Contents

[Business Central system topology in Saas. 2](#_Toc144204386)

[What is the life cycle of a Business Central extension in Saas? 3](#_Toc144204387)

[What are Business Central’s integration patterns (names, usages)? What is the main difference between them? 4](#_Toc144204388)

[What is the Branching strategy? (Usage examples are needed) 5](#_Toc144204389)

# Business Central system topology in Saas.

Business Central online is a service that consists of a Microsoft-maintained platform and business functionality.

In BC SaaS the data is stored in Azure SQL Database and users don’t need to own and maintain on-prem servers.

An essential part of Dynamics 365 Business Central in SaaS is the Azure Active Directory. All Microsoft cloud service subscriptions, including Microsoft Dynamics 365 Business Central, are accessed via an Azure AD tenant. This Azure AD tenant is connected to a specific region of the world. Users are defined in the Azure AD and then assigned the relevant licenses in the Microsoft 365 Admin Centre. Users from the Azure Active Directory tenant can access environments (and companies) that belong to the same Azure Active Directory tenant only.

Each Azure AD that has a standard BC license has 4 environments: 1 production and 3 sandboxes. These environments can be created via the BC admin center by the administrator. Environments can be created only in regions that are available from Microsoft.

In each environment, the user is allowed to create not more than 300 business entities that are called Companies.

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

1. Development and testing.

Developers create an AL extension to meet specific business requirements. Before the clients use the extension, it should be tested, and all bugs must be resolved.

1. Packaging and Deployment.

When the development is done, the code is packaged into .app file. This file can be deployed to the BC online.

1. Installation.

The user with appropriate permissions can install the app in BC.   
Deployment and installation also depend on the type of target environment. If the app is deployed and installed in the production environment, everything can be done only via the extension management page. If the target environment is a sandbox, it is possible to install the app both via the extension management page and via VS Code.

1. Configuration.

After the extension is installed, it may require some configuration.

The needed configuration can be added both through the code and through the user interface.

1. Usage.

Users interact with the Business Central application, utilizing the functionality provided by both the base application and the installed extensions.

1. Updates and upgrades.

As far as BC online is updated on a regular basis, the extension may require some additional development to comply with the updated base code.

Also, the extension can be improved without new BC updates (adding new features or bug fixing).

1. Testing and Validation.

Before an updated extension is deployed to a production environment, it should be tested in a development or testing environment to ensure that it works as needed and does not break the processes in BC.

1. Deployment of Updates.

Once an updated extension is tested, it can be deployed to the production environment and the user can access the latest features and fixes.

1. End-of-Life.

After some time, the extension might become obsolete due to changes in business needs or improvements in the standard BC code. In such cases, the extension may be removed or replaced with a newer solution.

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

There are three ways to implement the API in the Business Central:

* REST API
* OData Web Service
* SOAP

REST APIs permit you to create any type of application in any code language that interacts with Dynamics 365 Business Central by using HTTP CRUD operations. Any coding language capable of calling REST APIs can be used to use this feature.

To use the API send the GET, POST, PUT, DELETE, and PATCH HTTP commands by using the API endpoint and the proper authentication.

REST is used when there is a need to interact with a Data Source, for example, to retrieve data for all Products. It is easier and faster to parse data using REST APIs. It uses URI to expose Business Logic.

OData is a web protocol that is designed for querying tabular data and provides you with an alternative to SOAP-based web services. OData builds on web technologies such as HTTP and JSON to provide access to information from different applications, services, and stores. OData uses URIs for resource identification and commits to an HTTP-based, uniform interface for interacting with resources.

OData is best used to expose Services and APIs to Systems and Applications. It has facilities for extension to achieve the custom needs of the REST APIs.

SOAP web services allow full flexibility for building operation-centric services. They provide industry standard interoperability. The most common type of messaging pattern in SOAP is the Remote Procedure Call (RPC), where one network node (the client) sends a request message to another node (the server), and the server sends a response message to the client. SOAP is replaced by OData V4.

The main difference between OData and REST is the Data Transfer Format. REST supports the transfer of data in any format. Although most REST APIs use XML and JSON formats, REST is not specific regarding this. OData specifies that the data should be transferred in either JSON, XML, or Atom format.

For all this protocol Pages in BC can be used for CRUD operations. At the same time, only OData (through OData unbound actions) and SOAP (which is deprecated) can be used with codeunits.

# 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.