Git Good: Best Practices for CI/CD and Collaboration in Microsoft Fabric

THE RERUN

Peer Grønnerup

Technical Architect & Community Advocate
Tabular Editor, Denmark

WHO AM I?

PEER GRØNNERUP

Technical Architect & Community Advocate at Tabular Editor

+15 years working with BI, Data Platform design and automation
Part of the Fabric Private Preview (Project Trident)
... and all in on Fabric Automation and CI/CD!

https://www.linkedin.com/in/peergroennerup/

https://peerinsights.emono.dk/

Download the showcase code and presentation

https://github.com/gronnerup/FabricAutomation

Disclaimer:
The solution demonstrated in this session is provided for **showcasing purposes only**. It is **unsupported**, and there are no guarantees regarding functionality or future updates. You are free to use it **as-is** or modify it to fit your needs. Use at your own risk.

Where are we going – Git Good in Fabric

Architecture
as our foundation

→

Workspaces, repos
and environments

→

Branching for real-
world collaboration

→

Automation & CI/CD
pipelines

→

Tips, Tricks & Summing
up

✓

- Architecture & platform setup
- Fabric Git integration & repo structure
- Ways-of-working & collaboration
- Branching strategires and release flows
- CI/CD pipeline examples
- fabric-cicd library and Fabric CLI

✗

- Fabric solutions development & items
- How to implement dynamic code
- The Fabric Terraform Provider
- Fabric REST Apis and CLI in details
- Fabric Deployment pipelines
- Security & governance

Let’s Git Good (/gɪt gʊd/) - Why CI/CD & Git matters…

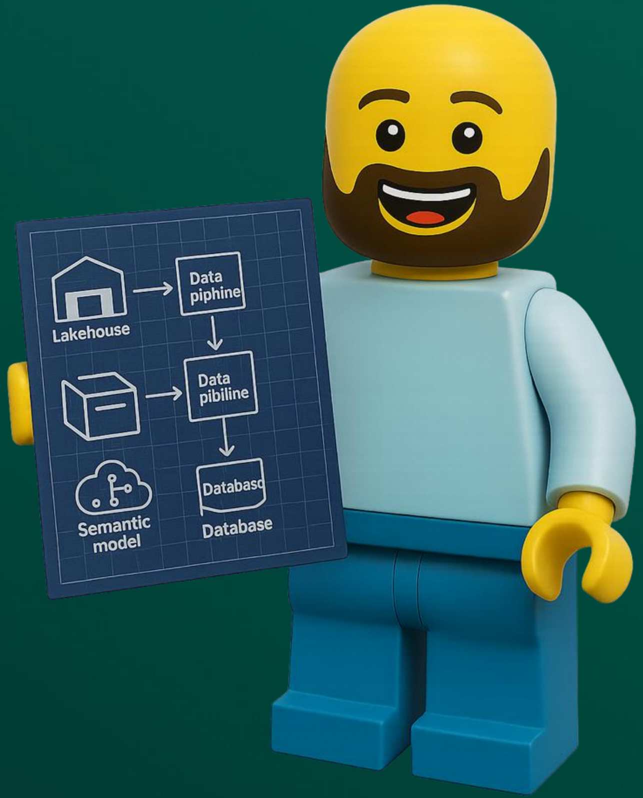
The classic foundations

- Version control
- Change reviews
- Testing before releasing
- Reproducibility
- Collaboration

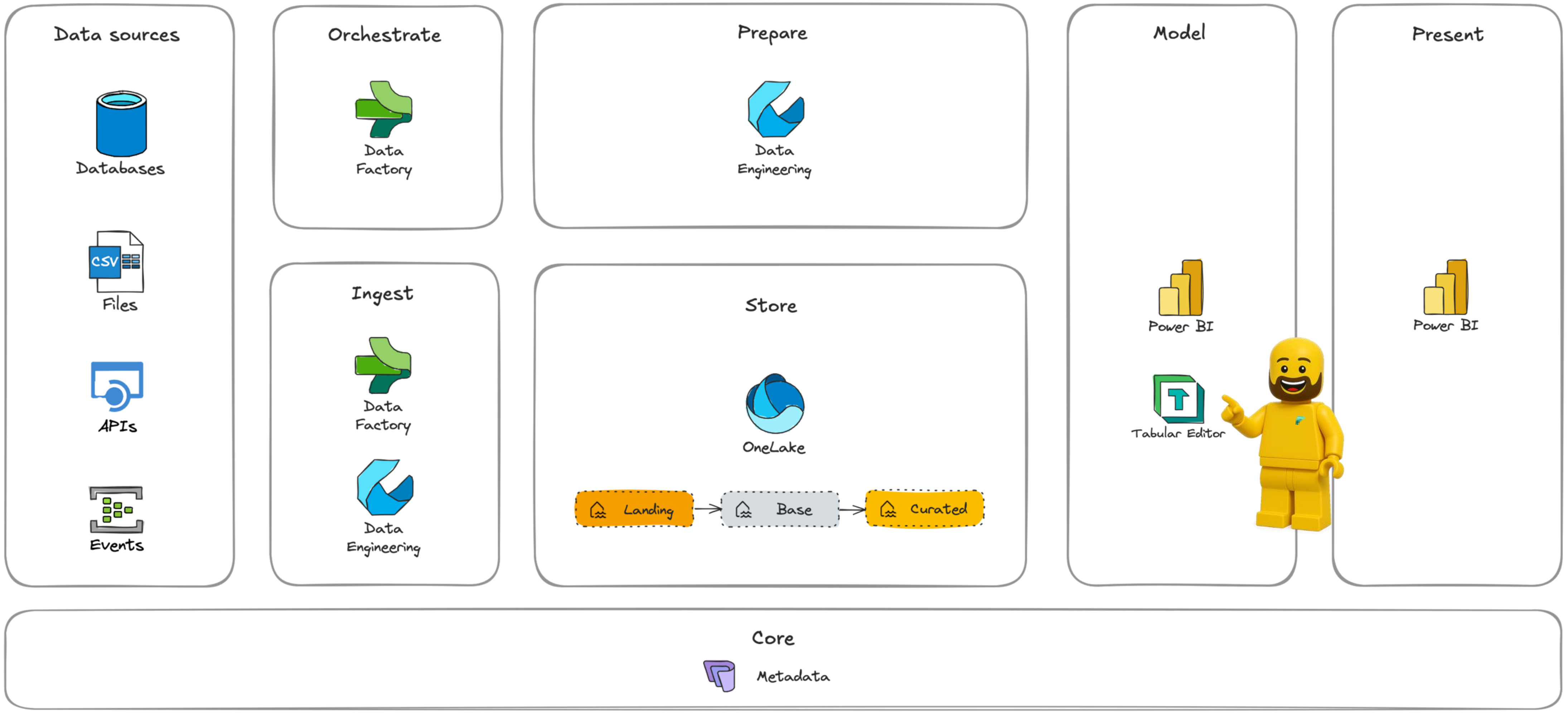
And the Fabric angle…

- Declarative and cloud-native
- Items map to Git folders
- Enables controlled, repeatable deployments
- Use of purpose-built tools
- Collaboration at scale

Architecture as our foundation

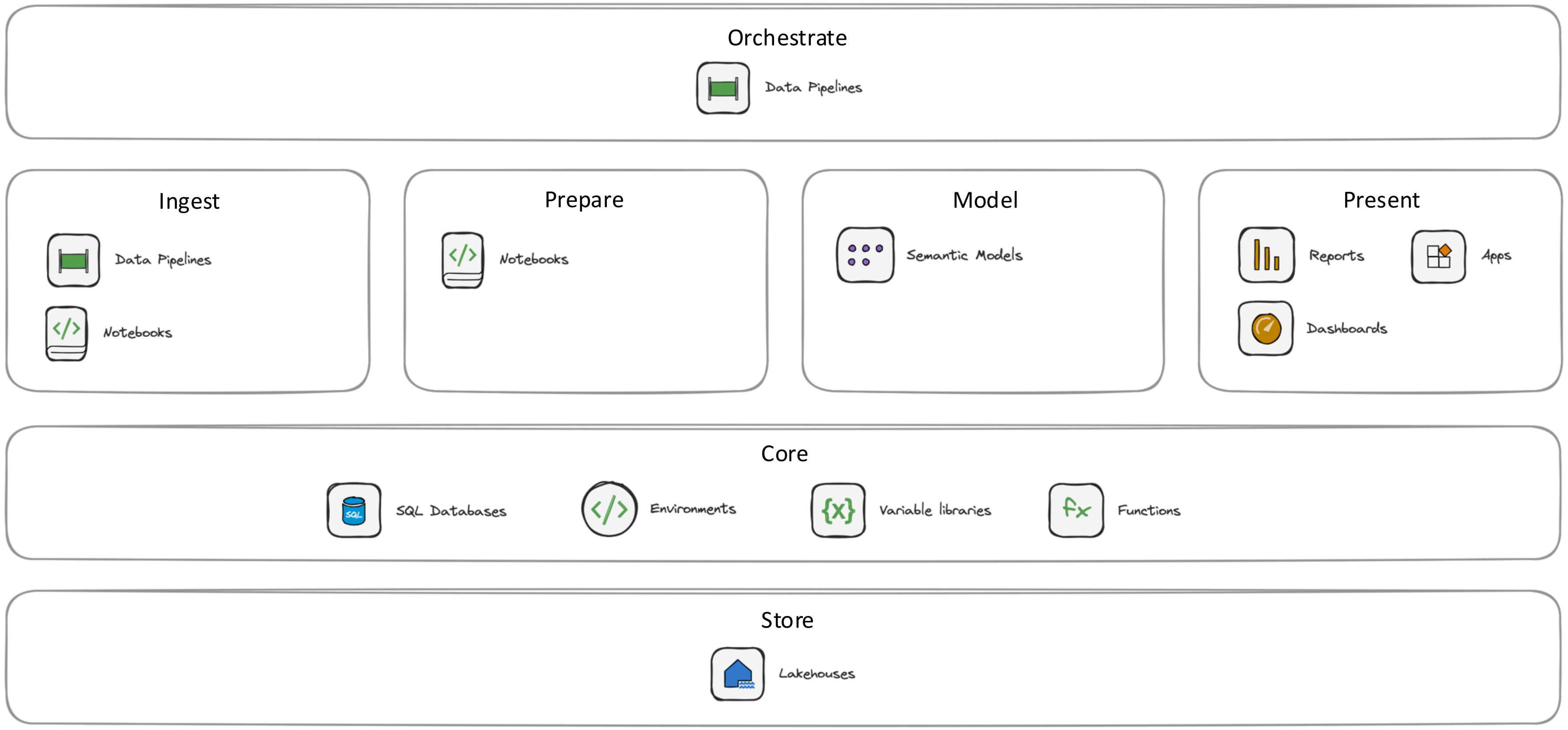


Reference Architecture as a Starting Point



Workspaces, repos and environments

Fabric Workspace structure

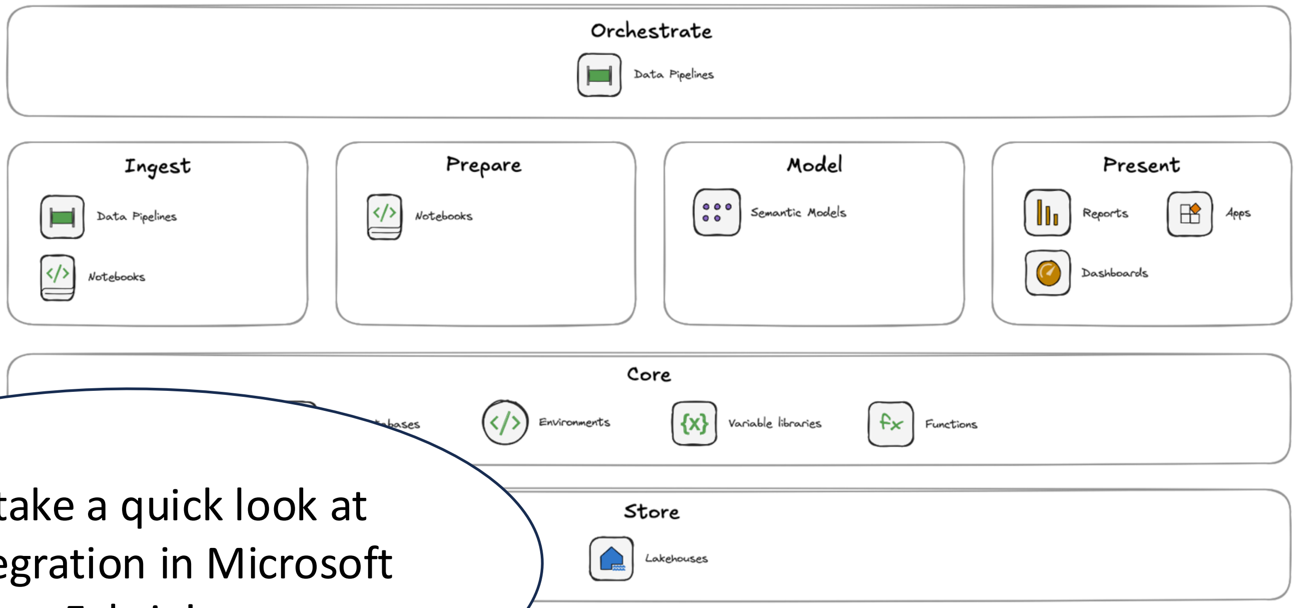


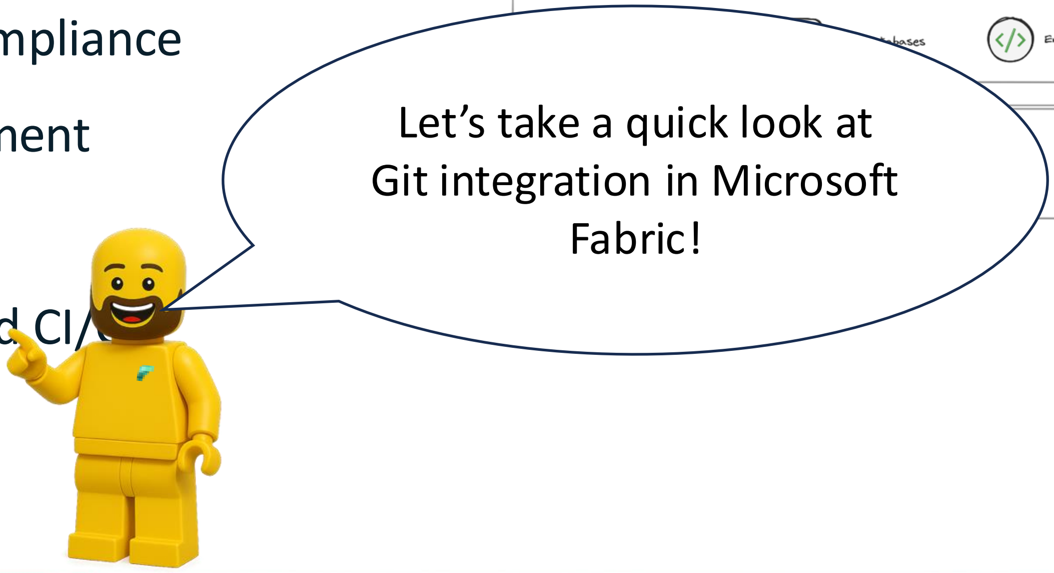
x3

[dev, tst, prd]

Fabric Workspace structure – Why this pattern?

- Security and Access
- Separation of duties
- Network & connectivity
- Capacity isolation
- Governance & compliance
- Testing & Deployment
- Item organization
- Git integration and CI/




Let's take a quick look at Git integration in Microsoft Fabric!

Git integration in Microsoft Fabric

- Integrates on workspace level
- Workspace -> branch
- Supports Github and Azure DevOps
- Sync is bi-directional (push/pull)
- Native support for most items

  (some in preview)
- API support (Fabric Core REST APIs)

Authentication options

Github	DevOps
Github PAT	OAuth 2.0
	Service Principal*

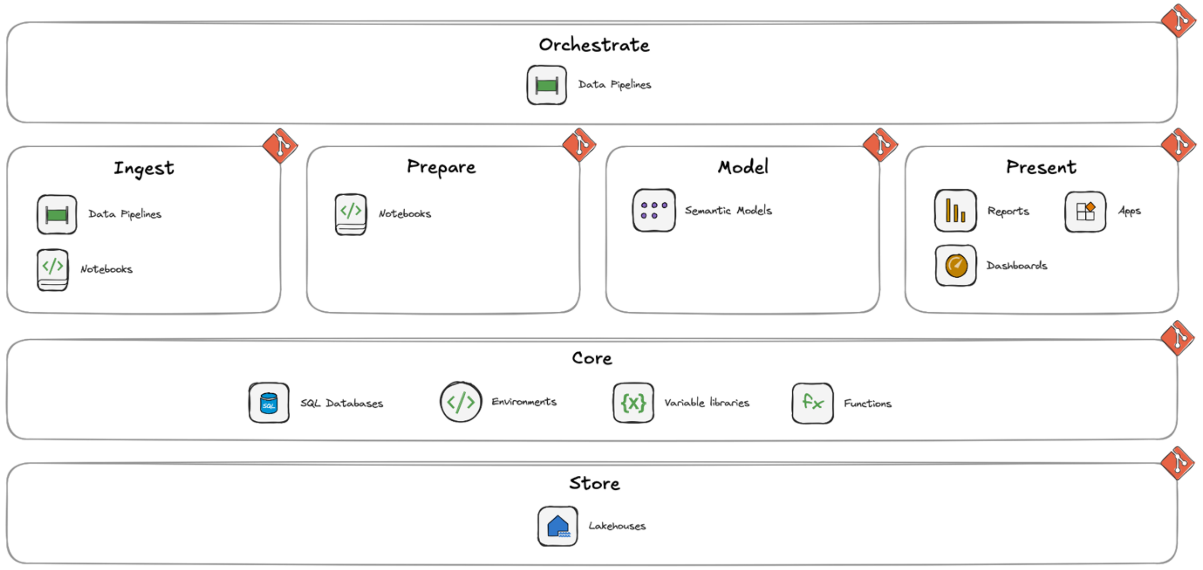
* SPN only partly supported.

Read more: https://learn.microsoft.com/en-us/fabric/cicd/git-integration/intro-to-git-integration

My go-to for Microsoft Fabric Git Integration

- Use Mono-repo
- Organized your workspaces by layer
- Using clear naming conventions
- Use the Git provider of your choice
- Branching strategy depends…

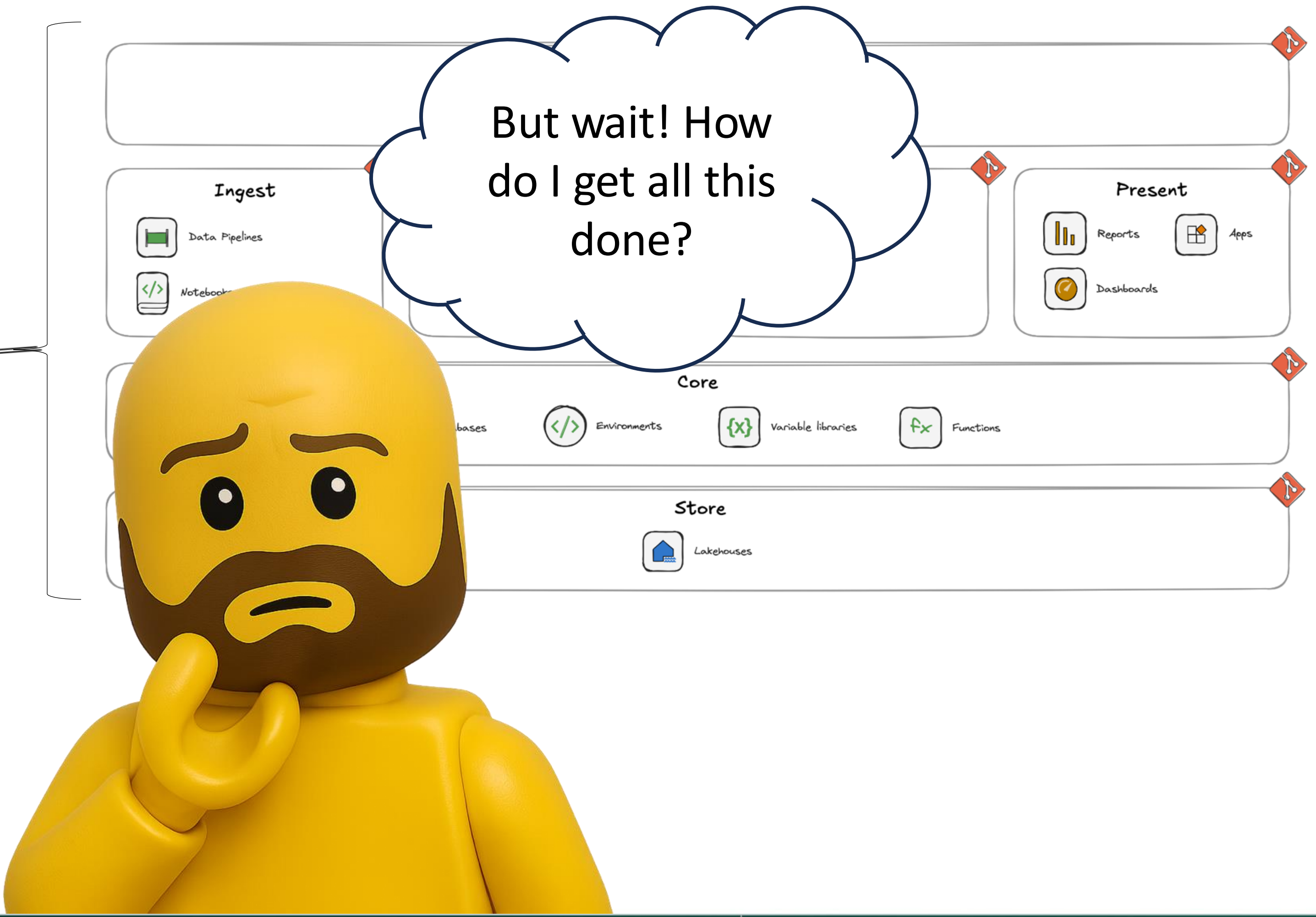
  there is no one-size fits all…
- Git integration all [dev] workspaces by default



My go-to for Microsoft Fabric Git Integration

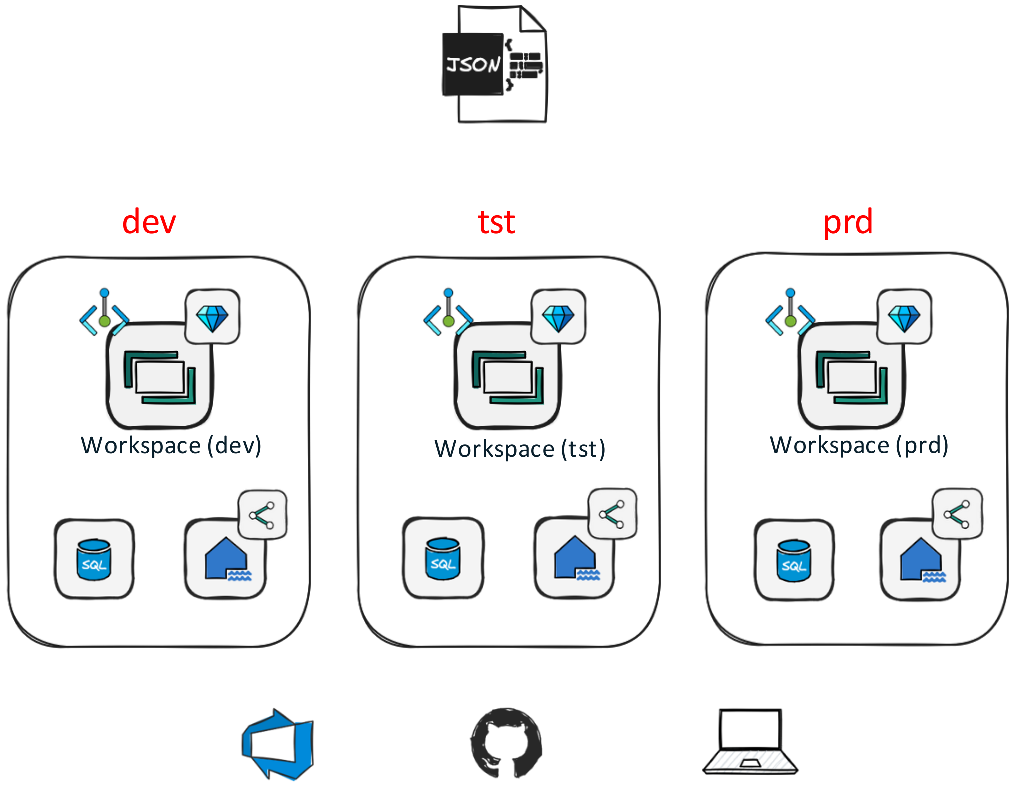
Git Repo structure

.azure-pipelines
.github
automation
documentation
solution
  /core
  /ingest
  /model
  /orchestrate
  /prepare
  /present
  /store



Automating Fabric solution setup

- Recipe based solution (json)
- Utilizes Python and the Fabric CLI
- Can run from Azure DevOps and GitHub as well as on your locale machine
- Minimum requirements:
  - A Service Principal
  - A Fabric Capacity

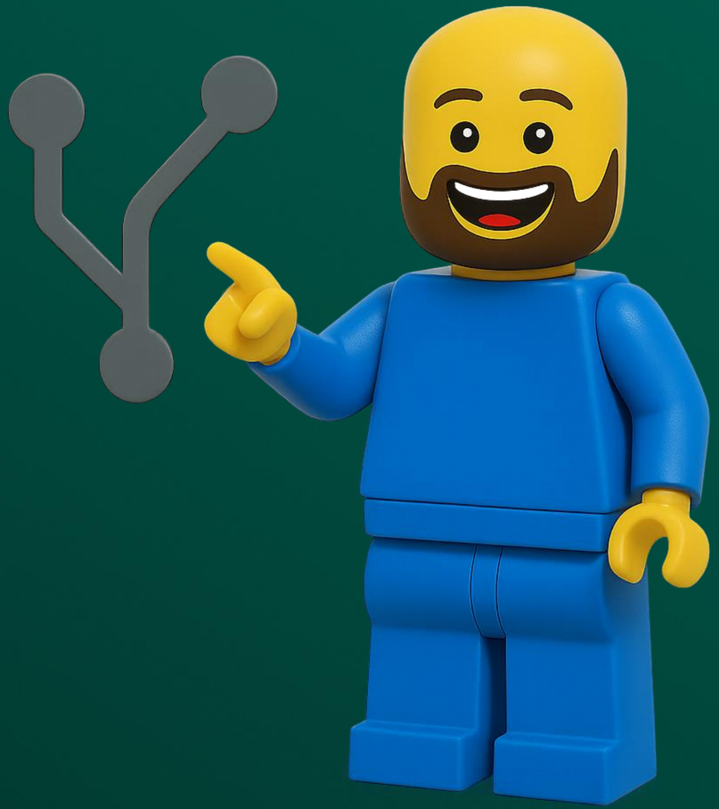


Github

Download source code: https://github.com/gronnerup/FabricAutomation

~~DEMO~~
SHOW TIME!

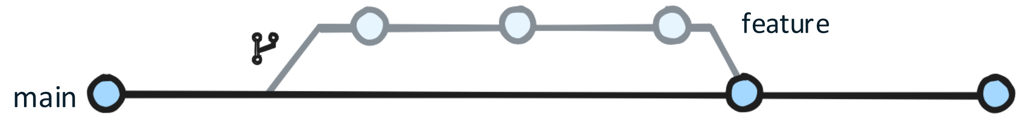
Branching for real-world collaboration



Branching for real-world collaboration

Branching Isn’t Just Git Hygiene - It’s a Collaboration Strategy!

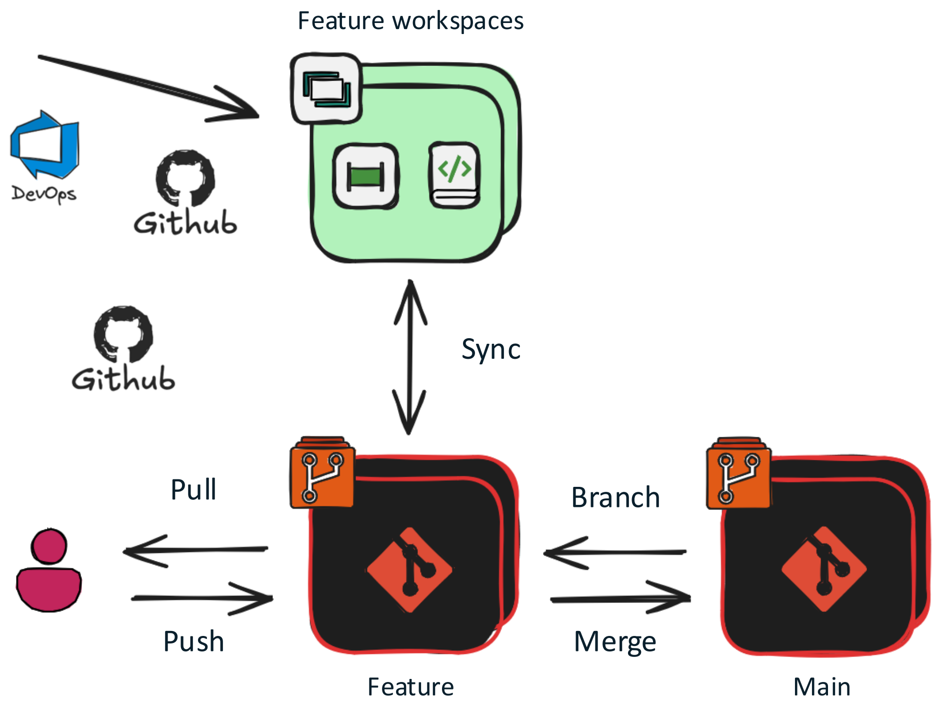
- Git is the source of truth, and branching is how we manage change
- Branches gives developers a seperate workspace for their code and…
  - Isolates development
  - Protects the mainline of our code
  - Enables parallel development
  - Help organize and structure releases
  - Is crucial for streamlined collaboration
  - Enables experiments



Development process – Ways-of-working

Development flow **- Supported by automation!**

1. Create branch from main
2. Develop
3. Create PR
4. Merge feature to main
5. Delete feature branch

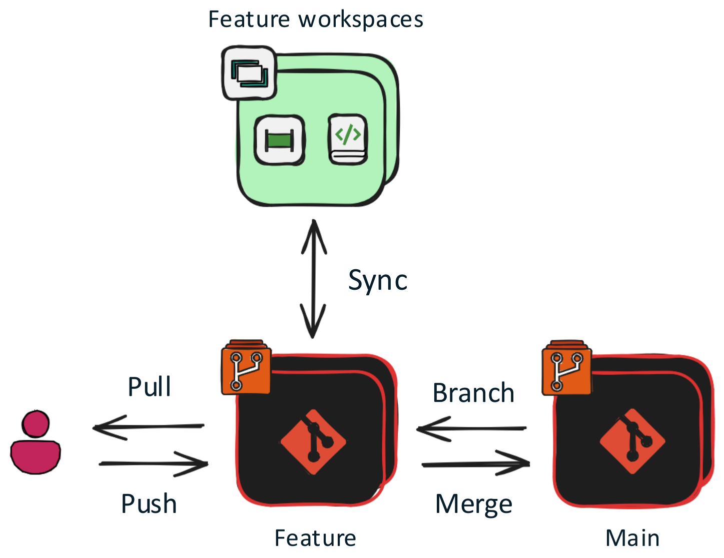


https://learn.microsoft.com/en-us/fabric/cicd/best-practices-cicd

Development process – Automated ways-of-working

Why automate this?

ClickOps	Automated
Manual process	Fully automated - just create the branch
1 workspace – 1 branch	Support for multiple workspaces
No transfer/setup of: - ACL - Spark settings - Private Endpoints - WS Identity	Customize: - ACL - Spark settings - Private Endpoints - WS Identity - Capacity
Inherits source capacity	Supports automated sync and cleanup
Requires manual cleanup	



https://peerinsights.emono.dk/automating-feature-workspace-maintainance-in-microsoft-fabric

https://justb.dk/blog/2025/02/fabric-spark-notebooks-and-cu-consumption/

DEMO TIME!

Automation & CI/CD Pipelines

Release process – The main options…

Fabric Deployment pipelines	Git based deployment	Git based deployment using build environment
• No code experience • For simpler solutions • No support for *: • Pre-deployment opr. • Post-deployment opr. • Test & validation	• Suitable when using Gitflow • Each environment connected to git branch • No need for building environments • Might require post-deployment operations	• Supports building enviroments • Deployment through ADO Pipelines/Github actions • Supports manipulation of item definitions

CI/CD Building Blocks for Microsoft Fabric

Tools & automation options

Azure DevOps Pipelines

Github actions

Python scripts & wrappers

Fabric CLI wrappers

fabric-cicd Python Library

Developer friendly

- Code-first approach
- No need to call Fabric APIs directly

Environment supports

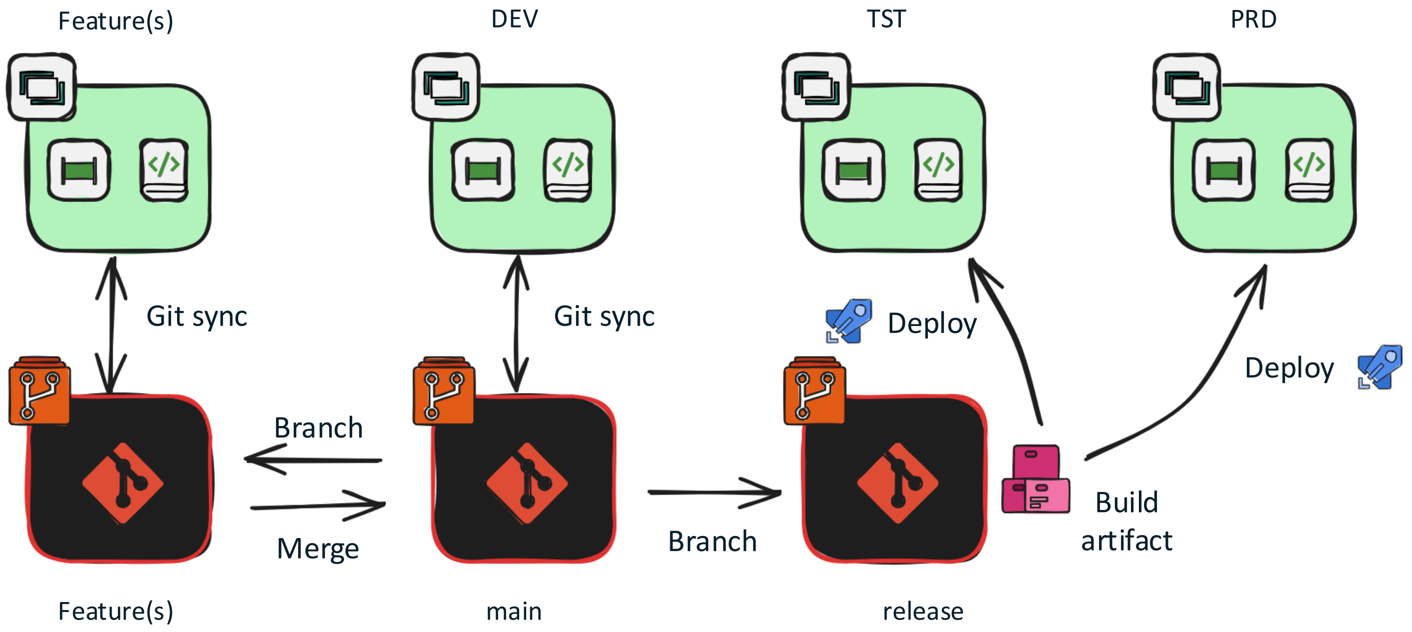
- Parameterization
- Reusable for locale, DevOps and more…

Smart deployment

- Deploy all supported items (with public APIs)
- Auto-unpublish orphaned artifacts

Release process – 3 selected scenarios

Scenario 1 – Main is always dev



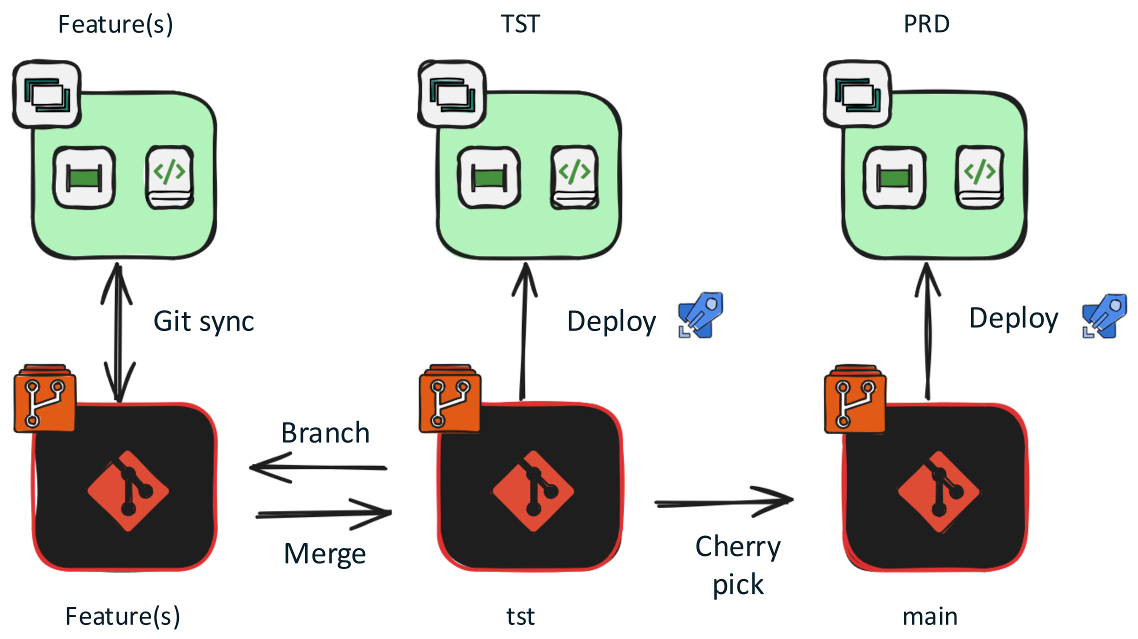
Suitable for:

- Smaller and more simple solitions
- Small teams
- Low interdependency between features

DEMO TIME!

Release process – 3 selected scenarios

Scenario 2 – PR to test branch, cherry pick to main



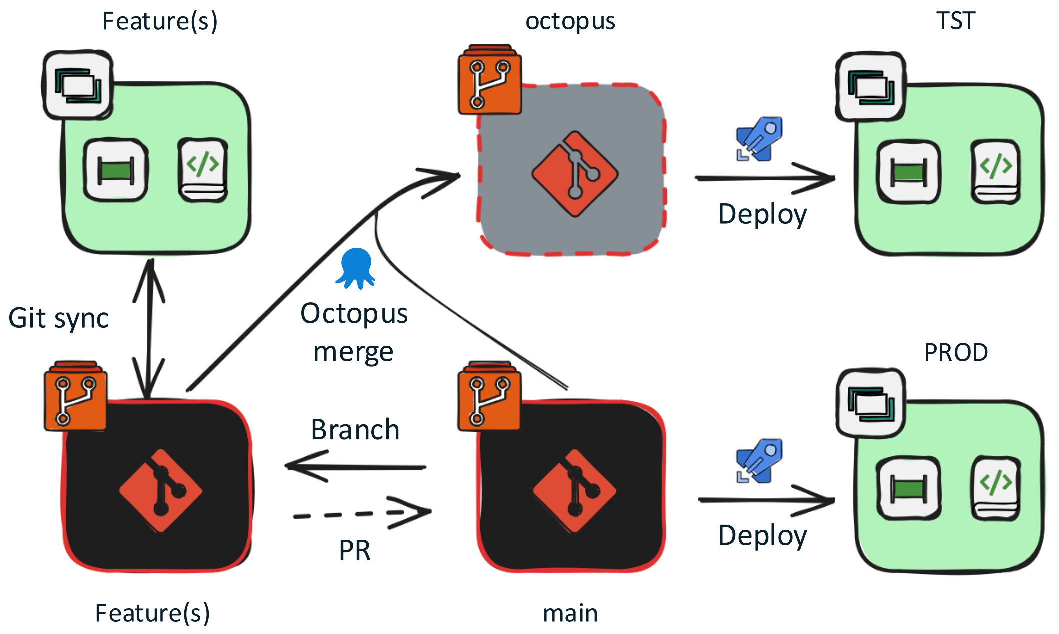
Suitable for:

- Medium to large solutions
- High-risk changes
- Supports per-feature validation
- Supports incremental and selective testing and deployment

Read the blog post by Jacob Knightly etc:
https://blog.fabric.microsoft.com/en-us/blog/optimizing-for-ci-cd-in-microsoft-fabric

Release process – 3 selected scenarios

Scenario 3 – Octopus merge



Suitable for:

- Large teams
- Many parallel features
- Interdependent changes
- High-risk changes
- Supports per-feature validation
- Supports incremental and selective testing and deployment

DEMO TIME!

Tips, Tricks & Summing up

Tips, Tricks & Summing Up!

- Split your workloads and layers across multiple workspace
- Use mono-repo structure as a starting point
- Leverage the Fabric CLI, REST APIs, and/or Terraform for automation
- Use the fabric-cicd Python library to deploy Fabric items
- Design everything with automation in mind
  - Invest in dynamic pipelines, notebooks and reusable patterns
  - Apply strict and consistent naming conventions
- Implement a metadata-driven framework for scalability robustness and speed
- Stay curious - Get inspired by what others build!

Download the showcase code and presentation

https://github.com/gronnerup/FabricAutomation

Disclaimer:
The solution demonstrated in this session is **experimental** and provided for **showcasing purposes only**. It is **unsupported**, and there are no guarantees regarding functionality or future updates. You are free to use it **as-is** or modify it to fit your needs. Use at your own risk.

TIME FOR QUESTIONS!

BEFORE CONNECTION IS TERMINATED BY PEER...