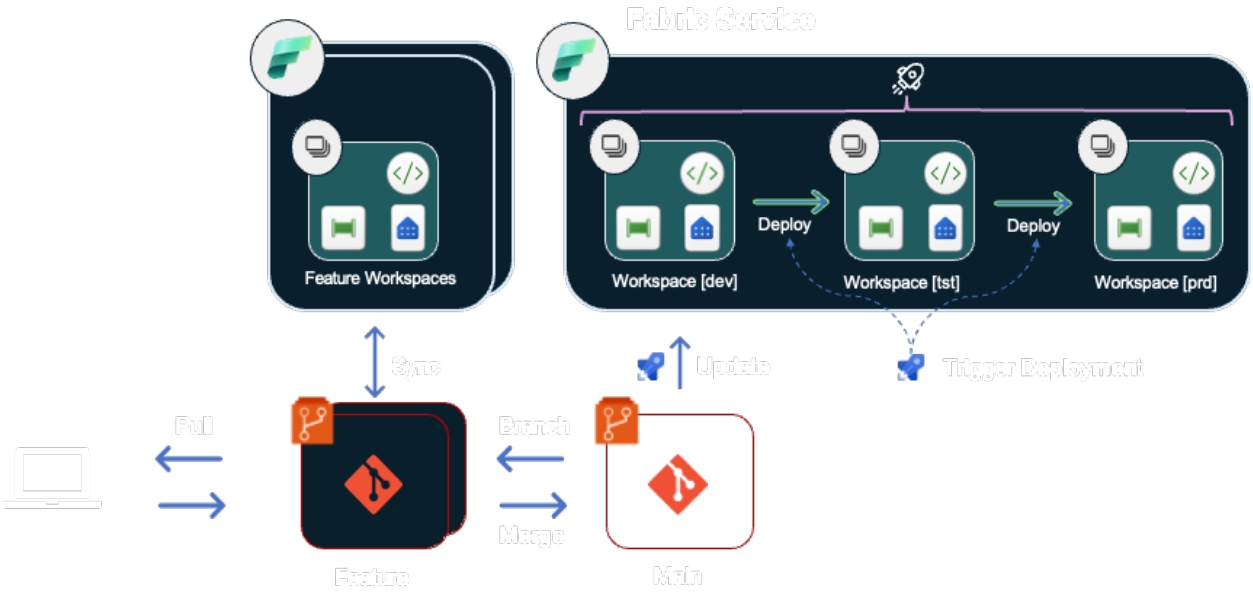
- After thorough testing in the dev branch, the code is ready for production.

- The final step involves merging the dev branch into the prod branch.

Step 5: Deployment from Prod Branch

- Code is automatically deployed from the prod branch using deployment pipelines.
- This ensures a reliable and consistent release process, minimizing manual intervention and errors.

Scenario 3: Git & Fabric pipelines



In this deployment scenario, development begins in a feature branch, where individual developers work on specific tasks or enhancements. Once the feature branch passes reviews and tests, it is merged into the dev branch.

Step 3: Continuous Integration in Dev Branch

The dev branch is integrated with the CI/CD pipeline, which triggers automated builds and tests. Using Fabric Git APIs, these updates synchronize with the development environment, ensuring the workspace is always in sync with the latest codebase changes.

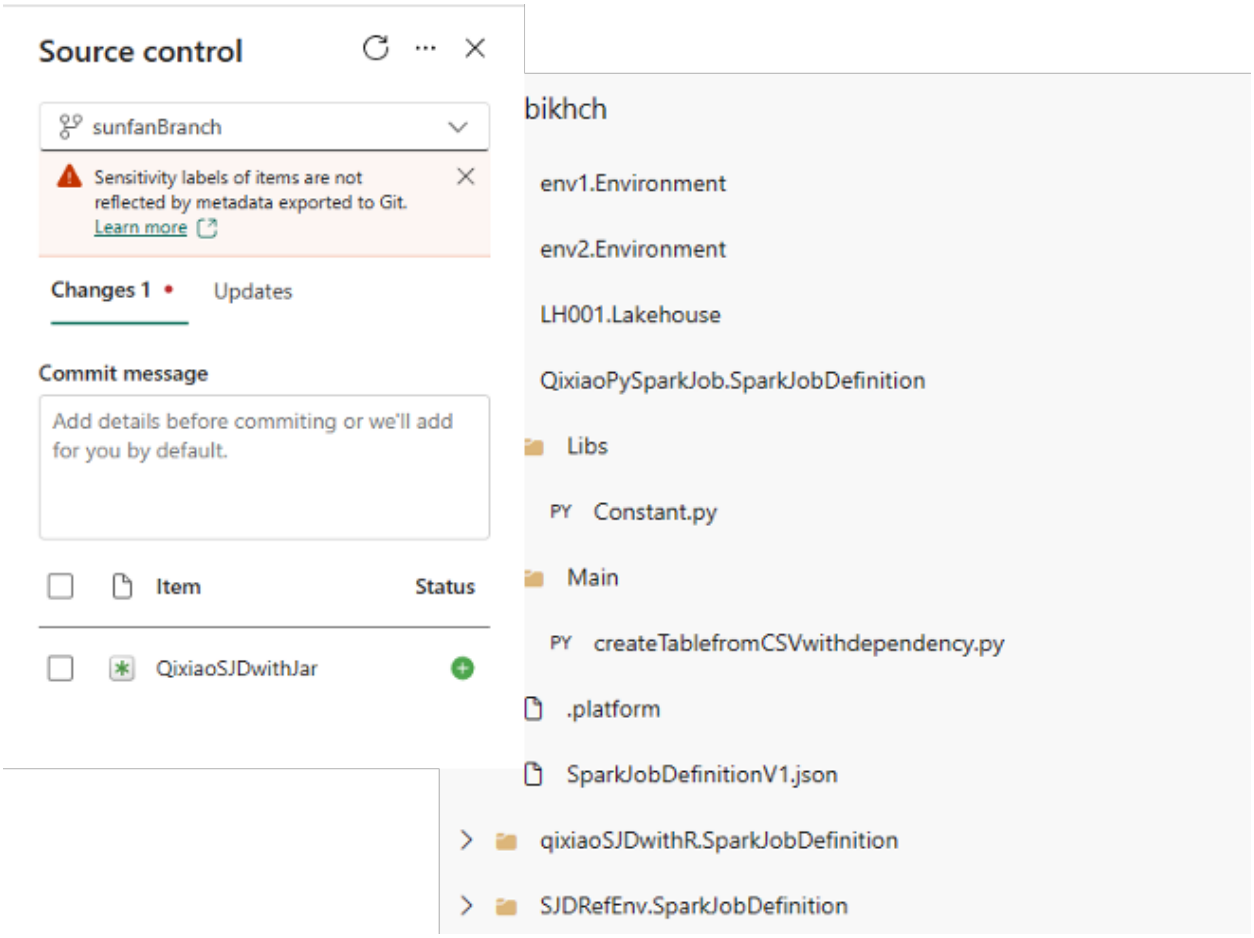
Step 4: Push to Prod Branch

Following rigorous testing in the dev branch, the code is deemed ready for production. The subsequent step involves merging the dev branch into the prod branch, preparing it for deployment.

Step 5: Deployment from Prod Branch

Deployment pipelines automatically deploy the code from the prod branch, ensuring a reliable and consistent release process that minimizes manual intervention and related errors.

Scenario 4: Managing Spark Jobs from Git

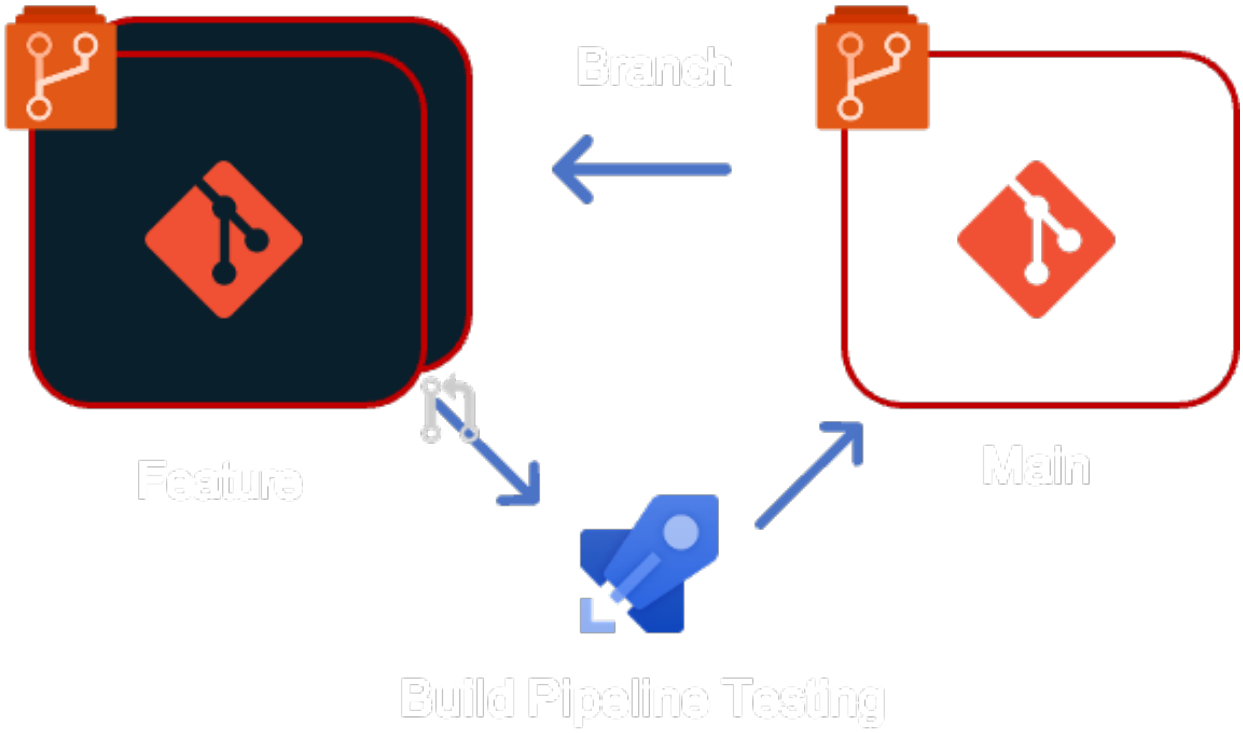


Managing Spark jobs efficiently involves integrating your code repository with your workspace settings. From your workspace settings, you can easily set up a connection to your repo to commit and sync changes.

This integration allows for seamless management of Spark job definitions and configurations directly from the Git repository, ensuring version control and collaborative development practices are maintained.

Code commits versioning etc would be managed as described in the scenarios above.

Unit test



We provide a standalone test framework for Microsoft Fabric

- Evaluate expressions by using the framework's internal expression parser. It supports all the functions and arguments that are available in the Data Factory expression language.
- Test an activity with a specific state and assert the evaluated expressions.
- Test a pipeline run by verifying the execution flow of activities for specific input parameters and assert the evaluated expressions of each activity.

https://github.com/microsoft/data-factory-testing-framework/tree/main

Multi-tenant Architecture

ISVs often face challenges in managing data for multiple tenants in a secure manner while keeping costs low. Traditional solutions may prove costly for scenarios with more than 100 tenants, especially with the common ISV scenario where the volume of trial and free tenants is much larger than the volume of paying tenants.

The motivation for ISVs to use Fabric is that it brings together experiences such as Data Engineering, Data Factory, Data Science, Data Warehouse, Real-Time Analytics, and Power BI onto a shared SaaS foundation.

We will explore the Workspace per tenant-based architecture, which is a cost-effective solution for managing data for all tenants in Microsoft Fabric, including ETL and reporting.

Multi-Tenant Data Solutions

The possibility of querying data from multiple tenants is quite common even though it is against the isolated concept. Fabric implementation concepts will be discussed here.

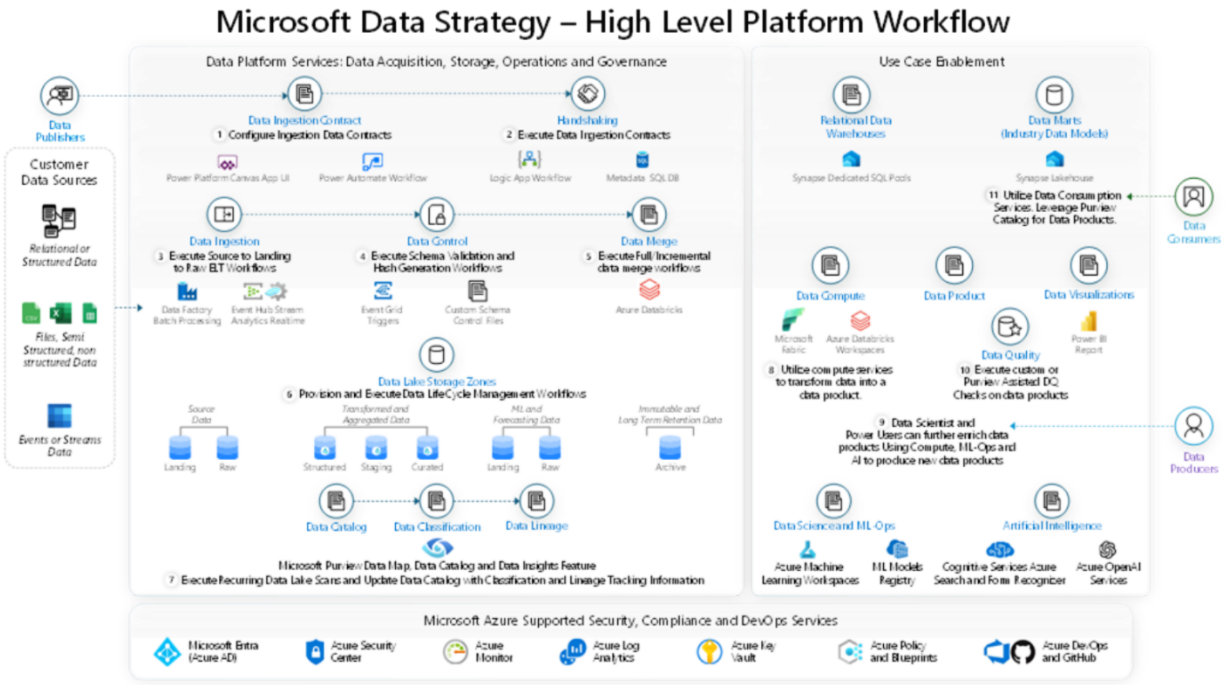
Application scenarios:

Microsoft Fabric is designed for multiple application scenarios. Typical ISVs projects that need to ensure that the architecture will support:

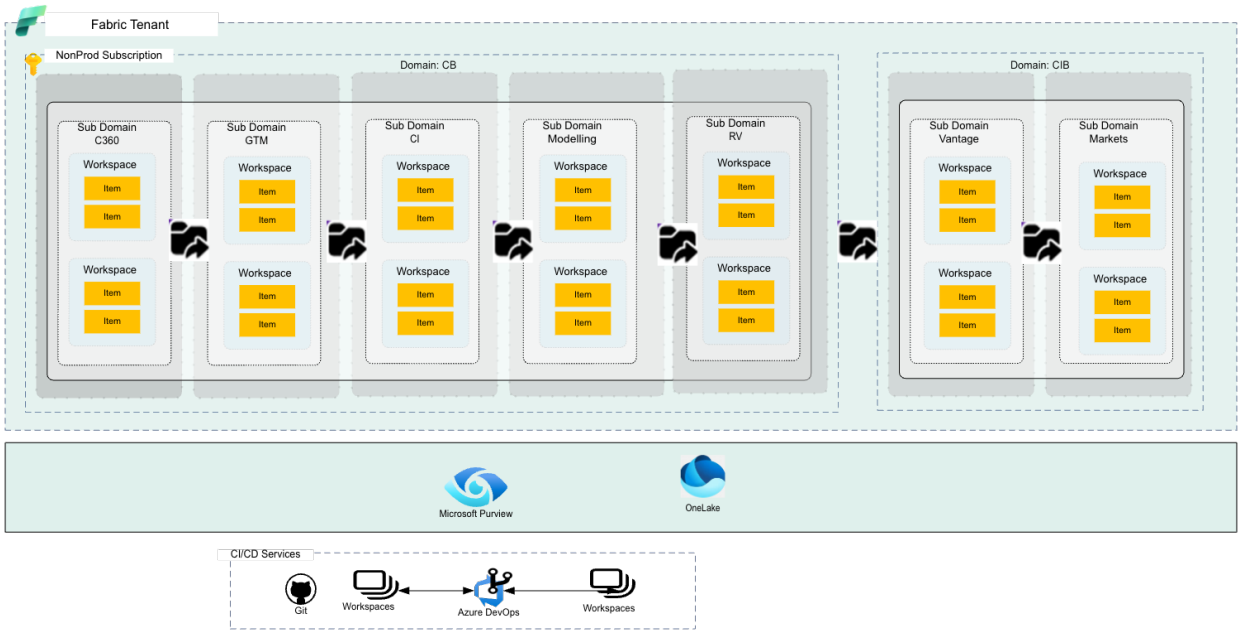
- Multi-tenants that need data isolation between different tenants.
- Power BI reporting.
- Performance and cost challenges with relational engine.
- Easy migration to Fabric.

The Workspace approach is well-suited to support all those scenarios.

Platform workflow:



Workspace based multi-tenant architecture



One of the ground building blocks of Fabric is a workspace. Workspaces are containers that are places to collaborate with colleagues to create collections of items such as lakehouses, warehouses, and reports. You can grant permission per workspace (see the

security part later), which can be extremely helpful to associate the tenant's login with the tenant's workspace and to his workspace only.

Tenant Data Isolation

Due to business, regulation and security considerations, any multi-tenant should ensure that each tenant can access only his data. From a high-level perspective, the solutions that enable us to achieve this granularity are divided into two types:

- Physical separation of the data to separate locations.
- Ensuring that the application will filter the data from the relevant tenants by mechanisms like Row Level Security.

This document discusses the physical separation type only since this type is aligned with Fabric's architecture.

Shared data

For shared data the suggested usage is to have a separate workspace that will be shared by a shortcut to all the tenants' environments. If the shared data is managed by a database, you might be able to use mirroring to sync the database to the shared data workspace.

Features of Fabric that support multitenancy

Capacities and multi-Region

In Fabric you will have only OneLake per the ISV's tenant. However, you can deploy your system in multiple regions by having capacity defined in each region.

- There is only one OneLake per tenant.
- A tenant can have multiple capacities in one or more regions.
- Any capacity is attached to a specific region.
- A workspace can be assigned to only one capacity.
- Every item stored in a lakehouse/warehouse of a workspace will be stored in the region of the tied capacity.

Multi-tenancy friendly cost structure

Storage

Delta-Parquet files which are the basic building block in Fabric. Those files charged per volume so the number of workspaces will not affect the cost.

Serverless Service

Fabric is a serverless solution which means that there is separation between storage and compute resources payments.

As you expect, you are paying for the storage you are using and you should try to optimize the size of the storage. Since storage costs are low, the storage cost will not be a significant percentage of your total cloud bill.

For compute, you will pay according to the usage. In the BI environment, the user load is expected to vary and such models will save money.

With classic Fabric implementation, you can skip the need for relational database which usually can be one of the main cloud expenses.

ETL

Most ISVs run ETL per tenant, therefore the cost will be the same.

In rare cases where one ETL process can deal with multiple tenants, a single workspace for all tenants might run with less pipelines and save costs.

Power BI

In Power BI, a workspace per tenant is the best practice. Please read the Develop scalable multitenancy applications with Power BI embedding article for deep discussion. From the Power BI perspective, the limitations are based on the largest workspace size (and not on the total workspace size) as defined here.

Capacity and Chargeback Management

The recommended approach for segregating tenants through distinct workspaces facilitates a frequently requested feature: chargeback support. By allocating separate capacities to each tenant’s workspace (or multiple workspaces), monitoring and accessing data regarding each tenant's usage becomes straightforward.
Microsoft Fabric concepts and licensing article provides essential guidance for creating a deployment that enables ISVs to implement chargeback mechanisms. This allows for precise billing of end customers based on their actual consumption, streamlining the process and ensuring transparency in usage and cost allocation.

Workspace Security

Granular permissions per tenant

As written above, you can use permission per workspace to ensure tenant's isolation per workspace. The same mechanism is used to give more granular permissions to specific items inside the users of the tenants (good description can be found here).

Note, the same concept is true for permissions inside a lakehouse or warehouse inside a workspace.

For example, the user Mark-CustomerA might be associated with the CustomerA tenant to see only the data related to his tenant. If you want to give him read access to the Orders data you will define a role named OrdersRead-CustomerA and associate Mark with this role. To define a global role OrdersRead instead is possible but will not be a satisfactory solution.

In Fabric you can give permissions by sharing – see here and here. Detail granular permission discussion is beyond the scope of this document – this document is discussing only the security aspects of the multi-tenant scenario.

Multi-tenants Network security

Secure Access in Microsoft Fabric for Multitenant Needs:

- Fabric Workspace Identity: An automatically managed service principal associated with a Fabric workspace.
- Secure Access: Fabric workspaces with a workspace identity can securely read or write to firewall-enabled Azure Data Lake Storage Gen2 (ADLS) accounts through trusted workspace access for OneLake shortcuts.
- Enablement: Workspace identity can be enabled in any F SKU capacity.
- Multitenant Environment: This feature is particularly useful in a multitenant environment. It allows you to connect each workspace with a specific ADLS, associating a pair of workspaces and ADLS according to the specific customer that owns the data.
- Trusted Workspace Access: Ensures secure access to firewall-enabled ADLS.

Identity Management

- We strongly recommend using different users per tenant and not letting an application-based security mechanism to be the only authorization gate.
- In these days, you can even utilize multitenant organization in Microsoft Entra ID which is in preview. Detail discussion of this option is beyond the scope of this article. Some highlights can be found in multitenant organization scenario and Microsoft Entra ID capabilities.

Cross-workspace

Cross-workspace queries

While the demand to have cross-tenant queries looks like opening the system for a security breach, in real life this demand is quite common.

Here are the typical scenarios:

- ISV level reporting.
- ETL to a data warehouse/data lake.
- Using metadata info and/or external data that is relevant to all tenants without the need to duplicate them to all tenants.

To achieve this ability, you should leverage the SQL analytics endpoint that enables querying of any table and easy sharing.

You will need to create a shortcut pointing to the required databases or tables from the other workspace. More details can be found in Cross workspace sharing and querying.

To avoid potential overriding privacy and regulation policies, you should allow cross-tenant queries only in specific cases. You should design such implementation carefully from both security and architecture aspects.

Cross-workspace Pipelines

Organizations might separate their data into multi workspaces due to internal security reasons (Separating Gold from Silver/Bronze, according to the data sensitivity).

For other ISVs, the need is even more complex. The data (or at least part of it) comes in a multi-tenant stream and this data should be divided into different single-tenant streams with a minimal effort.

Currently, Fabric does not support this functionality but the ability to enable Cross-workspace Pipelines is in the roadmap.

However, you can clone your data pipelines across workspaces by using the “Save as” button (see here). This makes it easier to develop pipelines collaboratively inside Fabric workspaces without having to redesign your pipelines from scratch. Another solution, based on dynamic content is described here.

Using Cross Workspace Pipelines might simplify the ETL code as well as reduce the expected costs. With proper design, the expected running time of the processes will be better.

Sample code available on GitHub

For those looking to get started with Microsoft Fabric Git integration, there is sample code available on GitHub. This example will guide you through the process of setting up your environment, ensuring that your DEV workspace is properly connected to a Git branch, and committing your changes efficiently. https://github.com/Azure-Samples/modern-data-warehouse-dataops/tree/main/single_tech_samples/fabric/feature_engineering_on_fabric

Conclusion

GitOps in Microsoft Fabric provides a robust framework for managing development and deployment processes. By leveraging Git integration, deployment pipelines, and public APIs, teams can ensure efficient and automated workflows, reducing manual intervention and enhancing productivity.

Even more capabilities to manage Fabric using GitOps is coming down the road, please see the roadmap slide submitted as part of the Rfx Answer.

I hope this document helps!