Contents

[Business Central system topology in Saas. 2](#_Toc144034116)

[What is the life cycle of a Business Central extension in Saas? 3](#_Toc144034117)

[What are Business Central’s integration patterns (names, usages)? What is the main difference between them? 4](#_Toc144034118)

[What is the Branching strategy? (Usage examples are needed) 5](#_Toc144034119)

# Business Central system topology in Saas.

# What is the life cycle of a Business Central extension in Saas?

# What are Business Central’s integration patterns (names, usages)? What is the main difference between them?

# What is the Branching strategy? (Usage examples are needed)

Branching strategy is an approach of writing, merging, and deploying code using version control system.

The purpose of the strategy is to avoid errors in the application after development and merging when there are several people who work on the same project at the same time.

This strategy also helps to plan and prepare releases and maintain the code in case of appearing bugs.

## Trunk-based development

One of the branching strategies is Trunk-based development. In this strategy, all developers are allowed to integrate their code right into the main branch. This enables teams to iterate quickly and implement CI/CD. However, they should not merge changes to the truck until they have verified that they can build successfully.

During this phase, conflicts may arise if modifications have been made since the new work began. These conflicts are increasingly complex as development teams grow and the code base scales. This happens when developers create separate branches that deviate from the source branch and other developers are simultaneously merging overlapping code. But anyway, it is allowed to create short-lived branches with a few small commits.

## GitFlow

This strategy enables parallel development where developers can work separately from the master branch on features where a feature branch is created from the master branch. Once the changes are completed, the developer merges the changes back into the master branch.

This strategy includes these branches:

* Release- to prepare a new production release.
* Master
* Develop
* Feature – for new features.
* Hotfix – also to prepare releases. The difference between this branch and the Release is that this branch arises from a bug that has been discovered and must be resolved.

The master and develop branches are the main branches, with an infinite lifetime, while the rest are supporting branches that are meant to aid parallel development among developers, usually short-lived.

This strategy is a good idea if the application has multiple versions of the production code.

However, as more branches are added, they may become difficult to manage as developers merge their changes from the development branch to the main. Developers will first need to create the release branch then make sure any final work is also merged back into the development branch and then that release branch will need to be merged into the main branch.

## GitHub Flow

This strategy is a simpler alternative to the GitFlow idea.

Unlike GitFlow, this model doesn’t have release branches. The development starts from the main branch then developers create feature branches that stem from the master. After work is done, created branches are merged back into the main. The feature branch is then deleted.

The main idea behind this model is to keep the master code in a constant deployable state and hence can support CI/CD.

Since there is no development branch you are testing and automating changes to one branch which allows for quick and continuous deployment.

This strategy is not suitable for handling multiple versions of the code.

### GitLab Flow

GitLab Flow is a simpler alternative to GitFlow that combines feature-driven development and feature branching with issue tracking.

With GitFlow, developers create a development branch and make that the default while GitLab Flow works with the main branch right away.

GitLab Flow has a pre-production branch to make bug fixes before merging changes back to the main before going to production. Teams can add as many pre-production branches as needed — for example, from main to test, from test to acceptance, and from acceptance to production.

Whenever the ‘main’ branch is ready to be deployed, users merge it into the production branch and release it. GitLab Flow is often used with release branches.