CI/CD for a database project

About me

✓Passionate about Data ☺

✓Founder at Softentity

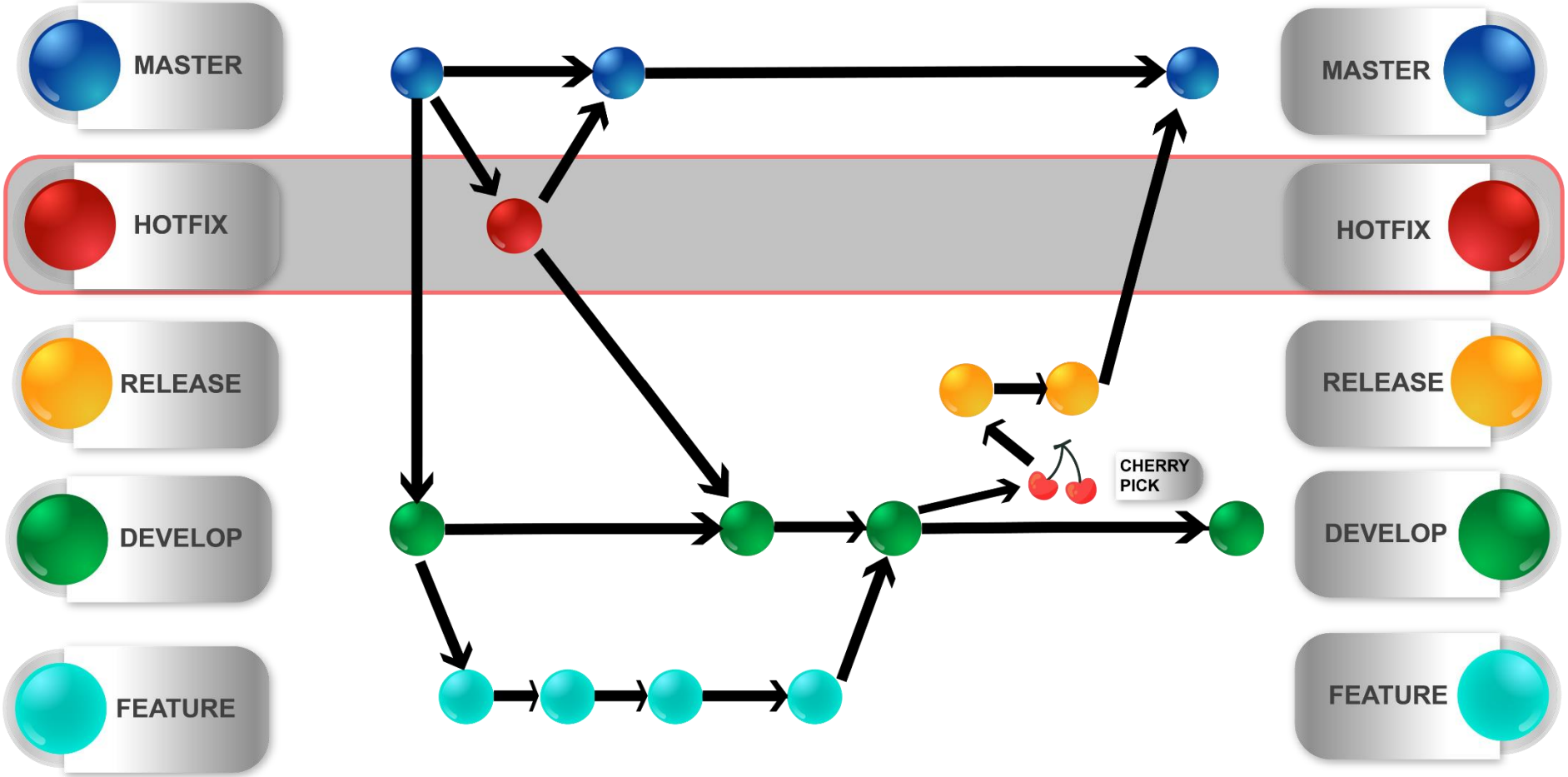
✓8+ years of experience in the industry

✓Certified Microsoft Data Engineer

Connect with me:

LinkedIn: https://www.linkedin.com/in/iulianatuhasu/

Email: tuhasu.Iuliana@gmail.com


MASTER
HOTFIX
RELEASE
DEVELOP
FEATURE
CHERRY PICK
MASTER
HOTFIX
RELEASE
DEVELOP
FEATURE

Gitflow Workflow – Summary

master branch	→ main remote branch that contains production data (SQL Server production environment) → only hotfix and release branches will merge into master
develop branch	→ second remote branch that contains development data (SQL Server development environment) → feature and release branches are created from develop → feature and hotfix branches merge into develop
feature branch	→ local branch created from develop to perform work and merged back into develop
release branch	→ release branch is initially created from master → encompass previous commits with functionalities ready to be released → the purpose of the release branch is to keep all the items we want to release at a certain period of time (daily/weekly) → at a certain point in time (E.g. 1/week) the release branch will be merged with master via PR and a CD pipeline will run for the deployment
hotfix branch	→ local branch created directly from master to quickly solve bugs → merged into master and back to develop for synchronization

Gitflow Workflow actions

For this presentation, the following actions are going to be covered:

- **Clone**: get a local copy of an existing repository
- **Fetch**: downloads new data from a remote repository without updating your working files
- **Pull**: downloads new data from a remote repository and updates your working files with these changes (Fetch + Merge)
- **Merge**: used to merge a specific commit into a branch / merge the whole branch into another branch
- **Commit**: a snapshot of your Git repository at one point in time used to record the changes in the repository
- **Stage**: used to mark specific files for inclusion in the next commit
- **Sync**: pulls changes from the remote to sync with local changes, and then pushes changes to the remote repository
- **Push**: push your local branch to the remote server
- **Cherry-pick**: picks a commit from a branch and applies it to another branch

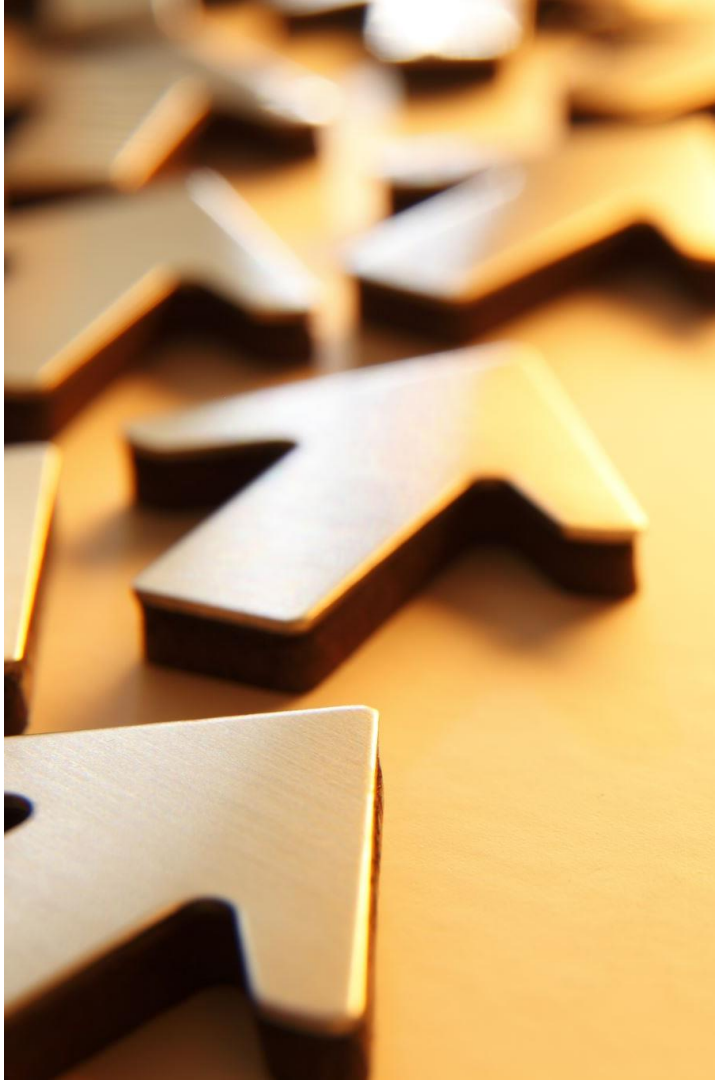
Demo - Scenario 1

We have a single environment that does not have version control for our SQL database. We have acquired another server that will serve as a dedicated SQL development environment. The repository on the new server is currently empty, and our goal is to synchronize the production data into the repository and deploy it to the development environment.

Demo – Scenario 2

You have a new requirement to create a table called "Employee". To implement this, we will follow these steps:

- 1.Create the Employee table in the SQL development environment.
- 2.Create a feature branch from the "develop" branch.
- 3.Use Schema Compare to update the local branch with the changes performed in the SQL development environment (Employee table)
- 4.Stage and commit the changes in the local branch.
- 5.Push the local branch to the remote server.
- 6.Initiate a pull request to merge the changes from your feature branch into the "develop" branch
- 7.Cherry-pick the specific commit related to the Employee table using Azure DevOps.
- 8.Merge the resultant branch, which includes the cherry-picked commit, into the "release" branch.
- 9.Initiate a pull request between the "release" and "main" branches to integrate the changes into the main branch and make them part of the production codebase.



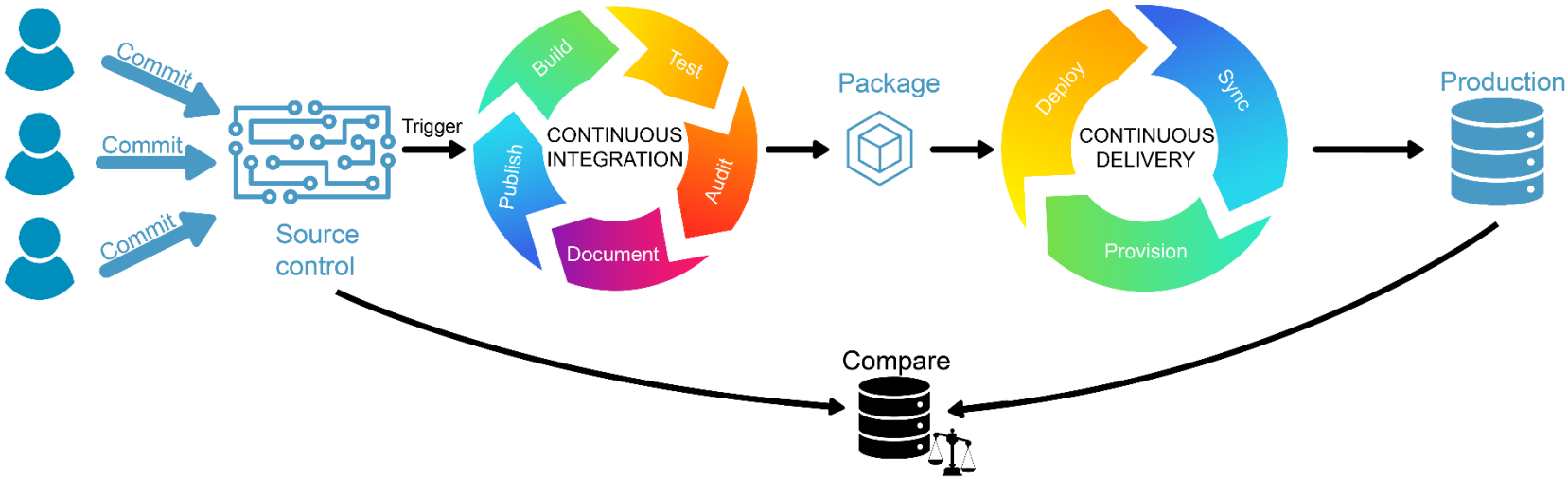
Demo – Scenario 3

You have a new requirement to:

- add a table called "Category"
- modify an existing stored procedure
- delete the "PurchaseOrderDetail" table

We will follow similar steps as before, but this time we will cherry-pick the commit from Visual Studio. In case of conflicts, Azure DevOps will return an error, and the cherry-pick process will need to be performed manually from Visual Studio.

CI/CD High-level diagram



Continuos Integration (CI)

The steps in a CI pipeline are executed by an agent, which can be either a Microsoft-based agent or a Self-Hosted agent:

- Visual Studio Build -> used to build the database project (.sqlproj) within a Visual Studio solution (.sln)
- Copy Files -> used to copy the files under /Debug or /Release build configuration mode into a staging directory
- Publish artifact -> used to copy the .dacpac file to a staging directory to be further used by the CD pipeline

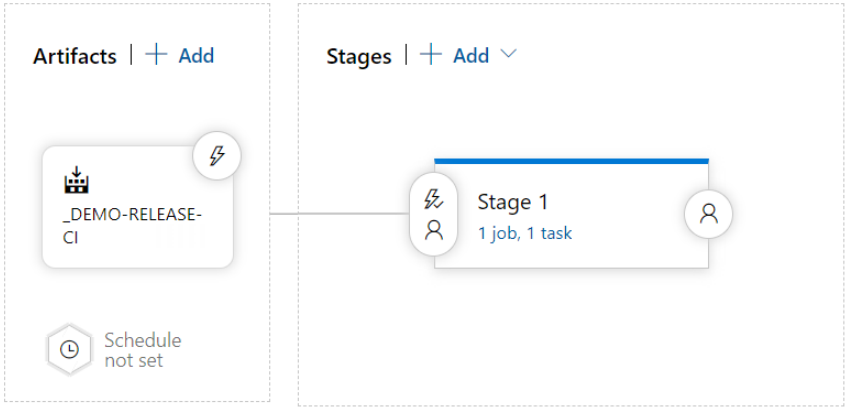
Terminology:

Demo.sqlproj.AssemblyReference.cache -> It exists under Build Configuration and it stores information about the resolved assembly references, such as their paths, versions, and other metadata. This information is used to quickly determine the references' availability and validity during subsequent builds, reducing the time required for resolving and validating the references

Build Configuration:
Debug Configuration -> Debug builds are not optimized for performance; the assembly is larger in size as it stores additional debug information in the compiled code
- Release Configuration -> used for creating a final version of the application for deployment and production use. It prioritizes performance and efficiency and it doesn't include debug information.

.dacpac file (Data-tier Application Package) -> the output of the database project build, in the form of a binary file

Continuous Deployment (CD)



1. Artifacts → It uses the .dacpac file produced by the CI pipeline

2. Stages
- *1 job* - can be either a Microsoft-based agent or a Self-Hosted agent
- *1 task* - there is a multitude of tasks that you can add, but for this session SQL Server database deploy has been used.

Demo – CI/CD

CI Pipelines:

- We will set up Azure DevOps CI pipelines for the develop, release, and main branches.
- Triggers and configuration settings will be established for each CI pipeline.
- The CI pipeline workflow will be defined to automate the build process.
- The output of these CI pipelines will be an artifact containing .dacpac, .dll, and .pdb files.

CD Pipeline:

- An Azure DevOps CD pipeline will be created to deploy changes to the SQL Production environment.
- The CD pipeline workflow will be designed to automate the deployment process.
- The input artifact for the CD pipeline will be the .dacpac file generated from the build pipeline.
- Deployment steps and configurations will be specified to ensure a successful deployment.
- The output of the CD pipeline will be the deployment of changes to the SQL Production environment.

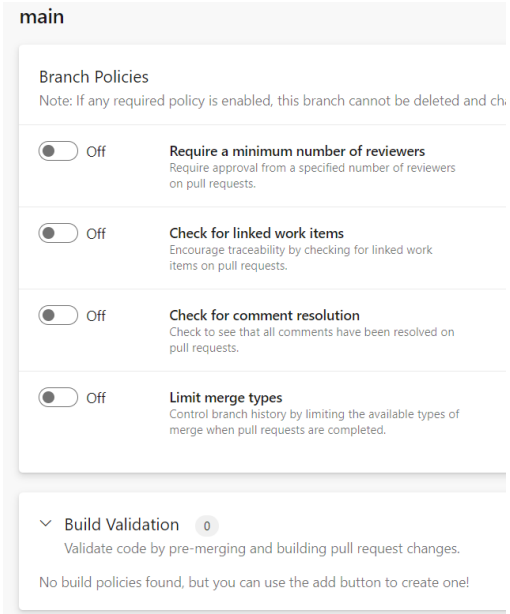
.gitignore

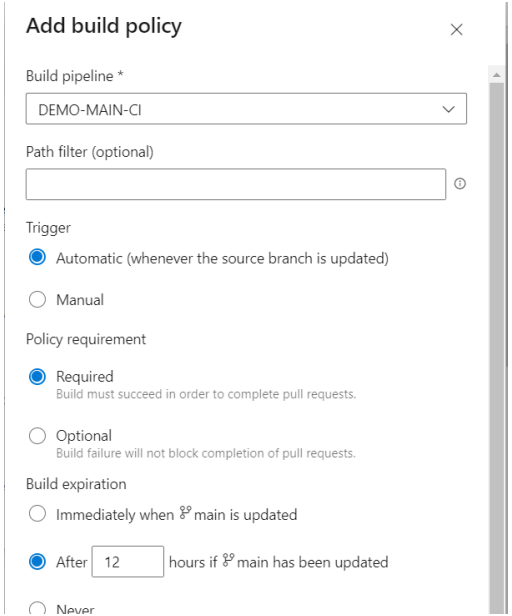
Visual Studio temporary files, build results and other files generated by VS add-ons should be added to /.gitignore:

- User-specific files – for instance .user or .suo
- Other examples:
  - .dbmdl file → it is a serialized file of your db model and is used as a cache for improving the performance of deployment. It is unique per user thus should not be checked into source control.
  - .jfm file → this is an issue caused by the ESENT engine relied on by SQL Projects adding in a new file and it should not be checked into source control.

How to check /.gitignore in Visual Studio Team Explorer → Local Git Repositories → Project → Settings → Git Repository Settings → General → Git Files → Ignore files

Branch policies





Thank you !

EVENT SPONSORS, THANKS!!!

GOLD

span

TRIA

SILVER

bonsai.tech

BRONZE

GETHYNELLIS.COM

unitfly

DATA SENSE
WHERE IT SPEAKS BUSINESS

comminus
FOR EVERY STEP OF THE WAY, THERE'S DATA

redgate

infobip

BUG

SSUGCRO

DATA
SATURDAYS

Have you seen a dragon?

Help him!