Git Branching: Moving to recommended model

Theory, practice and demo

Ian J Mitchell Dennis Behm (Mathieu Dalbin)

Table of contents

0.1 Agenda

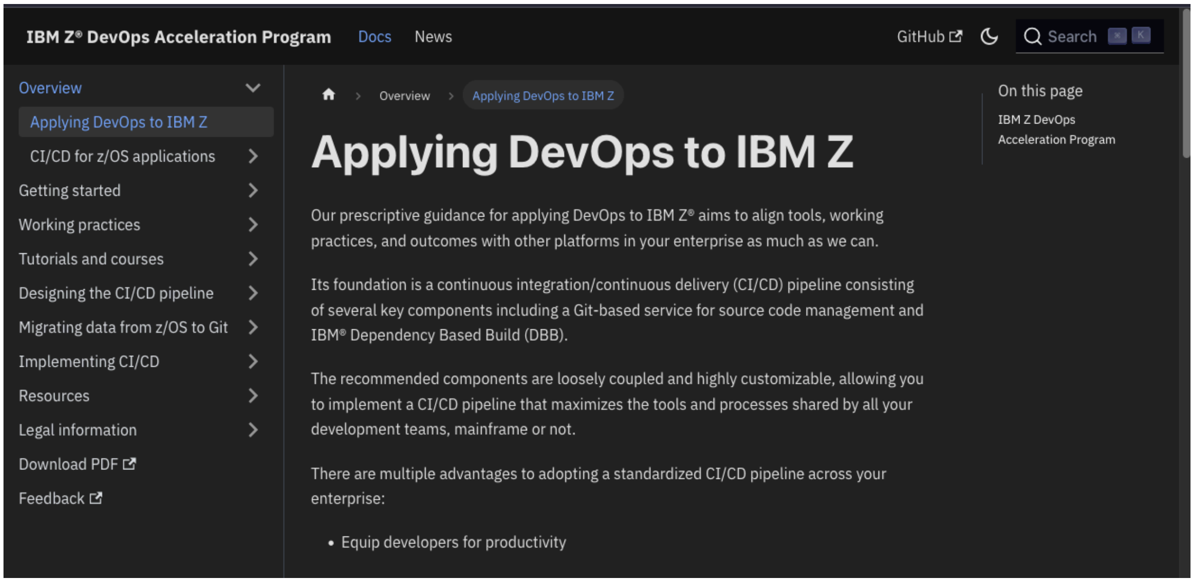
- Setting the context
  - Aims and Assumptions
- Fundamentals of the branching strategy
  - Supported workflows
- The pipelines in action - demo time!

0.2 Setting the context

- Devops is foundational to modernization.
- Modernization of Devops is no longer a voyage of exploration.
  - (But plenty people will still try to convince you otherwise!)
- No single tool or approach covers the entire journey to the destination.

We have a Guide... yes, it would be better if it were a map!

0.3 The Guide



Applying DevOps to IBM Z

Our prescriptive guidance for applying DevOps to IBM Z® aims to align tools, working practices, and outcomes with other platforms in your enterprise as much as we can.

Its foundation is a continuous integration/continuous delivery (CI/CD) pipeline consisting of several key components including a Git-based service for source code management and IBM® Dependency Based Build (DBB).

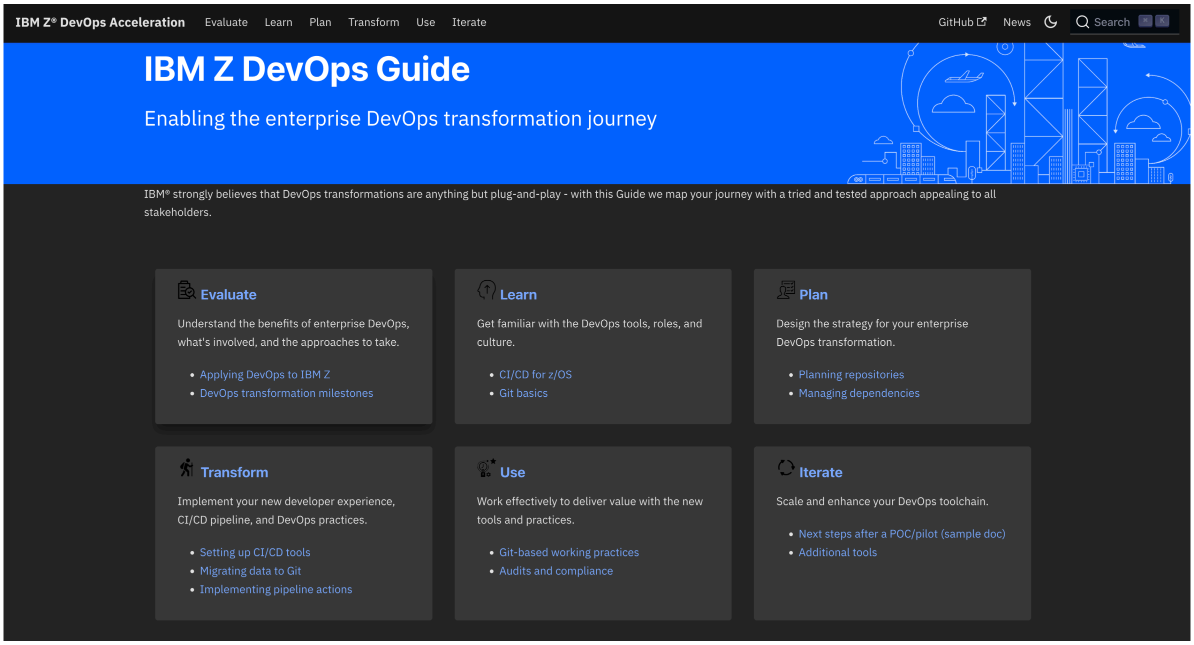
The recommended components are loosely coupled and highly customizable, allowing you to implement a CI/CD pipeline that maximizes the tools and processes shared by all your development teams, mainframe or not.

There are multiple advantages to adopting a standardized CI/CD pipeline across your enterprise:

- Equip developers for productivity

0.4

0.4.0.1 Sneak-peek - new Guide landing page



IBM Z DevOps Guide

Enabling the enterprise DevOps transformation journey

IBM® strongly believes that DevOps transformations are anything but plug-and-play - with this Guide we map your journey with a tried and tested approach appealing to all stakeholders.

Evaluate

Understand the benefits of enterprise DevOps, what's involved, and the approaches to take.

- Applying DevOps to IBM Z
- DevOps transformation milestones

Learn

Get familiar with the DevOps tools, roles, and culture.

- CI/CD for z/OS
- Git basics

Plan

Design the strategy for your enterprise DevOps transformation.

- Planning repositories
- Managing dependencies

Transform

Implement your new developer experience, CI/CD pipeline, and DevOps practices.

- Setting up CI/CD tools
- Migrating data to Git
- Implementing pipeline actions

Use

Work effectively to deliver value with the new tools and practices.

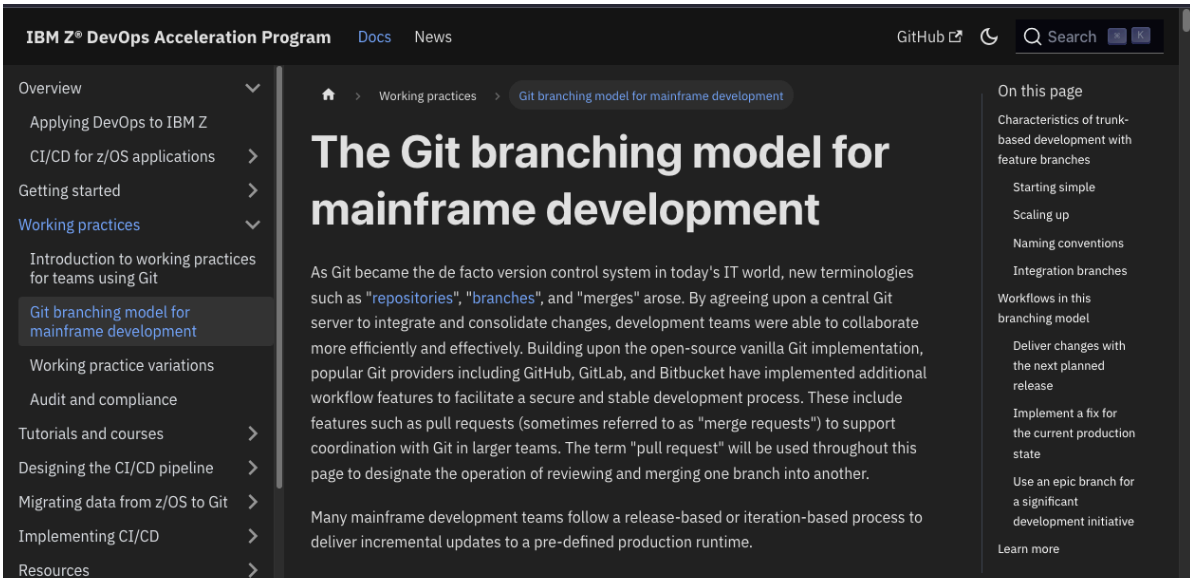
- Git-based working practices
- Audits and compliance

Iterate

Scale and enhance your DevOps toolchain.

- Next steps after a POC/pilot (sample doc)
- Additional tools

0.5 Today's topic - Branching Strategy for Mainframe Teams

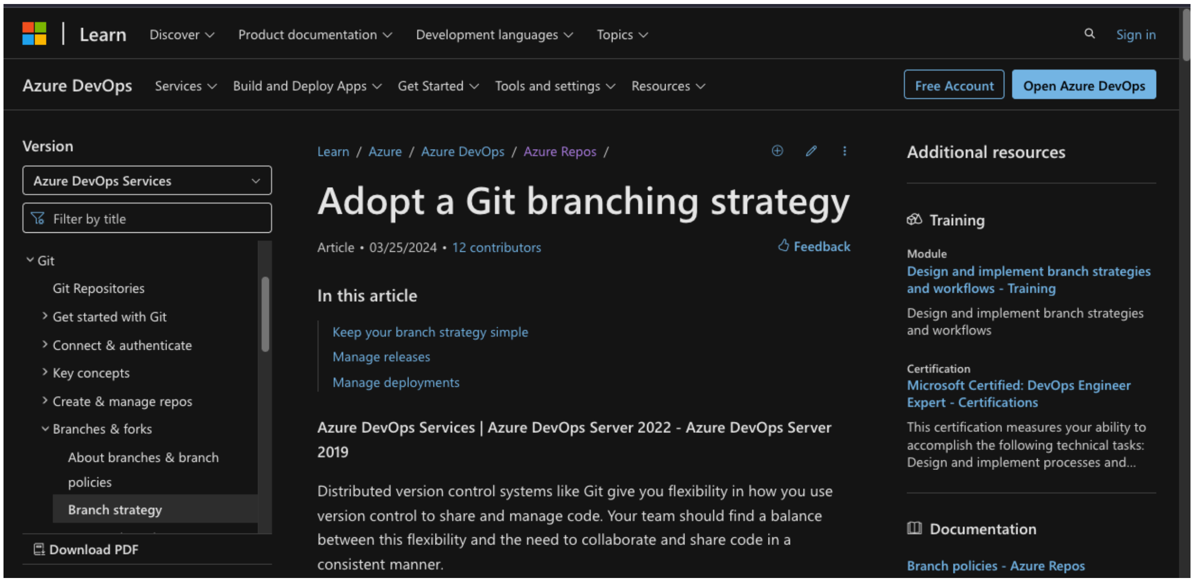

IBM Z® DevOps Acceleration Program
Docs
News
GitHub
Search
Overview
Applying DevOps to IBM Z
CI/CD for z/OS applications
Getting started
Working practices
Introduction to working practices for teams using Git
Git branching model for mainframe development
Working practice variations
Audit and compliance
Tutorials and courses
Designing the CI/CD pipeline
Migrating data from z/OS to Git
Implementing CI/CD
Resources
Working practices
Git branching model for mainframe development
On this page
Characteristics of trunk-based development with feature branches
Starting simple
Scaling up
Naming conventions
Integration branches
Workflows in this branching model
Deliver changes with the next planned release
Implement a fix for the current production state
Use an epic branch for a significant development initiative
Learn more

The Git branching model for mainframe development

As Git became the de facto version control system in today's IT world, new terminologies such as "repositories", "branches", and "merges" arose. By agreeing upon a central Git server to integrate and consolidate changes, development teams were able to collaborate more efficiently and effectively. Building upon the open-source vanilla Git implementation, popular Git providers including GitHub, GitLab, and Bitbucket have implemented additional workflow features to facilitate a secure and stable development process. These include features such as pull requests (sometimes referred to as "merge requests") to support coordination with Git in larger teams. The term "pull request" will be used throughout this page to designate the operation of reviewing and merging one branch into another.

Many mainframe development teams follow a release-based or iteration-based process to deliver incremental updates to a pre-defined production runtime.

0.6 ... don't just take our word for it



Adopt a Git branching strategy

Article • 03/25/2024 • 12 contributors

Feedback

In this article

- Keep your branch strategy simple
- Manage releases
- Manage deployments

Azure DevOps Services | Azure DevOps Server 2022 - Azure DevOps Server 2019

Distributed version control systems like Git give you flexibility in how you use version control to share and manage code. Your team should find a balance between this flexibility and the need to collaborate and share code in a consistent manner.

Microsoft Azure Devops Services

1 Lifecycle and terminology comparison

1.1 Code storage and developer experience

Developer Experience	Old	New
Source Code Storage	PDS Members	UNIX/Linux Filesystem
Editing	ISPF	IDEs: IDZ, VSCode
Versioning	Package IDs	Git **branches**/commits

1.2 Processes and Automation:

Process	Old	New
Building	JCL	**Automated Pipeline** scripts
Approval	Proprietary	Pull Requests
Deployment	JCL	IBM Wazi Deploy, IBM Urban Code Deploy

2 Aims and Assumptions

Some aims and assumptions that guide our recommendations...

2.1 No baggage

Well... *travel light* perhaps!

- Are we *prescriptive* or just *opinionated*?
  - Aim to be *git-native*!
- We start with a recommendation
  - Confidently
- We question everything
  - **YAGNI** - “you aren’t gonna need it”
- We strive for simplicity
  - For each user’s experience

2.2 Scaling

- The workflow and branching scheme should both scale up and scale down.
  - Small teams with simple and infrequent changes will be able to easily understand, adopt, and have a good experience.
  - Large, busy teams with many concurrent activities will be able to plan, track, and execute with maximum agility using the same fundamental principles.

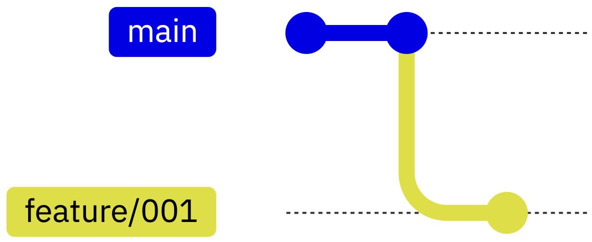
2.3 Planning

- Planning and design activities as well as code development aim to align to a regular release cadence.
- The cadence of the next planned release is defined by the application team.
- Planning assumptions should aim as much as possible to complete changes quickly, ideally within one release cycle.
  - DevOps/Agile practices to break down larger changes into sets of **smaller, incremental** deliverables that can each be completed within an iteration.

3 The Branching Strategy

3.1 Starting Simple

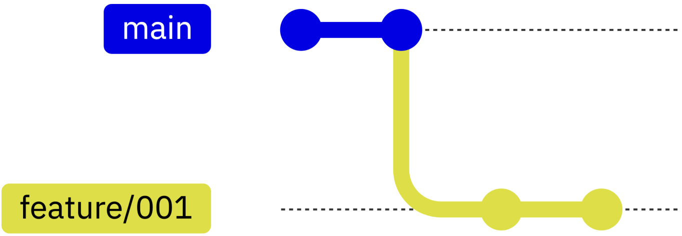
Every change starts in a branch.



- Developers work in the branch to make changes, perform user builds and unit tests.
- A branch holds multiple commits (changes to multiple files).

3.2 Starting Simple

Every change starts in a branch.



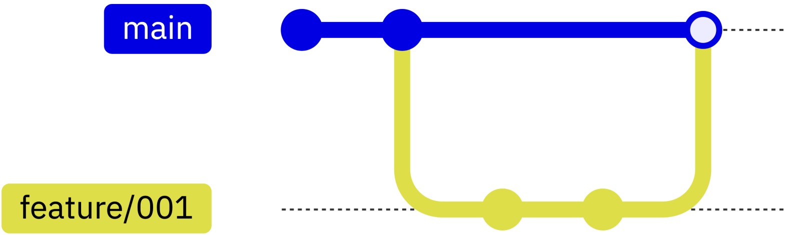
These changes on these branches are

- built,
- tested,
- reviewed and
- approved before merging to `main`.

3.3 Preparing to Merge into `main`

Feature Team/Developers will:

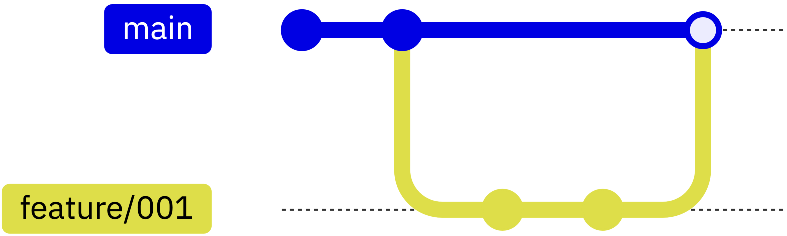
- Build
  - Builds may be done to any commit on any branch
  - Feature branch **must** build cleanly for a Pull Request
- Test
  - To prove quality of the changes in their feature branch



3.4 Merging into `main`

Create a Pull Request (PR) to signal to Team Leaders/Release Controllers to:

- Review
  - Code and Test results
- Approve
  - Safeguard the quality of `main`



`main` and other long-lived branches are *protected* (not everyone can `push` or `merge` to them).

3.5 Before you ask... no, there is no *Production* branch

CI/CD decouples the building and deploying to test environments and production.

We have no branches named `prod` (or `test` or `QA`)

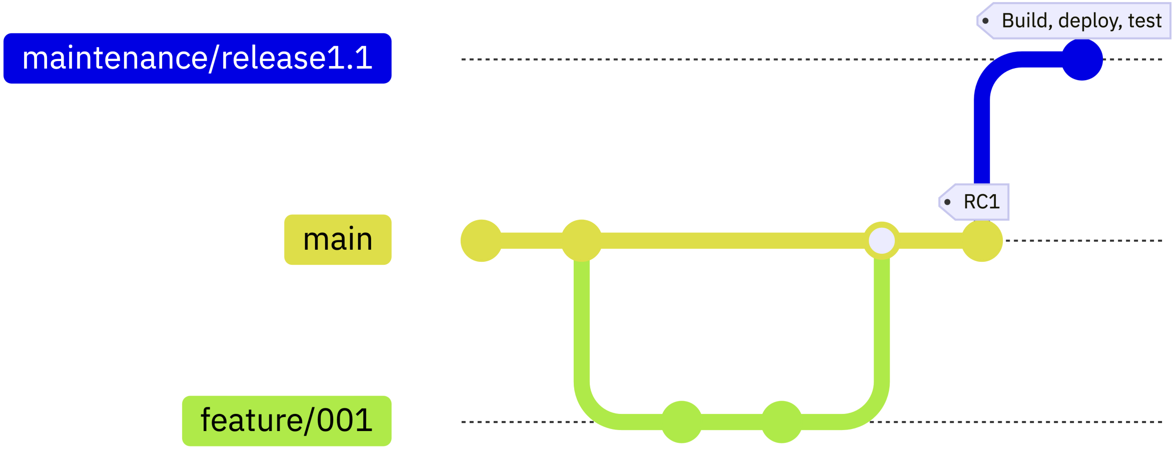
- Those are *environments* to which builds can be deployed
- Such extra branches:
  - are unnecessary
  - cause ambiguity
  - impose merging and building overheads
- Deployment manager is maintaining what is deployed where and provides traceability from developer to production deployment

3.6 Testing a release candidate

Any point in the history of `main` can be declared a *release candidate*.

- Build a *release candidate* package
- Deploy it

- Test it



Tag the commit (point in `main`'s history) with a release name.

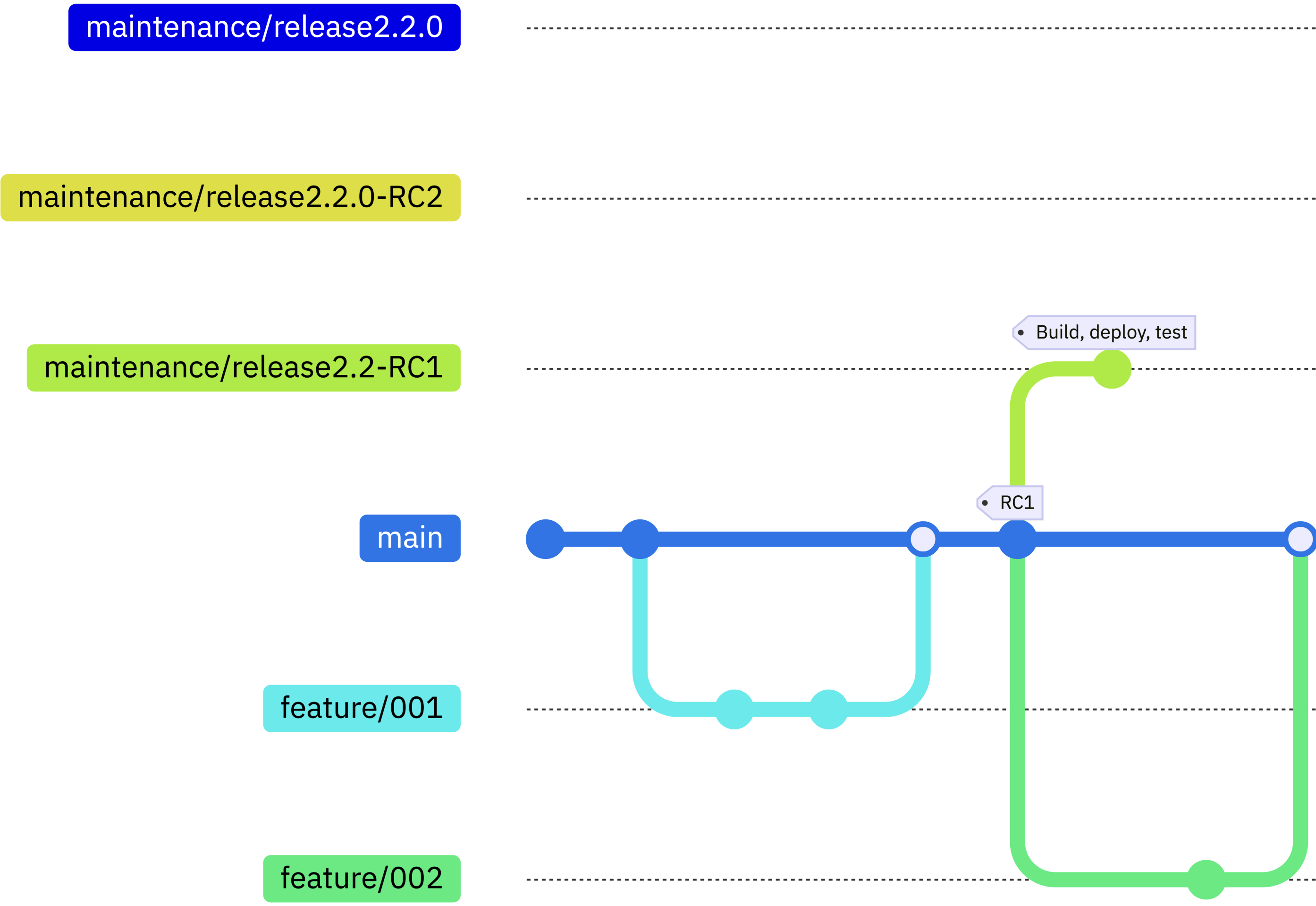
3.7 Deploying to production

When…

- all the committed work items for the next planned release are ready to be shipped, and
- all quality gates have passed successfully and the required approvals have been issued by the appropriate reviewers,

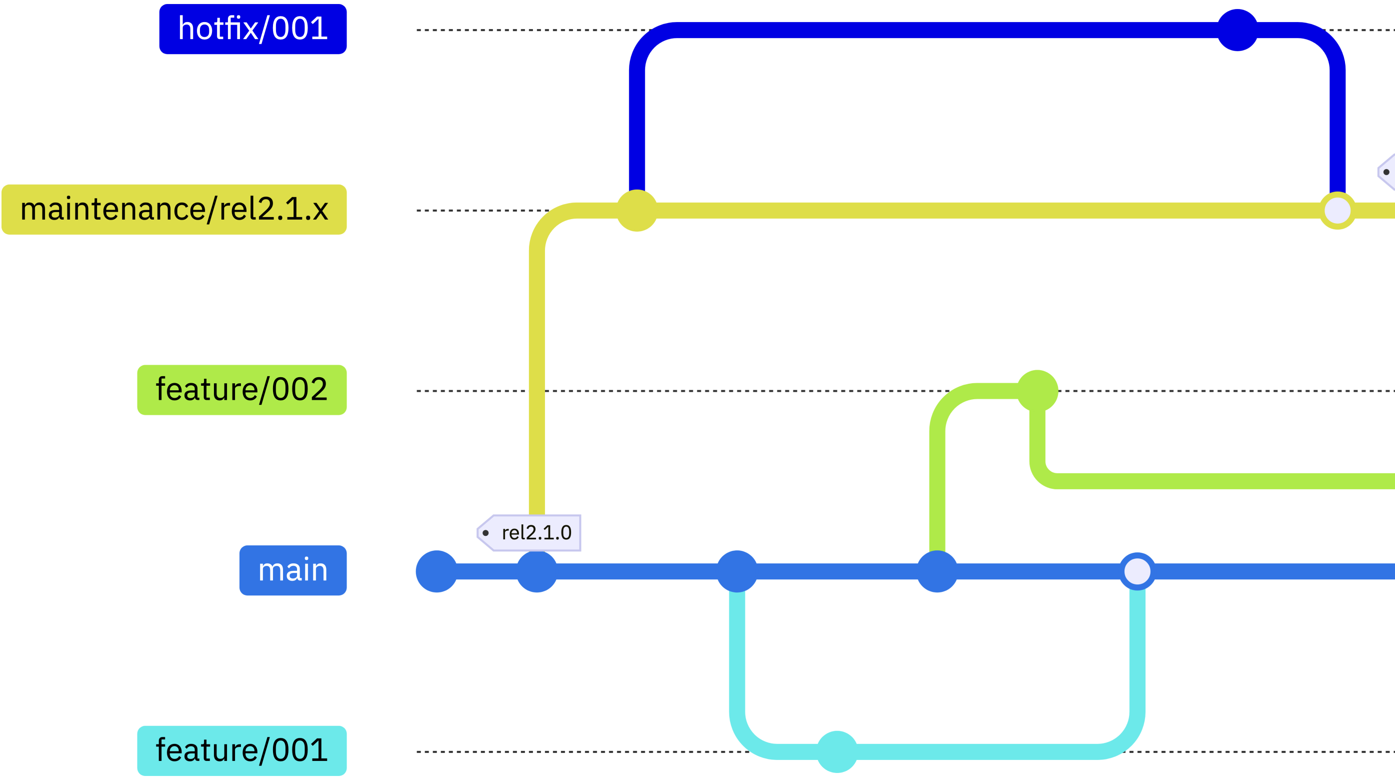
the release package can be deployed to production…

3.8 Deploying to production



3.9 Release maintenance branches

A *release maintenance* branch will be used if *hot-fixes* must be developed and delivered.



3.10 Scaling up

Concurrent *feature* branches scale very well, but assume short cycle times.

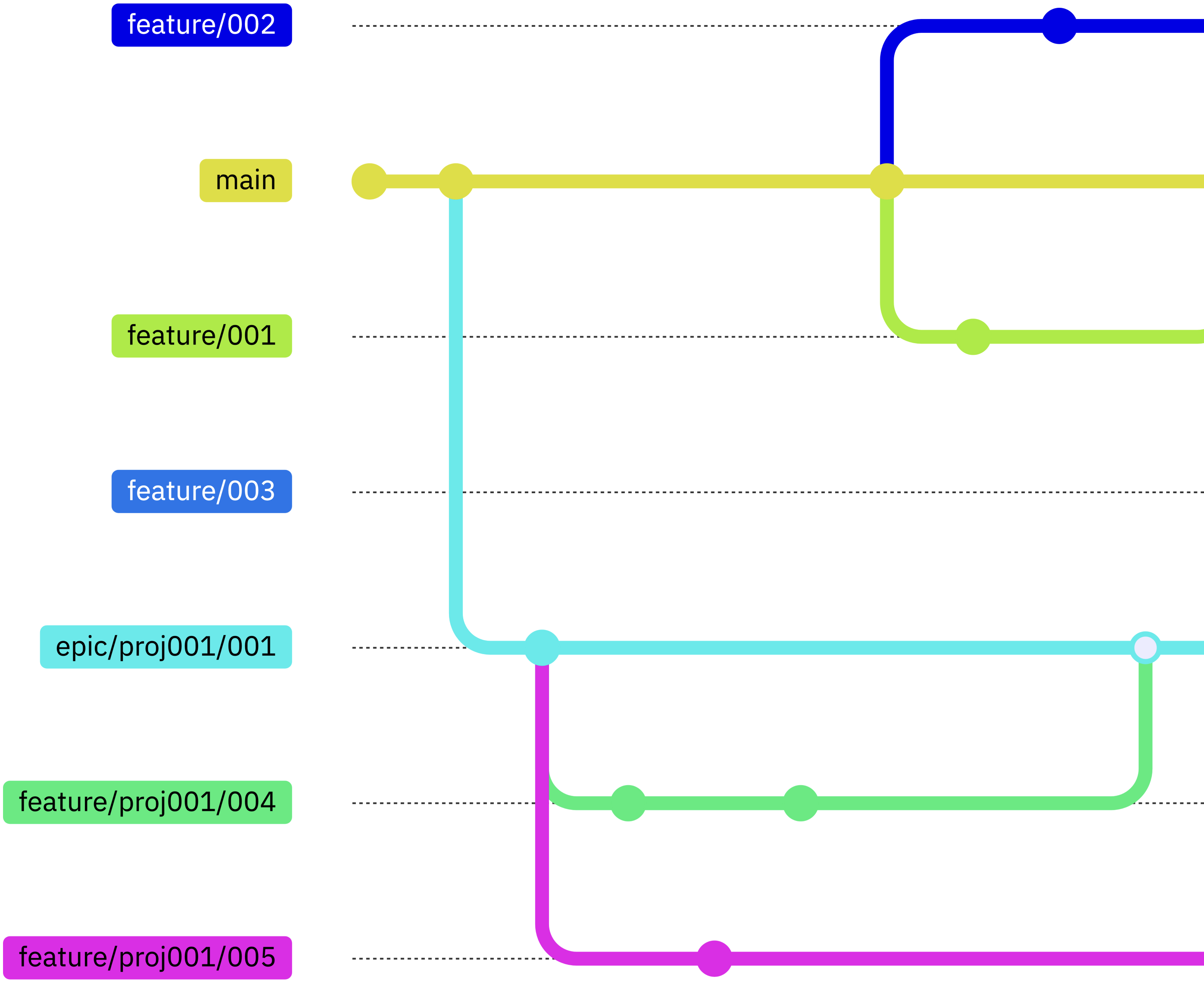
- Ideally live within a release delivery cycle
- But no big deal if they don't

Epic branches can collect multiple features

- Before going to `main`
- When the delivery is planned beyond the next release

(*Epic* branches are a form of *integration* branch.)

3.11 Integration branches



3.12 Convention over configuration

The principles are more important that the tools and names.

Naming conventions - making purpose obvious:

- `main` : single source of truth. The only long-living branch.

- `release/rel-2.0.1` : explicit versioning numbering to identify releases maintenance branch.
- `epic/47-ai-fraud-detection` : development initiative identified by epic id and description.

3.13 Convention over configuration

Changes are implemented on feature branches:

- `feature/<jira-id|servicenow-id>-new-mortgage-calculation` : references to other planning tools for new features for the next planned release.
- `feature/47-ai-fraud-detection/refactor-mortgage-calculation`: feature implemented for development initiative
- `hotfix/rel-2.0.1/fix-mortgage-calc`: fix implemented for release rel-2.0.1

Names of branches and tags flow through to builds and deployments.

4 Workflows supported by the strategy

How do teams work with and benefit from the branch strategy?

4.1 The types of workflows

1) Work and focus on the **next planned release** via the `main` branch. After planning the work items for the next release, the development team is adding changes to the main branch.

2) **Resolution of a production problem** in the currently-released version of the application by leveraging a release maintenance branch that is used for maintenance purposes,

3) **concurrent development activities for significant development initiatives**, which include multiple planned work items for a later delivery (including starting development of a future release) by creating an epic branch from a commit point in the history of main.

The following narratives complement the structure.

4.2 Features for the next planned release

Developer activities:

1. Take item from backlog
2. Create feature branch
3. Code locally
4. Build with DBB User Build
5. Commit into feature branch, build and test feature functionality
6. Create Pull Request for review and approval

4.3 Features for the next planned release

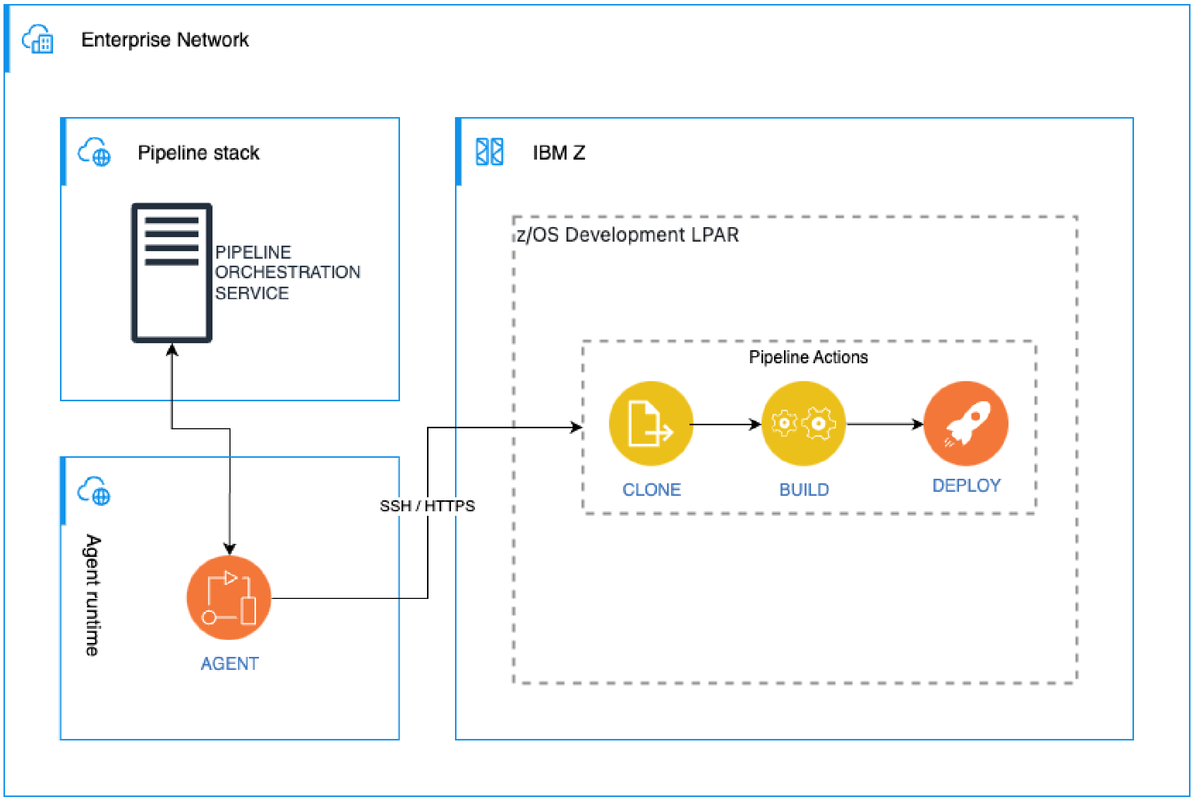
Release control activities:

7. Merge approved branches to `main` and build
8. When planned features have merged, build and tag Release Candidate package
9. Deploy Release Candidate to test environments
10. (*Release candidate can be superceded*)
11. Finalise Release package in Artifactory (or equivalent)
12. Tag commit from step 8 as the production release and deploy

5 Simplifying pipeline implementations

5.1 Plan your Pipeline Architecture

- Different pipelines orchestrators are used by customers
- Principle implementations need to safeguard discussed principles and rules

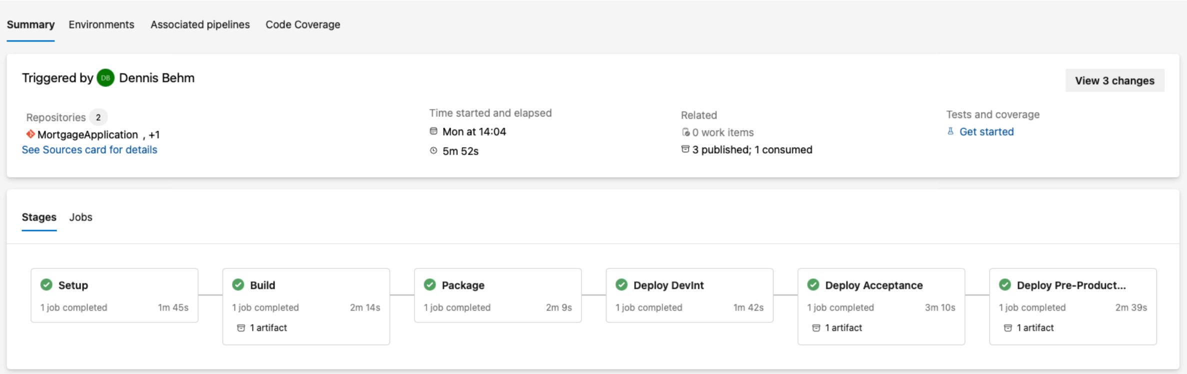


5.2 Common needs across all implementations

- Computation of a unique build high-level qualifier
- Set build configuration depending on the lifecycle the user is in
  - Compile with test options
  - Compile with optimize options
- Enforcing a common strategy to store packages in your enterprise binary repository

5.3 Pipeline templates and common backend services

Pipeline templates for the various pipeline orchestrators + common set of services that implement the recommended workflow.



Triggered by DB Dennis Behm

Repositories 2
MortgageApplication , +1
See Sources card for details

Time started and elapsed
Mon at 14:04
5m 52s

Related
0 work items
3 published; 1 consumed

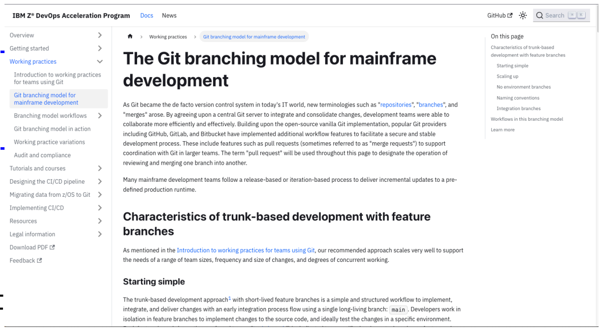
Tests and coverage
Get started

Stage	Status	Duration	Artifacts
Setup	1 job completed	1m 45s	
Build	1 job completed	2m 14s	1 artifact
Package	1 job completed	2m 9s	
Deploy DevInt	1 job completed	1m 42s	
Deploy Acceptance	1 job completed	3m 10s	1 artifact
Deploy Pre-Product...	1 job completed	2m 39s	1 artifact

6 Collateral

6.1 References

- zDevOps adoption enablement website - Git branching for mainframe development
- A no-baggage approach to new Mainframe development practices
- Pipeline templates - IBM DBB Community Repository
- Videos
  - Using the branching model (4 short videos)
  - Day in the life of a mainframe developer using Git



IBM Z® DevOps Acceleration Program
Docs
News
GitHub
Search
Overview
Getting started
Working practices
Introduction to working practices for teams using Git
Git branching model for mainframe development
Branching model workflows
Git branching model in action
Working practice variations
Audit and compliance
Tutorials and courses
Designing the CI/CD pipeline
Migrating data from z/OS to Git
Implementing CI/CD
Resources
Legal information
Download PDF
Feedback
Working practices > Git branching model for mainframe development
On this page
Characteristics of trunk-based development with feature branches
Starting simple
Scaling up
No environment branches
Naming conventions
Integration branches
Workflows in this branching model
Learn more

The Git branching model for mainframe development

As Git became the de facto version control system in today's IT world, new terminologies such as "repositories", "branches", and "merges" arose. By agreeing upon a central Git server to integrate and consolidate changes, development teams were able to collaborate more efficiently and effectively. Building upon the open-source vanilla Git implementation, popular Git providers including GitHub, GitLab, and Bitbucket have implemented additional workflow features to facilitate a secure and stable development process. These include features such as pull requests (sometimes referred to as "merge requests") to support coordination with Git in larger teams. The term "pull request" will be used throughout this page to designate the operation of reviewing and merging one branch into another.

Many mainframe development teams follow a release-based or iteration-based process to deliver incremental updates to a pre-defined production runtime.

Characteristics of trunk-based development with feature branches

As mentioned in the Introduction to working practices for teams using Git, our recommended approach scales very well to support the needs of a range of team sizes, frequency and size of changes, and degrees of concurrent working.

Starting simple

The trunk-based development approach[1] with short-lived feature branches is a simple and structured workflow to implement, integrate, and deliver changes with an early integration process flow using a single long-living branch: `main`. Developers work in isolation in feature branches to implement changes to the source code, and ideally test the changes in a specific environment.

Check out this presentation in Github!

7 THANK YOU!

Submit your feedback at:

https://conferences.gse.org.uk/2025V/feedback/AG

(Make sure you are signed into MyGSE)