INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




CI/CD em Dynamics 365 Business Central: Definição de uma abordagem sistemática para implementadores

BRUNO FILIPE FERNANDES ALVES
Outubro de 2022

POLITÉCNICO
DO PORTO

Instituto Superior de Engenharia do Porto

CI/CD em Dynamics 365 Business Central

Definição de uma abordagem sistemática para implementadores

Bruno Filipe Fernandes Alves

Dissertação para obtenção do Grau de Mestre em Engenharia Informática, Área de Especialização em Sistemas de Informação e Conhecimento

Orientadoras: Goreti Marreiros
Dalila Durães
Supervisora: Alice Marques

Porto, outubro 2022

Resumo

Práticas contínuas, diga-se integração, entrega e implantação contínuas, são as práticas da área de desenvolvimento de software que permitem que as organizações disponibilizem produtos e funcionalidades com frequência e confiabilidade.

A myPartner, sendo uma empresa de consultoria informática com diferentes soluções em vários clientes, tem todo o interesse em explorar estas práticas e como estas podem melhorar a produtividade e qualidade dos seus serviços.

Com dezenas de clientes em diversos setores do mercado e realidades de negócio variadas, a myPartner tem a necessidade de manter uma abordagem consistente e sistematizada nos vários projetos implementa.

Este documento, realizado no âmbito de uma dissertação de mestrado em Engenharia Informática, tem como objetivo a definição de uma abordagem sistemática para implementação das técnicas e metodologias de *Continuous Integration/Continuous Delivery* (CI/CD), a aplicar em projetos Dynamics 365 Business Central por empresas de serviços de consultoria informática, como a myPartner.

CI/CD permite a automatização de vários processos que, atualmente, são feitos manualmente, aumentando a produtividade e diminuindo o risco de erro humano.

Este documento apresenta uma solução para a aplicação destas práticas em projetos de implementação do Business Central, desde a definição dos novos processos de workflow a serem adotados pelas equipas de desenvolvimento, à implementação das várias componentes que constituem a solução.

Palavras-chave: CI/CD, Dynamics 365 Business Central

Abstract

Continuous practices, i.e., continuous integration, delivery, and deployment, are software development practices that enable organizations to make products and features available frequently and reliably.

Being a software consulting company with different solutions for multiple clients, myPartner would benefit from exploring these practices and how they can improve the productivity and quality of its services.

With dozens of clients in different market sectors and business contexts, myPartner has the need to maintain a consistent and systematic approach in the various projects it implements.

This document, carried out within the scope of a master's thesis in Computer Engineering, aims to define a systematic approach for the implementation of Continuous Integration/Continuous Delivery (CI/CD) techniques and methodologies, to be applied in Dynamics 365 Business Central projects by consulting services companies such as myPartner.

CI/CD allows for the automation of several processes that are currently done manually, increasing productivity, and reducing the risk of human error.

This document proposes a solution for the application of these practices in Business Central implementation projects, from the definition of new workflow processes to be adopted by the development teams, to the implementation of the various components that make up the solution.

Keywords: CI/CD, Dynamics 365 Business Central

Agradecimentos

A realização desta dissertação foi possível graças ao apoio incondicional dos meus pais e familiares que sempre me incentivaram a concluir o curso.

Quero também agradecer à myPartner e aos meus colegas de trabalho com quem troquei ideias e tirei dúvidas ao longo do projeto, também um agradecimento especial à Alice Marques, Miguel Oliveira e Rui Cardoso pela sugestão do tema, supervisão e acompanhamento do projeto.

É também devido um agradecimento às docentes orientadores, a professora Dalila Durães e a professora Goreti Marreiros pela orientação e apoio prestado à elaboração da dissertação.

Por último, quero agradecer a todos os colegas e amigos com quem tive oportunidade de trabalhar ao longo do mestrado nas diferentes cadeiras do curso, que também ajudaram a tornar possível a minha posição atual.

Índice

Lista de Figuras

Lista de Tabelas

Acrónimos e Símbolos

Lista de Acrónimos

AHP	Analytic Hierarchy Process
AL	Linguagem AL
API	Application Programming Interface
Azure AD	Azure Active Directory
BC	Microsoft Dynamics 365 Business Central
BD	Base de Dados
BI	Business Intelligence
CI/CD	Continuous Integration/Continuous Delivery
CRM	Customer Relationship Management
DT	Dívida Técnica
ERP	Enterprise Resource Planning
IDE	Integrated Development Environment
IoT	Internet of Things
ISEP	Instituto Superior de Engenharia Informática
MEI	Mestrado em Engenharia Informática
NAV	Microsoft Dynamics NAV
NCD	New Concept Development
Npm	Node Package Manager
NST	NAV Service Tier
RC	Razão de Consistência
S2S	Service-to-Service
SaaS	Software as a Service
SAS	Shared Access Signature

SQL	Structured Query Language
SWOT	Strengths, Weaknesses, Opportunities, Threats
TBD	Trunk-based Development
UAT	User Acceptance Test
UI	User Interface
URI	Uniform Resource Identifier
VCS	Version Control System
VPN	Virtual Private Network
VS Code	Visual Studio Code
XML	Extensible Markup Language

1 Introdução

Neste capítulo será feito um enquadramento ao projeto em questão e realizada uma descrição e contextualização do problema, assim como uma curta introdução às várias ferramentas e tecnologias envolvidas. Serão também definidos os objetivos que se pretende atingir com este projeto.

Este documento está estruturado em 7 capítulos: Introdução; Contexto e Estado da arte; Análise de Valor; Design; Implementação; Avaliação e Conclusão.

Na introdução (1), foi feito um enquadramento, uma descrição do problema e delineados os objetivos principais do projeto.

De seguida, é apresentado o Contexto e Estado da Arte (2) onde é feito um levantamento dos principais conceitos, tecnologias e ferramentas relevantes ao projeto, e um estudo das principais abordagens existentes no que diz respeito a implementação de práticas de Continuous Integration/Continuous Deployment (CI/CD) em projetos quer de Business Central, quer de outros âmbitos.

No terceiro capítulo (3) é feita uma análise de valor ao projeto em mãos, onde se avaliou o valor que este projeto trará à empresa.

Passando da fase de estudo à fase de criação, no capítulo 4 é apresentado o desenho da solução implementada. Isto inclui a definição dos processos, ferramentas a utilizar e a arquitetura da solução.

Seguidamente, no capítulo 5, é descrito como foi implementada a solução, passo a passo, decisões tomadas e desenvolvimentos realizados.

Para avaliar a solução, no capítulo 6, Avaliação, foram verificados se os objetivos iniciais foram cumpridos com a solução desenvolvida, e avaliado se esta se adequa ao âmbito do projeto.

Finalmente, o capítulo 7 é a conclusão onde é feita uma reflexão sobre os objetivos alcançados, problemas e limitações da solução desenvolvida e trabalho futuro a realizar.

Ao longo deste documento, todos os termos técnicos serão referidos na sua denominação em inglês de modo a uniformizar e facilitar a leitura e compreensão dos conceitos.

1.1 Enquadramento

Este projeto enquadra-se no âmbito da unidade curricular Tese/Dissertação do Mestrado em Sistemas de Informação e Conhecimento do curso de Engenharia Informática do ISEP.

O projeto surge no contexto da empresa myPartner, Consultoria Informática S.A. que é uma empresa de consultoria informática parceira da Microsoft que, como o nome sugere, presta serviços de consultoria informática a clientes (empresas) envolvidos em diversos setores do mercado como a moda, hotelaria, indústria, alimentação, entre outros.

Estes serviços incluem essencialmente a implementação, customização e manutenção de soluções Microsoft nas áreas de ERP, CRM e BI.

O objetivo deste projeto prende-se apenas a implementações de projetos de ERP. O software ERP no qual a myPartner se especializa é o Dynamics 365 Business Central (BC), anteriormente conhecido como Microsoft Dynamics NAV (NAV).

No universo do BC, existem essencialmente 2 áreas em que as empresas se especializam:

- **Serviços:** Empresas de serviços, entre as quais a myPartner se inclui, assistem os seus clientes na implementação, customização, utilização e manutenção do BC. Aqui, cada projeto diz respeito a um só cliente e a solução final é o ambiente de produção desse cliente. É a estas empresas que o termo "implementador" se refere, já que o principal serviço prestado é a implementação do ERP em clientes.
- **Produto:** Empresas de produto desenvolvem novas funcionalidades, que posteriormente vendem aos seus clientes como uma expansão às funcionalidades base do ERP. Aqui, uma solução final será um Add-on que tipicamente se foca numa área de negócio específica e que será vendido a clientes que operam nessa área de negócio.

Muitas empresas especializam-se em ambas as áreas tendo uma componente de prestação de serviços e outra de desenvolvimento de produto.

Com a transição do NAV para BC em 2018, a metodologia de desenvolvimento e customização de funcionalidades mudou drasticamente. Tendo a mudança mais significativa a ver com a gestão do código fonte.

Em NAV, o código era desenvolvido diretamente no ambiente de desenvolvimento (IDE) integrado do NAV, que não fazia distinção entre o código base e o código customizado e guardava diretamente todas as alterações na base de dados, fazendo a gestão do código pelos

developers. Mas o BC introduziu o conceito de extensões, que isola o código base do código customizado.

Uma extensão representa uma expansão das funcionalidades base do ERP, permitindo customizá-lo de acordo com as necessidades dos clientes e diminuindo o risco de impacto nas funcionalidades de base (Active BS, 2022).

1.2 Problema

Com esta alteração de paradigma, os developers têm agora de se preocupar com a gestão do código das suas extensões, e as empresas implementadoras têm de definir processos novos no que diz respeito ao desenvolvimento, testagem e deployment das suas customizações, que agora tomam a forma de extensões.

Nos projetos BC da myPartner, distinguem-se essencialmente 2 fases:

- **Implementação**: Nesta fase, o projeto ainda está em fase de desenvolvimento e o cliente ainda utiliza o sistema antigo em produção.
- **Suporte**: Nesta fase, o cliente já utiliza o BC em ambiente de produção, mas é normal surgirem novos pedidos de customização ou erros que necessitam de intervenção técnica.

Neste contexto, ganha relevância o conceito de CI/CD.

Enquanto já existem metodologias definidas para a aplicação de CI/CD em projetos BC, aplicadas com recurso ao Azure Devops, estas dizem apenas respeito ao desenvolvimento de produto. É, portanto, necessário avaliar de que forma é que as metodologias já definidas podem ser adaptadas à realidade da myPartner.

Neste sentido, levantam-se questões sobre:

- como deverá ser feita a gestão dos repositórios de cada projeto?
- como devem ser definidas as configurações de branching?
- que pipelines devem ser desenvolvidas?
- como serão realizados testes manuais e automáticos?
- Como serão feitas reposições de backups?
- Como será feito o deployment das funcionalidades para ambiente produtivo?

1.3 Objetivos

Este documento pretende responder às questões enumeradas no subcapítulo anterior e definir uma abordagem sistematizada de aplicação dos conceitos CI/CD em projetos BC que possa ser aplicada nos vários clientes da myPartner.

Para isso, existe uma série de objetivos principais que se pretendem alcançar com este projeto:

- Inicialmente, deverá ser realizado um levantamento do estado da arte dos conceitos e metodologias de CI/CD existentes direcionadas a projetos de BC;
- Com as informações obtidas deverá ser feita uma análise de como se podem adaptar estas soluções ao contexto da myPartner e/ou definir novos processos caso haja necessidades não correspondidas pelas soluções existentes.
- Após esta análise, pretende-se que seja definida a abordagem que a myPartner deverá aplicar nos seus projetos BC, tendo em consideração todas as questões acima enumeradas.
- Por fim pretende-se a criação de um projeto de protótipo em Azure DevOps com a estrutura definida na análise e onde deverão ser aplicadas as regras de branching estabelecidas e criadas as pipelines que ajudarão a automatizar os processos delineados.

Existem, portanto, 4 objetivos sequenciais que podem ser resumidos como: estudo, análise, definição/desenho e protótipo.

Estes objetivos não são particularmente detalhados devido ao facto de a solução final pretendida não ter ainda um conjunto de requisitos bem definidos, sendo o objetivo final mais uma "prova de conceito" do que uma solução pronta a ser utilizada pela empresa.

É importante referir também que existem 2 tipos de ambientes quando se fala em instalações de BC em clientes:

- **SaaS (Software as a Service)**: Ambiente alojado na cloud - online.
- **OnPrem**: Ambiente alojado localmente - servidores locais.

Por se tratar de um projeto com características de protótipo e para evitar demasiada complexidade, o presente documento focar-se-á em projetos SaaS.

2 Contexto e Estado da Arte

Neste capítulo será feita uma introdução às ferramentas e softwares relevantes, explicando o que são, para que servem e como funcionam. Será também feita uma introdução a alguns conceitos e termos importantes à interpretação do tema e, por fim, um levantamento do estado da arte dos conceitos e metodologias existentes que dizem respeito à aplicação de Continuous Integration/Continuous Delivery (CI/CD) quer em projetos de Business Central (BC), como noutro tipo de projetos, analisando as soluções existentes.

2.1 Linguagens de programação

Neste subcapítulo são apresentadas as linguagens relevantes e que poderão ser necessárias para a realização deste trabalho.

2.1.1 Windows Powershell

Windows PowerShell é uma linguagem de linha de comandos para o Microsoft Windows usada tipicamente para administração de sistemas. É uma solução de automação de tarefas multiplataforma composta por uma consola de linha de comandos, uma linguagem de script e uma estrutura de gestão de configuração destes scripts. O PowerShell pode ser executado em Windows, Linux e macOS.

Como linguagem de script, o PowerShell é tipicamente utilizado para automatizar a administração de sistemas. Também é utilizado para criar, testar e implantar soluções em ambientes de CI/CD. (Microsoft, 2022)

O código-fonte básico do PowerShell está disponível no GitHub e aberto a contribuições da comunidade.

Um conceito importante na linguagem de PowerShell são os módulos. Um módulo é essencialmente um pacote de scripts de PowerShell adaptados para contextos específicos.

Os programadores (ou developers) que desenvolvem scripts podem usar módulos para organizar os seus comandos e compartilhá-los com outros developers. Developers que recebem módulos podem utilizar os comandos desses módulos nas suas sessões de PowerShell e usá-los como os comandos internos. (Microsoft, 2022b)

Existem 2 módulos essenciais que são automaticamente importados ao instalar o Business Central, são eles:

- **Business Central Administration Shell:** inclui cmdlets (comandos PowerShell) para administrar a implantação do Business Central, como adicionar e configurar instâncias de servidor, bases de dados e utilizadores do BC. Também estão incluídos cmdlets para administrar pacotes de extensão.
- **Business Central Development Shell:** inclui cmdlets para mesclar e modificar ficheiros de objetos da aplicação. Também estão incluídos cmdlets para criar pacotes de extensão. (NavisionPlanet,2022)

A Figura 1 mostra um exemplo de uma série de cmdlets de PowerShell no contexto do desenvolvimento de Business Central.

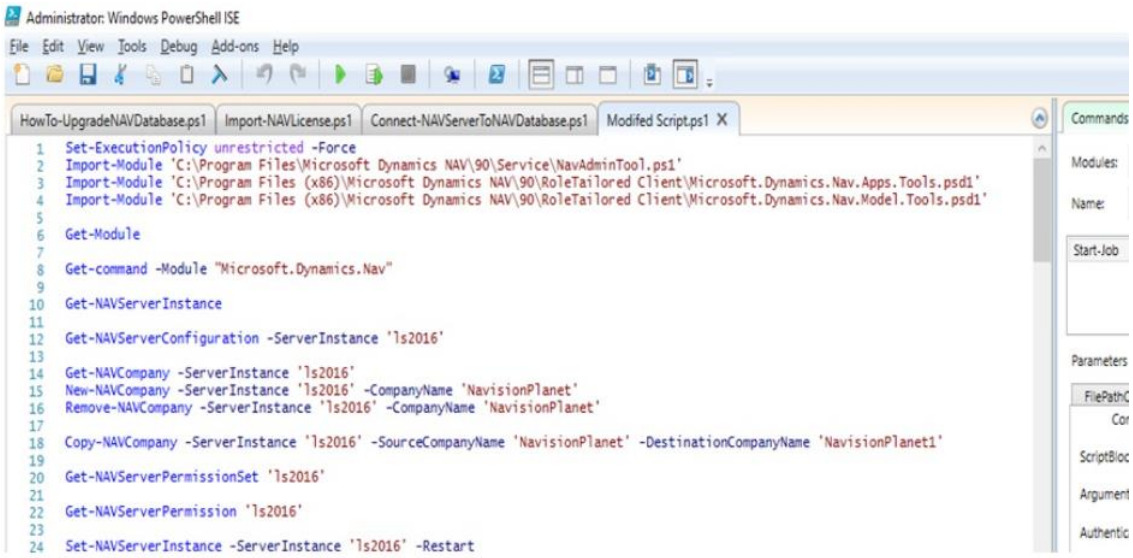


Figura 1: Linguagem PowerShell (NavisionPlanet,2022)

Um outro módulo, desenvolvido especificamente para facilitar e automatizar muitos dos processos mais comuns no desenvolvimento de aplicações de BC é o módulo *BcContainerHelper*. Este módulo não é importado automaticamente com o Business Central, e deve ser instalado separadamente para poder ser utilizado.

Este módulo contém funções para as mais diversas necessidades, desde a criação de ambientes de BC para desenvolvimento e a sua manutenção, como a criação de pipelines direcionadas também a projetos de BC.

Na Figura 2 é apresentada parte da lista de funções incluídas no módulo *BcContainerHelper*, salientando a função *Run-AlPipeline*.

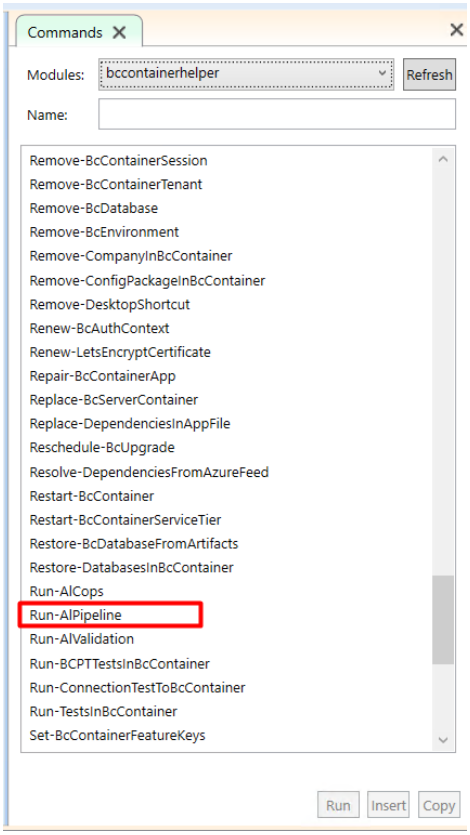


Figura 2: Módulo BcContainerHelper

Run-AlPipeline é uma função no *BcContainerHelper* que tem como objetivo ser um mecanismo consistente para permitir compilar aplicações de Business Central (Freddy Kristiansen, 2020) e pode ser uma mais-valia na criação de pipelines neste projeto.

2.1.2 YAML

YAML é um formato de serialização de dados projetado para legibilidade humana e interação com linguagens de script (como PowerShell). O YAML é otimizado para serialização de dados, definição de configurações, arquivos de log e filtragens. É amplamente útil para necessidades de programação que variam de arquivos de configuração a mensagens da Internet, persistência de objetos e auditoria de dados (YAML, 2001). A Figura 3 apresenta um exemplo de um ficheiro YAML que executa um ficheiro PowerShell.

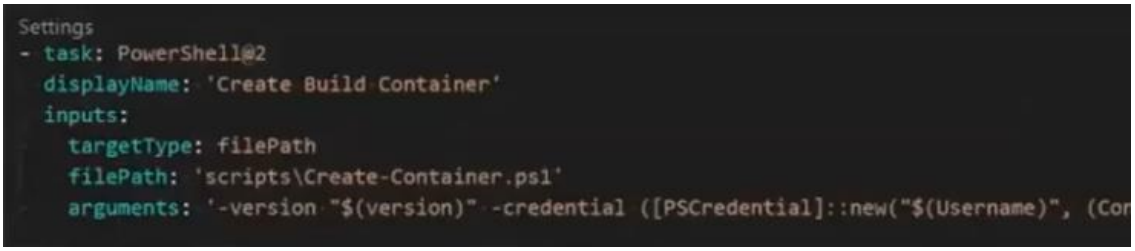


Figura 3: Exemplo de ficheiro YAML que executa um script PowerShell

2.2 Ferramentas e Softwares

2.2.1 Dynamics 365 Business Central

O Microsoft Dynamics 365 Business Central (BC) é um sistema de Enterprise Resource Planning (ERP) da Microsoft e é o sucessor do Microsoft Dynamics NAV, tipicamente referido como NAV. O produto faz parte da família Microsoft Dynamics 365, que é uma linha de produtos direcionados a pequenas e médias empresas.

Como qualquer software ERP, o BC assiste estas empresas na gestão de finanças, compras e vendas, produção, distribuição, marketing, relação com clientes, entre outras funcionalidades. O produto possibilita funções base comuns à maioria dos ramos de negócio, mas, de grosso modo, o público-alvo está em empresas que procuram mais funcionalidades do que aquelas que o software oferece por defeito. A maioria das instalações é vendida através de mais de 2.200 parceiros da Microsoft em mais de 100 países diferentes (Veldkamp, 2020) que as personalizam às necessidades de cada cliente, sendo a myPartner um desses parceiros.

Inicialmente, o desenvolvimento em NAV era feito num ambiente integrado na interface do utilizador e ao qual este tinha acesso, o C/SIDE (Microsoft, 2017). A partir da versão 2013 o ambiente de desenvolvimento passou a ser um componente separado da aplicação entregue ao cliente com o nome *Dynamics NAV Development Environment*. Com o lançamento do Business Central em 2018, a forma de desenvolvimento neste ERP mudou substancialmente.

Uma mudança significativa foi a própria linguagem de programação, que deixou de ser o C/AL, e passou a ser AL. De grosso modo, AL é muito parecida com C/AL, estando as principais diferenças nas funcionalidades mais específicas em que AL oferece mais opções e usabilidade.

Uma das mais importantes mudanças está na forma como o código base é disponibilizado. Enquanto em NAV não era feita qualquer distinção entre código base e código customizado, em BC o código base está inacessível aos developers e estes devem desenvolver extensões isoladas para customizar o ERP, estando dependentes de um tipo de funções específicas denominadas de eventos, que a Microsoft disponibiliza em partes do código base, permitindo aos developers chamar estas funções de modo a alterar comportamentos standard (de raiz).

Estas extensões são tipicamente desenvolvidas no Visual Studio Code (VS Code) e necessitam de estar conectadas a uma base de dados de desenvolvimento para poderem ser compiladas.

Com o VS Code, existe também um leque de extensões (do VS Code) específicas para desenvolvimento em AL e outras que permitem integrações diretas com o Docker e repositórios Git.

2.2.1.1 Extensões de Business Central

As extensões de BC são um modelo de programação em que a funcionalidade é definida como uma adição aos objetos existentes e define como eles são diferentes ou modificam o comportamento da solução.

Todas as funcionalidades do Business Central são codificadas em objetos. O modelo de extensão é baseado em objetos. O programador cria novos objetos e estende objetos existentes dependendo do que deseja que sua extensão faça. (Microsoft, 2022f)

Objetos de tabela, por exemplo, definem o esquema das tabelas que contém dados. Objetos de página, por outro lado, representam as páginas apresentadas na interface do utilizador. Objetos de *Codeunit* contêm código para cálculos lógicos e para o comportamento da aplicação.

Esses objetos são armazenados como código escrito em linguagem AL, e são guardados em ficheiros com a extensão ".al".

2.2.1.2 Linguagem AL

Como já foi mencionado, o Business Central tem a sua própria linguagem de programação. O AL é uma linguagem orientada a objetos e que é exclusivamente utilizada para o desenvolvimento de extensões para BC, permitindo a alteração de funcionalidades existentes e a criação de novas, e com funções específicas que despoletam *queries* no SQL, permitindo a manipulação dos registos da base de dados do ERP.

Em BC, o código programado em AL é guardado em objetos que, por sua vez, existem numa base de dados. Isto significa que toda a lógica de negócio e customizações feitas ao ERP são guardadas na própria base de dados do ERP. Os objetos mais importantes são:

- **Table:** representa uma tabela física na base de dados (BD), e contém *triggers* que permitem adicionar lógica de negócio quando determinadas ações são realizadas (Ex.: ao adicionar um registo, ou alterar um campo).
- **Table Extension:** permite adicionar código ou alterar/adicionar novos campos a uma tabela existente.
- **Page:** é o objeto com o qual o utilizador interage diretamente na interface de utilizador (UI) do ERP, contendo campos, botões e ações com lógica de negócio associada.
- **Page Extension:** permite alterar/adicionar funcionalidades a uma page existente.

- **Report:** é utilizado para imprimir/visualizar informação da base de dados (Ex.: faturas, notas de crédito, balancetes, etc.)
- **Codeunit:** é um contentor de código AL que é normalmente invocado por outros objetos. É nas codeunits que se encontra grande parte da lógica de negócio do ERP.
- **Query:** permite a consulta de uma ou várias tabelas e combinar esses dados num só *dataset*, permitindo também operações simples como encontrar a soma ou média de uma coluna numérica.
- **XMLPort:** é utilizado para importar e exportar dados entre fontes externas ao Business Central na forma de ficheiros de XML.

O desenvolvimento em AL implica sempre a criação/alteração destes objetos, sendo através deles que se manipulam os dados, a UI, e toda a lógica de negócio do ERP.

2.2.1.3 Arquitetura Multi-Tenant do Business Central

O Business Central permite que várias empresas diferentes acedam a uma aplicação de BC mantida centralmente. Ao usar esse suporte de Multi-Tenant, é possível adicionar novos clientes à solução com facilidade e distribuir atualizações rapidamente com tempo de inatividade reduzido para os clientes. (Microsoft, 2022e)

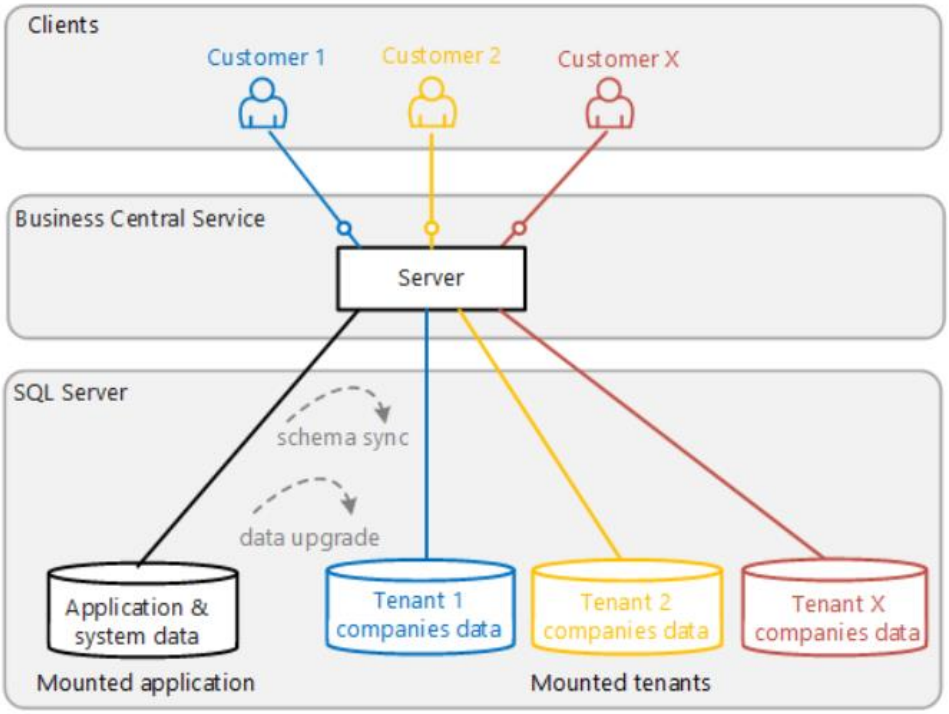


Figura 4: Arquitetura Multi-Tenant do Business Central (Microsoft, 2022e)

Numa implantação Multi-Tenant, os dados da aplicação Business Central são armazenados numa BD de aplicação específica. Os dados dos clientes são armazenados em BDs comerciais separadas, cada uma das quais é um *tenant* na implantação. Ao separar a aplicação dos dados,

é possível implantar a mesma solução para vários clientes com manutenção centralizada da aplicação e isolamento de cada *tenant*. (Microsoft, 2022e)

Esta é o tipo de implantação adotada por empresas que desenvolvem soluções de BC para venda e distribuição a clientes que não desejam ter o BC alojado nos próprios servidores (on-Prem), mas sim utilizar a aplicação como serviço (SaaS).

Não é obrigatório que uma implantação do BC seja do tipo Multi-Tenant. É possível instalar e executar o Business Central como uma implantação clássica de apenas um servidor e uma base de dados. Este é o tipo de solução que a grande maioria dos clientes da myPartner adotam.

2.2.1.4 RapidStart

O RapidStart é uma ferramenta do Business Central que permite aos utilizadores extrair, importar e atualizar dados do sistema. Podem ser usados para criar configurações, como pacotes de configuração RapidStart ou para carregar grupos de registo, planos de contas ou lançamentos de diário.

O RapidStart é tipicamente usado para extrair dados históricos durante uma implementação, podendo depois serem importados no novo sistema em caso de upgrade, ou como recuperação de dados corrompidos. É também utilizado durante o processo de desenvolvimento de modo a preparar conjuntos de dados de teste que podem ser importados nos ambientes de desenvolvimento e teste das equipas de desenvolvimento.

2.2.2 Azure DevOps

DevOps é um conceito da área do desenvolvimento de software que se refere às filosofias culturais, práticas e ferramentas que aumentam a capacidade de uma organização de planear, desenvolver e fornecer aplicações e serviços com rapidez e qualidade (Amazon, 2022).

O Azure DevOps é um produto da Microsoft que se foca nestes conceitos, oferecendo serviços de controlo de versões, relatórios, gestão de requisitos, gestão de projetos, compilações automatizadas, testes e recursos de gestão de lançamento, cobrindo todo o ciclo de vida de uma aplicação. O Azure DevOps pode ser usado como *back-end* para vários ambientes de desenvolvimento integrado (IDEs), estando especialmente adaptado para Microsoft Visual Studio e Eclipse (Microsoft,2017b).

Estes serviços podem ser utilizados localmente (*on-premises*) com o Azure DevOps Server, ou na *Cloud* com o Azure Devops Services e podem ser acedidos através de um web browser ou de um cliente IDE.

Os serviços oferecidos pelo Azure DevOps incluem:

- **Azure Repos:** fornece repositórios Git ou Team Foundation Version Control (TFVC) para controlo de versões.

- **Azure Pipelines:** fornece serviços de compilação e lançamento para dar suporte a integrações e entregas contínuas (CI/CD) das aplicações.
- **Azure Boards:** oferece um conjunto de ferramentas ágeis para dar suporte ao planeamento e gestão de trabalho, bugs de código, e problemas usando os métodos Kanban e Scrum.
- **Azure Test Plans:** fornece várias ferramentas para testar as aplicações, incluindo testes manuais e testes contínuos automatizados.
- **Azure Artifacts:** permite a gestão e partilha de pacotes como Maven, npm, Python, NuGet, entre outros, e integrá-los nas suas pipelines.

Outras ferramentas incluem: *dashboards* de equipa customizáveis para partilha de informações, progresso e tendências; *Wikis* integrados para partilha de informações; Notificações configuráveis.

O Azure Devops também suporta extensões que permitem a integração com outros serviços como Campfire, Slack, Trello, UserVoice, entre outros. E também permite a integração com o GitHub (Chcomley, 2022).

Para o âmbito de projetos de BC, existe a extensão ALOps. ALOps é uma extensão para Azure DevOps desenvolvida pela empresa Hodor. Esta extensão oferece ferramentas para construção de pipelines em projetos de BC.

As funcionalidades do ALOps estão definidas em scripts de PowerShell que são invocados através de ficheiros YAML.

Algumas das suas funcionalidades incluem:

- Manutenção de agentes
- Compilar extensões
- Publicar de extensões
- Correr testes automáticos

2.2.3 GitHub

O GitHub, também da Microsoft, é uma plataforma em nuvem que usa o Git como a sua tecnologia principal. É utilizado para simplificar o processo de colaboração em projetos através do seu site, ferramentas de linha de comando e fluxo geral que permite que developers e utilizadores trabalhem juntos. O GitHub atua como repositório remoto para soluções de software. (Microsoft, 2022g)

O GitHub oferece várias funcionalidades que são úteis, mas que não são relevantes para este projeto. No entanto, existem funcionalidades que são relevantes, como o facto de suportar a implementação das práticas de CI/CD nos repositórios alojados na plataforma.

2.2.4 Visual Studio Code

O VS Code é um editor de código-fonte desenvolvido pela Microsoft que é leve, mas poderoso e que corre localmente na máquina do utilizador, estando disponível para Windows, macOS e Linux.

O VS Code inclui recursos de depuração, IntelliSense, snippets, refatoração de código e Git incorporado. Suporta também a instalação de extensões que permitem a integração com outros serviços, incluindo Azure DevOps.

Existem várias extensões criadas para o desenvolvimento em AL, sendo algumas necessárias para poder programar na linguagem, e outras opcionais cujo objetivo é melhorar a "qualidade de vida" do developers.

Algumas das extensões de VS Code utilizadas no desenvolvimento de extensões para BC são (Dvernytskyi, 2020):

- **AZ AL Dev Tools/AL Code Outline:** é um conjunto de ferramentas úteis numa só extensão. Estas ferramentas incluem:
    - Wizards para criação de objetos em AL;
    - Geradores de código;
    - Ações de código;
    - Suporte para comentários de documentação;
- **AL Object Designer:** Permite a visualização e navegação dos objetos (tables, pages, codeunits, etc) em forma de lista à semelhança do C/SIDE.
- **GitLens:** Contém um conjunto de funcionalidades que facilitam a integração com o repositório Git onde as extensões são guardadas.
- **AL Language Tools:** contém ferramentas úteis para criação de testes automatizados, traduções e reports.
- **Waldo's CRS AL Language Extension:** Pack de ferramentas e snippets customizados para facilitar a aplicação de boas práticas no desenvolvimento em AL.
- **AL Extension Pack:** Conjunto de outras extensões úteis para desenvolvimento em AL (Waldo, 2021)

2.2.5 Docker

O Docker é um conjunto de produtos de plataforma como serviço que usam virtualização ao nível do sistema operacional para entregar software em pacotes chamados contentores. (Maureen O'Gara, 2013)

Contentores podem ser vistos como máquinas virtuais leves que permitem configurar um ambiente computacional, incluindo todas as dependências necessárias (por exemplo, bibliotecas), configuração, código e dados necessários, dentro de uma única unidade (chamada imagem). As etapas necessárias para atingir o estado em tal imagem são documentadas num

Dockerfile, um script que contém todas as configurações e comandos da infraestrutura. As imagens podem ser distribuídas publicamente e executadas sem problemas e também têm suporte para os principais sistemas operacionais por meio da máquina Docker. A principal diferença para as máquinas virtuais é que as imagens do Docker compartilham o *kernel* com a máquina onde o contentor está a ser executado, o que permite tamanhos de imagem muito menores e desempenho superior. (Jurgen et al., 2016)

A Figura 5 exemplifica visualmente como esta arquitetura é estruturada na máquina anfitriã (leia-se, a máquina sobre a qual o contentor está montado).

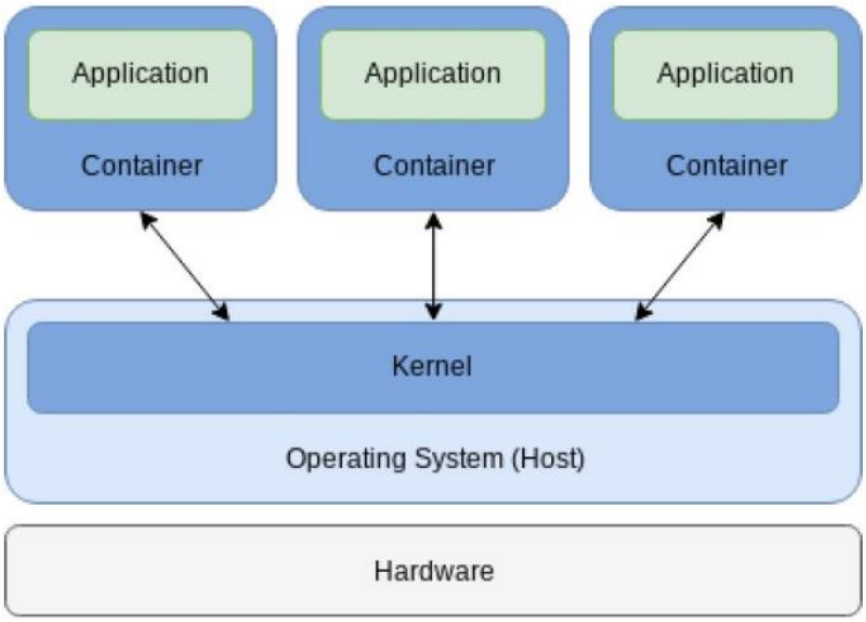


Figura 5: Arquitetura de funcionamento dos contentores Docker (Scolati et al., 2019)

Tudo o que é preciso para executar uma determinada aplicação está na imagem. As imagens do Docker são armazenadas num registo do Docker onde os utilizadores podem fazer upload e download de imagens. O registo mais conhecido é o Docker Hub, mas para o Business Central (e outros produtos da Microsoft), a Microsoft hospeda o seu próprio registo. (Microsoft, 2022d)

Convenientemente, o Docker oferece imagens para todas as versões lançadas em todos os países. (Microsoft, 2022c)

Cada imagem é construída sobre uma imagem base que contém os arquivos do sistema operacional. Os contentores do Windows têm quatro imagens de base:

- **Windows Server Core** - Suporta aplicações .NET Framework
- **Nano Server** - Criado para aplicações .NET Core
- **Windows** - Fornece o conjunto completo de APIs do Windows
- **Windows IoT Core** - Criado para aplicações de IoT (Internet of Things)

Uma imagem do Docker é geralmente confundida com uma máquina virtual, mas não são a mesma coisa. Uma imagem não precisa de um sistema operacional completo para ser executada porque usa o *kernel* do sistema operacional anfitrião. Além disso, uma imagem não tem interface gráfica e não é possível conectar-se à imagem via *Remote Desktop*.

Ao executar uma imagem, é criada uma instância dessa imagem. Uma instância de uma imagem é chamada de contentor. É possível instanciar vários contentores da mesma imagem e todos armazenarão as alterações em arquivos diferentes não interferindo entre si.

É comum, em projetos de software, ocorrem situações em que, após a instalação em um ambiente de produção, ocorrem erros que não estavam presentes no ambiente de desenvolvimento.

O Docker permite ter o mesmo ambiente no processo de desenvolvimento e simplesmente implantar esse ambiente em produção. (Microsoft, 2022d)

2.3 Conceitos e termos

Nesta secção são introduzidos todos os conceitos e termos importantes para a contextualização e compreensão dos temas que serão abordados. Optou-se por apresentar este subcapítulo depois das ferramentas porque alguns dos conceitos aqui destacados dizem respeito às ferramentas abordadas no subcapítulo anterior.

2.3.1 Continuous Integration/Continuous Delivery

Continuous Integration/Continuous Delivery (CI/CD) representa um conjunto de metodologias e filosofias de desenvolvimento de software que agilizam o processo de integração e entrega das aplicações aos clientes. Os conceitos associados a CI/CD são Continuous Integration, Continuous Delivery e Continuous Deployment.

Continuous Integration (CI) é uma prática em que o código é compilado e testado de cada vez que um developer submete alterações para o repositório da aplicação. Ou seja, quando uma alteração é lançada para o repositório, é iniciado um conjunto de processos automatizados que compilam a solução com as novas alterações e executam os testes automatizados definidos, ajudando a prevenir bugs e a garantir a integridade da aplicação.

Esta prática torna-se ainda mais relevante quando uma aplicação está a ser desenvolvida por uma equipa com vários elementos, todos trabalhando no mesmo código e submetendo alterações simultaneamente.

A sigla CD refere-se tanto a Continuous Delivery como a Continuous Deployment que são conceitos relacionados e por vezes intercambiáveis.

CD geralmente significa que as alterações de um developer na aplicação são automaticamente testadas e carregadas para o repositório onde podem ser implantadas num ambiente de produção. É uma resposta para o problema de pouca visibilidade e comunicação entre as equipas de desenvolvimento e de negócios. Para esse fim, o objetivo da entrega contínua é garantir que seja necessário um esforço mínimo para a entrega das novas alterações efetuadas à aplicação (Red Hat, 2018).

Continuous Deployment pode referir-se à disponibilização automática das alterações de um developer do repositório para o ambiente de produção, onde podem ser usadas pelos clientes. Isto ajuda a mitigar as demoras que surgem dos processos geralmente manuais necessários para efetuar estas tarefas. Este processo pode ser visto como uma continuação do CD em que para além de disponibilizar as alterações para o repositório, estas são também disponibilizadas para o ambiente de produção.

Figura 6: Como os 3 conceitos associados a CI/CD estão relacionados (Red Hat, 2018)

Na Figura 6 é possível visualizar os conceitos de CI/CD, e as 3 componentes constituintes. No entanto, a componente de Continuous Deployment é opcional, sendo que muitas vezes apenas são aplicadas as duas primeiras.

2.3.2 Task/Tarefa

Uma tarefa é um bloco de construção que define a função de uma pipeline. Uma tarefa pode consistir simplesmente num script ou procedimento a executar a partir de um conjunto de entradas. Exemplos de tarefas são os processos de build, ou deploy.

2.3.3 Build

O termo build refere-se ao processo pelo qual o código fonte é compilado e convertido num programa que pode ser executado e utilizado pelo utilizador. Uma das etapas mais importantes na construção de um software é o processo de compilação, onde os ficheiros de código-fonte são convertidos em código executável. O processo de construção de software é geralmente realizado por uma ferramenta de build. As compilações são criadas quando um determinado ponto do desenvolvimento foi alcançado ou o código foi considerado pronto para implementação, seja para teste ou lançamento definitivo (Techopedia, 2011).

2.3.4 Release

Uma release é a distribuição da versão final ou a versão mais recente de um produto de software. Uma release de software pode ser pública ou privada e geralmente significa o lançamento de uma versão nova ou atualizada do produto.

No desenvolvimento ágil de software, ilustrado na Figura 7, uma release é um pacote de software implantável que culmina em várias iterações e pode ser feito antes do final de uma iteração.

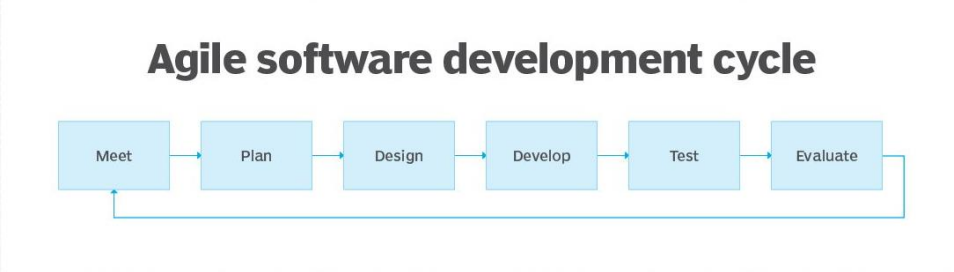


Figura 7: Ciclo do desenvolvimento ágil de Software . (Hanna, 2022)

A gestão das releases requer planeamento e execução cuidadosos, com metas centradas no cliente, eficiência de tempo e orçamento e impacto negativo mínimo nos clientes. (Hanna, 2022)

O conceito de release é especialmente relevante no contexto de distribuições de *Software as a Service* (Saas). Uma vez que nestes casos, o cliente apenas faz uma instalação inicial, e todas as atualizações subsequentes deverão ser automáticas, em contraste com os métodos de distribuição antigos. A Tabela 1 destaca estas diferenças.

Tabela 1: Métodos de distribuição de software antigos e atuais (Plutora, 2022)

Método de Distribuição (Release)	**Tempo médio entre**	**Tipo de Desenvolvimento**	**Características**
Produtos físicos (CDs, disquetes, modelo de hardware)	Anos	Waterfall	Não era possível atualizar, modificar ou monitorar. Grandes atualizações em cada versão para justificar o preço. Tinha de funcionar perfeitamente na primeira release. Planeado com bastante antecedência.
Downloads de Software (internet)	Meses ou Anos	Waterfall-Agile	O aumento da velocidade da Internet permitiu a distribuição através de downloads de software. Problemas e bugs podem ser corrigidos. Foco na melhoria incremental. O foco passa do projeto para o fluxo de valor. O feedback do cliente torna-se a base para as próximas iterações.
SaaS (Cloud storage)	Semanas ou Meses	Agile – DevOps	Resposta constante ao feedback do cliente. Subscrições mensais financiam desenvolvimentos adicionais

2.3.5 Deployment

Software Deployment, ou implantação, refere-se ao conjunto de atividades que permitem que um sistema de software fique disponível para ser utilizado.

O deployment só deve ocorrer após todos os testes terem sido completados para garantir que todas as falhas e bugs sejam identificadas e corrigidas.

No contexto do Business Central, é frequente a utilização do termo deployment ao referir-se passagens de funcionalidades entre os ambientes de desenvolvimento, testes e produção.

2.3.6 Merging

Merging refere-se ao processo de combinar as várias versões de um ficheiro ou pasta. No desenvolvimento de software, este recurso é normalmente encontrado em softwares de controlo de versão, sendo uma operação fundamental que é responsável pela unificação das alterações realizadas por diferentes developers no mesmo ficheiro (Techopedia, 2015).

2.3.7 Branching

Um recurso proeminente dos sistemas de controlo de versões é a capacidade de rastrear a evolução do software ao longo de várias linhas de código - alterações paralelas na mesma base de código. Essas linhas de código são comumente chamadas de ramificações (branching). A ramificação promove desenvolvimento e manutenção isolados; alterações feitas para uma ramificação não afetará nenhuma outra ramificação. Este recurso permite que as equipas de desenvolvimento particionem o trabalho e evitem interrupções de fontes externas. (Philips et al., 2011)

Um branch representa uma versão do código, gerada por um sistema de controlo de versões. Branching ajuda as equipas de desenvolvimento de software a trabalharem em paralelo, separando o “trabalho em andamento” do código testado e estável. (Schiestl, B., 2022)

2.3.8 Pipelines

No contexto do desenvolvimento de software, uma pipeline é um conjunto de processos automatizados que permitem aos developers e profissionais de DevOps que compilem, construam e implantem o seu código de forma confiável e eficiente entre os vários ambientes.

Os componentes mais comuns de uma pipeline são:

- Automatização do processo de build / integração contínua
- Automatização de testes
- Automatização do deployment

O objetivo principal de uma pipeline de entrega de software é a automatização sem etapas manuais ou alterações necessárias em ou entre qualquer etapa da pipeline, ajudando a prevenir o erro humano que pode ocorrer e ocorre ao realizar tarefas chatas e repetitivas manualmente

e, em última análise, afeta a capacidade de atender às entregas devido a implantações mal feitas (Merron, D., 2020).

2.3.9 Artefactos

Os artefactos são um dos muitos tipos de subprodutos tangíveis produzidos durante o desenvolvimento de software.

Artefactos podem ser bases de dados, modelos de dados, código compilado, documentos ou scripts. Os artefactos ajudam na manutenção e atualização do software, já que os developers podem usá-los como material de referência para ajudar a resolver problemas. São normalmente documentados e armazenados num repositório para que possam ser utilizados pelos developers (Gillis, A. S., 2022).

2.3.10 Stages

Stages são os diferentes passos que constituem uma pipeline: "*Build App*", "*Run Tests*" e "*Deploy to Production*" são exemplos de stages. São limites lógicos na pipeline entre os quais são realizadas pausas para validações ou verificações à aplicação.

Cada pipeline tem pelo menos um stage, mesmo que este não seja definido explicitamente.

Stages em pipelines de Azure DevOps podem ser despoltados de 3 formas:

- Após a release de um artefacto
- Depois de outro(s) stage(s)
- Manualmente

2.3.11 Gates

Os gates permitem a coleta automática de sinais de integridade de serviços externos e, com base nesses dados, controlar o fluxo de uma pipeline. Normalmente, os gates são usados em conexão com gestão de incidentes, gestão de problemas, gestão de alterações, monitorização e sistemas de aprovação externa.

Estes gates são tipicamente colocados imediatamente antes de um stage (pré-requisitos de um deploy) ou imediatamente depois de um stage (pós-requisitos de um deploy).

2.3.12 Agentes

De modo a compilar ou fazer deploy da aplicação usando *Azure Pipelines*, é necessário pelo menos um agente. Um agente é uma infraestrutura de computação que executa uma tarefa da pipeline.

Em *Azure Pipelines* existem 2 tipos de agentes:

- **Microsoft-hosted Agents:** O agente está alojado numa máquina virtual da Microsoft criada especificamente com o propósito de executar uma tarefa da pipeline e é descartada assim que essa tarefa seja concluída.
- **Self-hosted Agents:** O agente está alojado num computador local, permite maior controlo sobre instalações necessárias para a execução de algumas tarefas. Também permite manter caches e configurações que podem aumentar a rapidez das execuções.

Por defeito, o *Azure Pipelines* fornece um *Agent Pool* predefinido com *Microsoft-hosted Agents*.

2.4 Abordagens existentes

A aplicação das metodologias de CI/CD em BC não é um conceito novo, sendo que já existem abordagens estudadas e implementadas em casos reais. No entanto, estas abordagens estão geralmente direcionadas ao desenvolvimento de produto, isto é, à criação de add-ons para vender a clientes.

A intenção da myPartner é estudar como CI/CD pode ser aplicado nos seus projetos em que o produto final deverá ser o ambiente produtivo do cliente ao qual está associado o projeto em questão, em vez de um add-on.

2.4.1 Abordagens atuais da myPartner

Atualmente, a myPartner tem processos definidos que implementa nos seus novos projetos de ERP. Estes processos dividem-se em diferentes fases ao longo de todo o ciclo de vida do projeto, sendo elas:

- Iniciação/Planeamento;
- Análise e Sistematização de Processos;
- Configuração e Desenvolvimento;
- Protótipo;
- Formação;
- Go Live (Arranque);
- Estabilização.

Para o âmbito desta dissertação, o foco será na fase de desenvolvimento que dura até ao fim da estabilização pós-arranque, em que a equipa de desenvolvimento geralmente implementa funcionalidades novas que se revelam necessárias e dão suporte técnico a bugs que não foram detetados anteriormente.

Tipicamente, um projeto ERP da myPartner num cliente tem pelo menos 3 bases de dados:

- **DEV (Development)**- É aqui que todas as alterações são desenvolvidas, sendo tipicamente utilizada apenas pelos developers.
- **TEST** - Depois de desenvolvidas, as alterações são migradas para a BD de TEST em que um consultor funcional realiza testes mais aprofundados à funcionalidade.
- **PROD (Production)**- Esta é a BD utilizada em ambiente de produção. Nenhuma alteração deve ser feita diretamente aqui.

Pode ainda existir uma quarta BD de Protótipo ou *PreProduction* entre TEST e PROD, em que o cliente pode ver e testar a funcionalidade antes desta passar para ambiente produtivo.

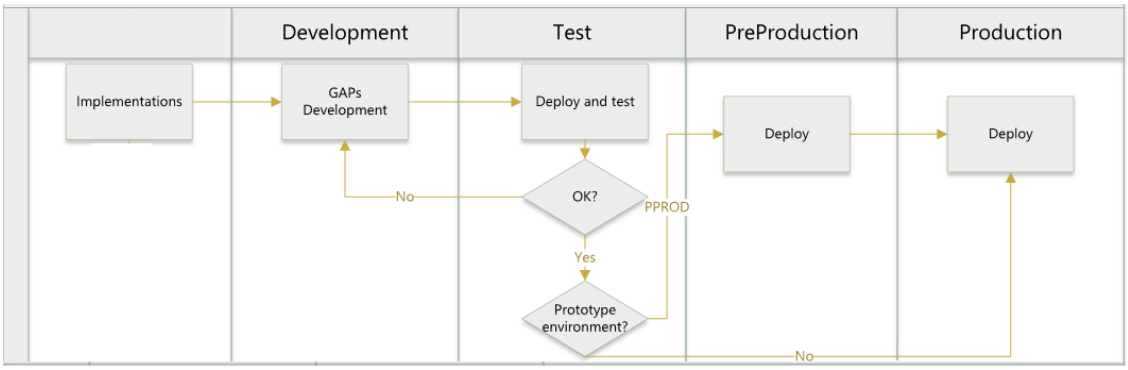


Figura 8: Processo da myPartner para desenvolvimento de nova funcionalidade num projeto BC.

A Figura 8 mostra como os processos atuais da myPartner fluem entre estas 3 ou 4 bases de dados que existem em todos os projetos. Atualmente, todo o processo de migração entre bases de dados é feito manualmente.

2.4.2 Modelos de Branching em projetos não BC

Nos dias de hoje, uma estratégia de backup é essencial para qualquer projeto iniciado por uma organização ou indivíduo. Como os dados são efémeros e podem ser perdidos facilmente – por meio de uma alteração errónea de código ou uma falha catastrófica de disco, digamos – é aconselhável manter um arquivo vivo de todo o trabalho. Para projetos de texto e código, a estratégia de backup normalmente inclui controlo de versões ou rastreamento e gestão de alterações ao código. Cada developer pode fazer várias alterações por dia, e o corpus cada vez maior serve simultaneamente como repositório, meio de comunicação e ferramenta de gestão da equipa e produto. Dado o seu papel fundamental, o controlo de versões é mais eficaz quando adaptado aos hábitos de trabalho e objetivos da equipa de projeto. Uma ferramenta que gere e rastreia diferentes versões de software ou outro conteúdo é chamada genericamente de Version Control System (VCS). Cada sistema aborda essencialmente a mesma questão: desenvolver e manter um repositório de conteúdo, fornecer acesso a edições históricas de cada dado e registar todas as alterações em log (Loeliger et al., 2012).

Embora o conceito de branching não seja exclusivo ao Git, esta secção focar-se-á neste software devido à sua hegemonia entre VCSs.

2.4.2.1 Git Flow

O workflow do Git Flow define um modelo de branching rigoroso projetado em torno das releases do projeto. Isso fornece uma estrutura robusta para gerir projetos maiores.

O Git Flow é ideal para projetos que têm um ciclo de lançamento periódico e para a prática de Continuous Delivery. Atribui funções muito específicas a diferentes branches e define como e quando devem interagir. Este workflow define branches individuais para preparar, manter e registar releases. (Knóldus, 2021)

2.4.2.1.1 Branches permanentes

Em vez de um único branch master, o Git Flow baseia-se em dois branches principais para registar o histórico do projeto:

- **Master:** contém o código de produção e armazena o histórico de release oficial.
- **Develop:** contém o código de pré-produção e serve como um branch de integração para as funcionalidades.

O workflow entre estes 2 branches é demonstrado na Figura 9.

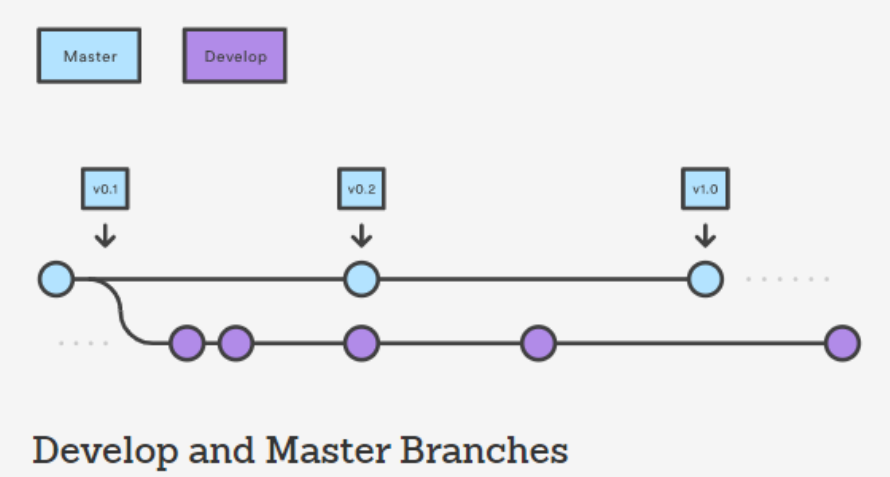


Figura 9: Branches Master e Develop - Git Flow (Babbar, 2021)

2.4.2.1.2 *Feature* branches

Cada nova funcionalidade deve residir no seu branch específico, e pode ser enviada para o repositório central para backup/colaboração. Os *Feature branches* usam a versão mais recente do *Develop branch* como o seu branch pai. Quando uma funcionalidade é concluída, o código é mesclado novamente ao *Develop*. Os F*eature branches* nunca devem interagir diretamente com o *master branch*.

O workflow do modelo de *Feature branches* é demonstrado na Figura 10.

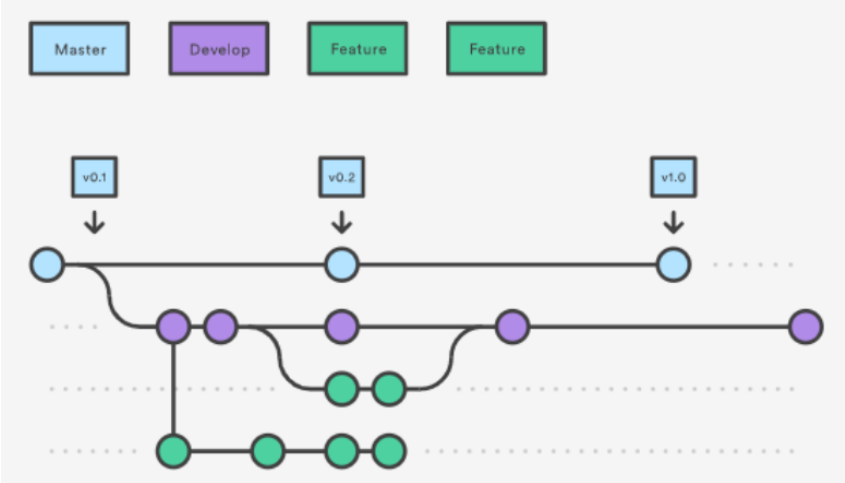


Figura 10: Modelo de *Feature branches* – Git Flow (Babbar, 2021)

2.4.2.1.3 Branches de release

Assim que o *Develop* tenha novas funcionalidades suficientes para justificar uma nova release (ou uma data de predeterminada aproxima-se), é biforcado um novo branch a partir do *Develop* ao qual se chama *Release*. A criação desse branch inicia o próximo ciclo de release, portanto, nenhuma nova funcionalidade pode ser adicionada após esse ponto — apenas correções de bugs, geração de documentação e outras tarefas orientadas à release devem ser incluídas. O branch *Release* ramifica-se do *Develop* e deve ser mesclado ao *master* e ao *Develop*.

O workflow do modelo de branching com releases é demonstrado na Figura 11.

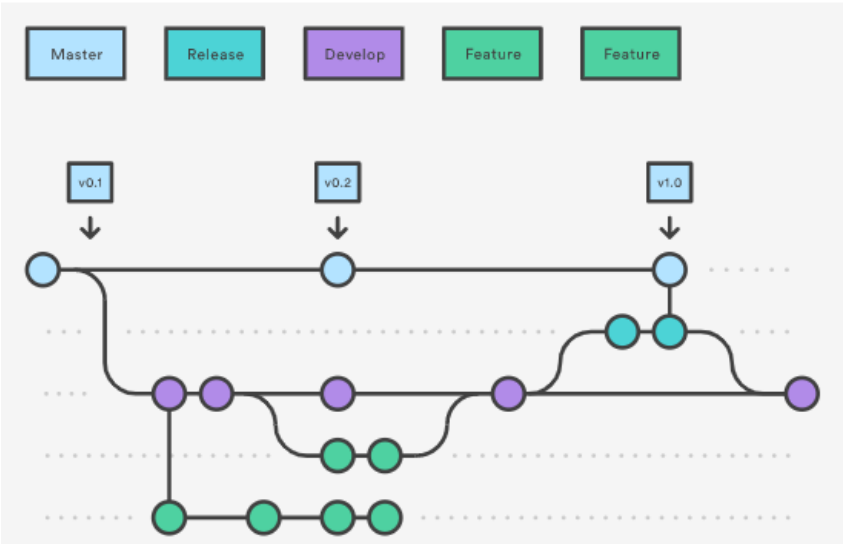


Figura 11: Modelo de branching com releases – Git Flow (Babbar, 2021)

2.4.2.1.4 Branches de hotfix

Os branches de manutenção ou "hotfix" são usados para corrigir rapidamente as versões de produção. Estes branches são necessários para agir imediatamente a erros ou bugs existentes no *master*. Os branches de hotfix são semelhantes aos de release e de feature, com a diferença de que são baseadas diretamente no *master* em vez do *Develop*. Este é o único *branch* que deve ser bifurcado diretamente do *master*. Assim que a correção estiver concluída, ela deve ser mesclada ao *master* e ao *develop* (ou ao branch da release atual).

O workflow do modelo de branching direcionado a hotfixes é demonstrado na Figura 12.

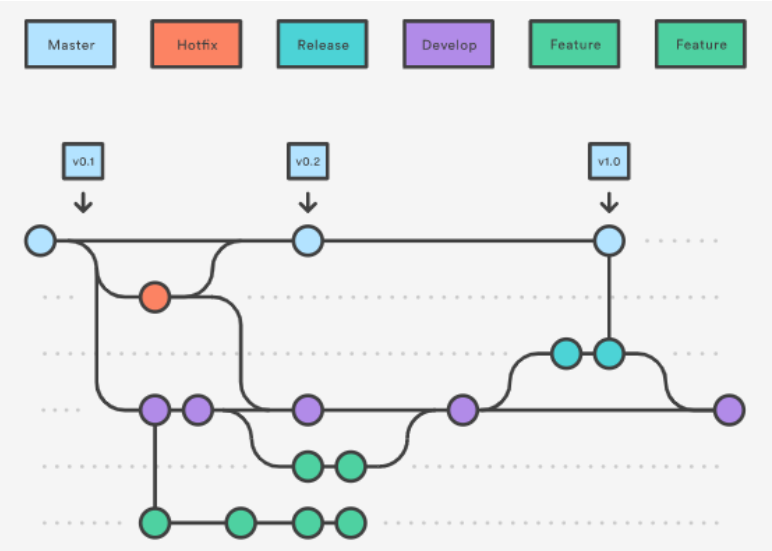


Figura 12: Modelo de branching para Hotfixes – Git Flow (Babbar, 2021)

2.4.2.2 GitHub Flow

Ao contrário do Git Flow, este modelo não possui branches de release. Existe o branch principal *master*, a partir do qual os developers criam branches de funcionalidades onde trabalham isoladamente e, posteriormente, são mesclados de volta ao *master*.

A principal ideia por trás deste modelo é manter o código mestre num estado de implantação constante e, portanto, pode suportar processos de integração e entrega contínuas. (Haddad, 2022)

A Figura 13 descreve o workflow do modelo de branching GitHub Flow.

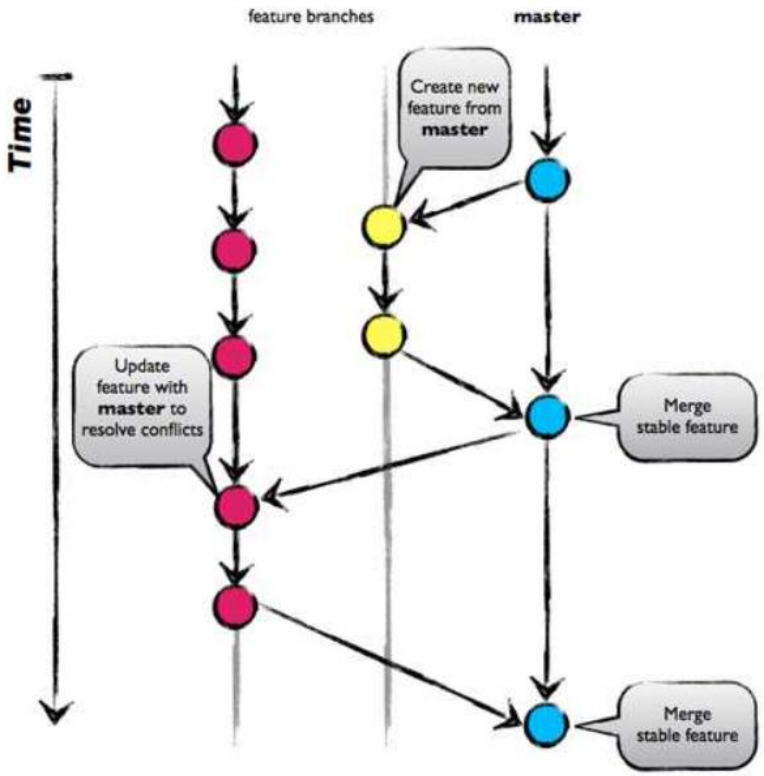


Figura 13: Workflow do modelo de branching Github flow (Haddad, 2022)

O Github Flow é uma estratégia de branching rápida e simplificada com ciclos de produção curtos e releases frequentes.

Este modelo também permite ciclos de feedback rápidos para que as equipas de desenvolvimento possam identificar e resolver problemas rapidamente.

Como, ao contrário do Git Flow, não há um branch de desenvolvimento, os testes e desenvolvimento são feitos no mesmo branch, o que permite uma implantação rápida e contínua.

Este modelo é particularmente adequado para equipas pequenas e é ideal quando é necessário manter uma única versão de produção. Portanto, esta estratégia não é adequada para lidar com várias versões do código.

Além disso, a falta de branches de desenvolvimento torna esta estratégia mais suscetível a bugs e, portanto, pode levar a um código de produção instável se as funcionalidades não forem testadas adequadamente nos *feature branches* antes da fusão com o *master*.

Uma desvantagem adicional é que, como este modelo é mais adequado para equipas pequenas, à medida que as equipas crescem, conflitos de merge podem ocorrer, pois vários developers poderão tentar fazer merge no mesmo branch e há falta de transparência, ou seja, os developers não conseguem ver em quê que os outros developers estão a trabalhar. (Haddad, 2022)

2.4.2.3 GitLab Flow

Um dos problemas do GitHub Flow é que pressupõe o deploy em ambiente de produção sempre que é feito um merge de uma nova funcionalidade para o *master branch*. Embora isso seja possível nalguns casos, como em aplicações de SaaS, há alguns casos em que isso não é possível. Por exemplo, alterações que causem o sistema a ficar indisponível por um curto intervalo de tempo não podem ser efetuadas em horário de produção sem aviso prévio.

2.4.2.3.1 *Production branches*

Nesses casos, pode-se criar um novo branch *Production* que reflita o código implantado em produção. Desta forma, quando se pretender lançar uma nova versão para ambiente de produção, basta fazer merge do *master branch* para o *Production*. A Figura 14 ilustra este processo.

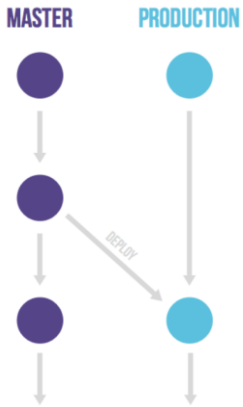


Figura 14: Production branches no modelo GitLab Flow (GitLab, 2022)

2.4.2.3.2 Branches de ambiente

Uma alternativa a ter apenas o ambiente de produção representado por um branch, é ter um ambiente que seja atualizado automaticamente para o branch principal. Supondo que existe um ambiente de teste, um ambiente de pré-produção e um ambiente de produção, nesse caso, pode-se implantar o branch *master* diretamente para o ambiente de testes. (GitLab, 2022)

Para implantar em pré-produção, existirá um branch *Pre-Production* para onde deverá ser feito um pedido de merge sempre que se pretenda instalar a funcionalidade neste ambiente.

Para instalar em produção, faz-se um novo merge do branch *Pre-Production* para o *Production*.

Esse fluxo de trabalho, em que as modificações ao código fluem apenas a jusante, garante que tudo seja testado em todos os ambientes. Caso seja necessário realizar um hotfix a uma alteração específica, é comum desenvolvê-lo num *Feature branch* e depois fazer merge no *master*. A Figura 15 ilustra o workflow do modelo GitLab Flow aplicado no contexto de branches de ambiente.

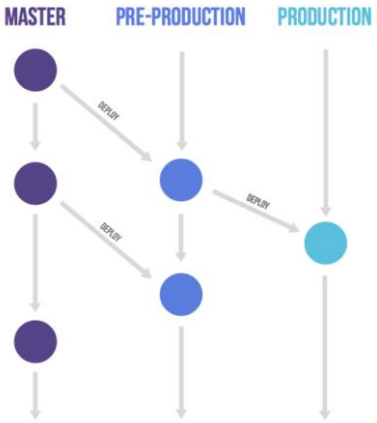


Figura 15: Branches de ambiente no modelo GitLab Flow (GitLab, 2022)

2.4.2.3.3 Branches de Release

Só é necessário trabalhar com branches de release se for preciso lançar software para o mundo exterior (clientes). No modelo GitLab Flow, cada branch contém uma versão secundária, como *2-3-stable* ou *2-4-stable*. Estes branches estáveis devem derivar do *master* como ponto de partida e ser ramificados o mais tarde possível. Ao fazer isso, minimiza-se o tempo durante o qual se precisa aplicar correções de bugs a vários branches. Cada vez que seja incluída uma correção de bug num branch de release, deve ser aumentada a versão do patch definindo uma nova tag. Neste fluxo, não é comum ter um branch de produção (GitLab, 2022). A Figura 16 ilustra o fluxo de alterações num modelo GitLab Flow com branches de release.

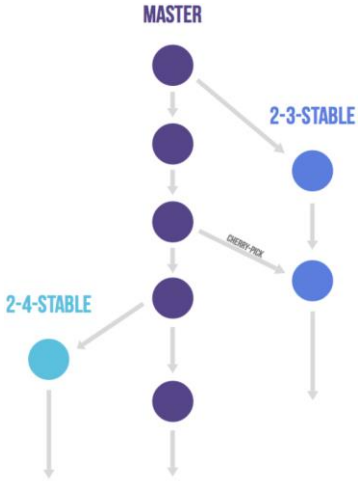


Figura 16: Branches de release no modelo GitLab Flow (GitLab, 2022)

2.4.2.4 Trunk-based Development

Trunk-based Development (TBD) é uma estratégia de branching que, na verdade, não requer branches, mas, em vez disso, os developers integram as suas alterações num tronco compartilhado pelo menos uma vez por dia. Este tronco compartilhado deve estar pronto para release a qualquer momento.

A ideia principal por trás desta estratégia é que os developers façam alterações menores com mais frequência e, portanto, o objetivo é limitar branches de longa duração e evitar conflitos de merge, pois todos os developers trabalham essencialmente no mesmo branch. (Haddad, 2022)

Consequentemente, o TBD é um facilitador chave da integração contínua (CI) e da entrega contínua (CD), uma vez que as alterações são feitas com mais frequência no tronco, muitas vezes várias vezes ao dia (CI), permitindo que os recursos sejam lançados muito mais rapidamente (CD).

Como num modelo de TBD não existe o conceito de branch de hotfix, as funcionalidades são tipicamente lançadas para produção num estado desativado, e são posteriormente ativadas em produção, esta metodologia é tipicamente referida como *Feature Toggles* ou *Feature Flags*.

A Figura 17 ilustra um modelo de TBD.

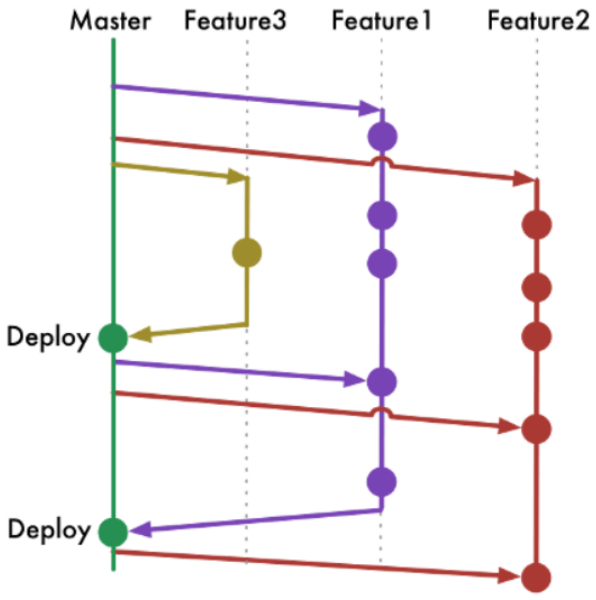


Figura 17: Modelo de Trunk-Based Development (Sharma, 2020)

A Tabela 2 apresenta um estudo comparativo entre os diferentes modelos abordados e como estes respondem a algumas questões relevantes ao seu funcionamento.

Tabela 2: Estudo comparativo entre os diferentes modelos de branching

Característica	**Git Flow**	**GitHub Flow**	**GitLab Flow**	**TBD**
Complexidade	Significante	Baixa	Média	Baixa
Branch de deploy	Release	Feature	Opcional	master
Trigger para deployment	Ciclo da release	Aprovação de Pull Request	Merge para Production	merge para master
Branch mais estável	master	master	Production	master
Hotfixes via	Hotfix Branch	Feature Branch	Hotfix Branch	master
Ritmo de entrega	Por Release	Contínua	Contínua	Contínua
Esforço de merge	Significante	Mínimo	Mínimo a Médio consoante o nº de branches de ambiente	Mínimo
Período de feedback	Longo	Curto	Médio	Curto
Problemas em produção	Reversão (Rollback)	Correção rápida via Feature Branch	Rollback	Desativação da funcionalidade (Feature Toggle)
Tipos de projeto	Projetos empresariais	Projetos open-source e Startups	Projetos empresariais	Projetos empresariais

2.4.3 Abordagens de CI/CD existentes em projetos BC

Neste subcapítulo serão descritas as metodologias existentes à aplicação das práticas de CI/CD em projetos de BC.

2.4.3.1 Abordagem de Michael Megel

Michael Megel é um arquiteto de soluções empresariais que trabalha atualmente para a COSMO CONSULT, uma empresa que desenvolve soluções para BC.

Em 2019, escreveu uma série de blogs em que descreve uma proposta de implementação das metodologias de CI/CD em BC através do Azure DevOps, assim como instruções técnicas de como utilizar a ferramenta e implementar estas soluções. Neste subcapítulo, toda a informação apresentada refere-se à metodologia proposta nestes artigos.

Artefactos e fases de Deployment

No contexto do Azure Devops, um artefacto é um pacote. Este pode conter a aplicação (incluindo também o ambiente de dev e test) de Business Central e, por vezes, dados sob a forma de pacotes de RapidStart.

Um artefacto nunca deve ser alterado. Em vez disso, deve ser gerada uma nova versão do artefacto despoletando a pipeline de CI.

Na sua proposta, Michael descreve 4 fases de deployment, isto é, momentos em que uma migração entre ambientes é realizada (Megel, 2019):

- **D-Stage:** é a fase mais próxima do desenvolvimento e contém os artefactos gerados mais recentes, tipicamente do ambiente de DEV. A BD é tipicamente um conjunto de dados de teste baseado na empresa CRONUS (uma empresa fictícia que vem com a aplicação base do BC). É nesta fase que será feita a execução periódica dos testes, normalmente à noite.
- **Q-Stage:** é usado para o controlo de qualidade. Esta fase é mais estável que o *D-Stage* e a BD pode ser usada para configuração manual dos dados de teste. Esta fase é tipicamente usada pelos consultores para preparar as configurações da BD, pacotes RapidStart e executar testes de integração adicionais. Quando esta etapa é bem-sucedida, o artefacto é enviado para o ambiente do cliente.
- **UAT-Stage** (User Acceptance Test): é usado para que o cliente e os seus utilizadores-chave tenham a possibilidade de testar e aprovar as alterações antes de estas passarem para ambiente produtivo. Esta fase é baseada numa cópia da BD de produção.
- **P-Stage:** é o objetivo final de cada deployment. Se o artefacto ultrapassar todas as outras fases, este será implantado no ambiente de produção do cliente. Este é o ponto sem retorno. Se as alterações efetuadas não foram bem testadas em etapas anteriores causando a corrupção de dados, é importante ter uma boa estratégia de backup.

Como os artefactos nunca devem ser alterados, os seus deploys são feitos apenas numa direção – para a frente - até que algum *stage* da pipeline falhe ou até que este chegue ao destino final. Surge então a questão de como controlar o fluxo da pipeline?

Em cada stage, o fluxo é controlado respondendo às questões:

- **Quando** deve ser iniciado um *stage*? – *triggers*.
- **Quem** permite que o artefato entre no *stage* e **o quê** que deve ser passado ou informado? – Pré-condições e *Gates.*
- **Quem** decide que o *stage* está concluído e **o quê** que deve ser passado ou informado? – Pós-condições e *Gates.*

As aprovações pré e pós-deploy ajudam a controlar o fluxo da pipeline. Megel sugere incluir o cliente neste processo. Quem (por exemplo, os utilizadores-chave) deve aprovar a funcionalidade/hotfix e permitir o deploy do artefacto em produção?

Fluxos de artefactos

Artefatos que vêm diretos do desenvolvimento - [dev] Branch - são tipicamente "instáveis". É útil tratá-los separadamente num fluxo próprio ou pipeline de CD.

Quando um artefato é gerado pelo CI, este será implantado imediatamente nos stages de desenvolvimento e de testes automatizados. Consultores e developers podem conferir o artefacto aqui. A realização dos testes automáticos é normalmente agendada para ser executada todas as noites. As aprovações não são desnecessárias.

A Figura 18 exemplifica estes 2 stages em fluxos isolados do resto da pipeline.

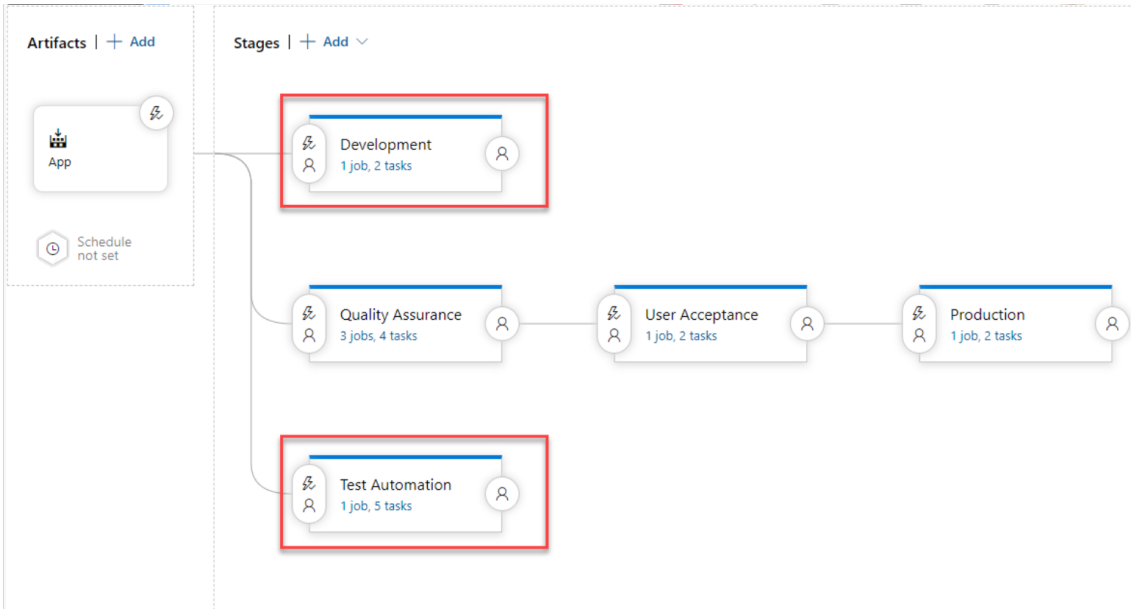


Figura 18: Stages Development e Test Automation (Megel, 2019b)

O artefacto usado para o stage que faz o deploy para ambiente de produção depende do processo de desenvolvimento e do modelo de branching utilizado. É a resposta à pergunta: "O que pretende lançar para o cliente?"

A Figura 19 mostra um exemplo do fluxo de deploy para ambiente de produção.

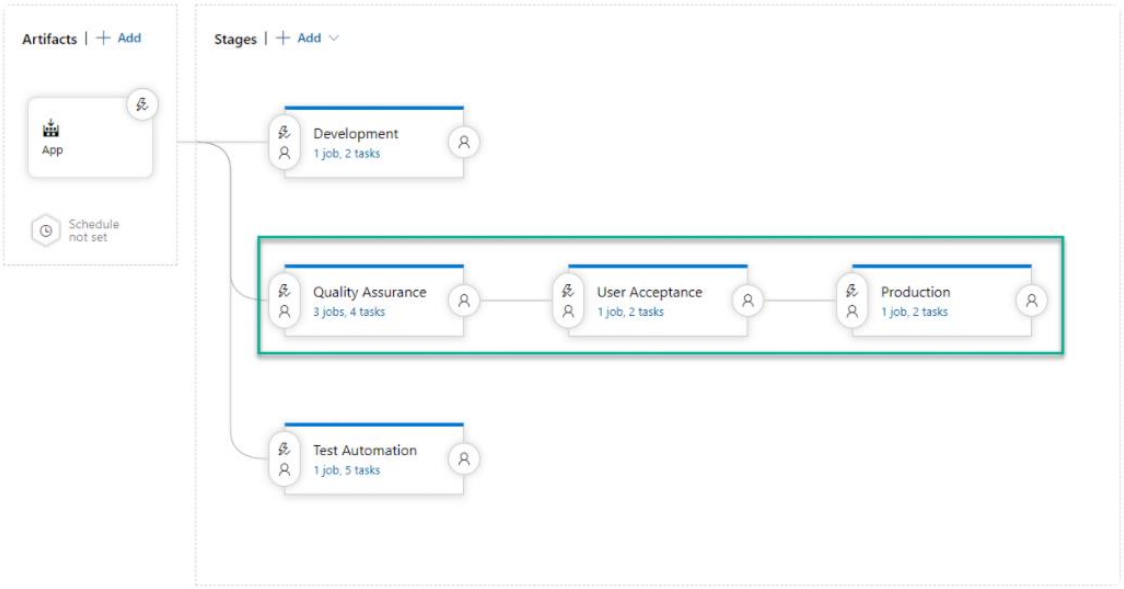


Figura 19: Fluxo de *deploy* para produção (Megel, 2019b)

Neste exemplo são utilizados 3 stages:

- **Quality Assurance** – Controlo de Qualidade - O artefacto de lançamento pode ser agendado (manhã, almoço, etc.) de modo a evitar deploys e resultados de testes imprevisíveis. Aqui são também realizadas aprovações por parte dos consultores.
- **User Acceptance Test (UAT)** – Quando o controlo de qualidade é aprovado, o artefacto é sujeito aos dados de produção num ambiente sandbox ou ambiente de testes. As pré-aprovações permitem um lançamento manual para este stage e ambiente. As pós-aprovações por utilizadores-chave autorizam o artefacto para o próximo stage ou interrompem o fluxo - "aprovado" ou "reprovado".
- **Production** – Produção – O fluxo chega a este stage quando todos os stages anteriores são ultrapassados. O artefacto é testado, aprovado e está pronto para deploy no ambiente de produção do cliente. As pré-aprovações autorizam o deploy e o agendamento define a janela de deployment.

Deploy em ambientes com o Azure DevOps

Quando falamos em ambientes de Business Central, existem tipicamente 2 tipos de ambientes: SaaS e OnPrem. Como foi delineado no capítulo 1.3, este projeto irá apenas abordar ambientes SaaS.

Aqui, os artefactos libertados da pipeline devem ser implantados no ambiente SaaS, alojado no Azure DevOps.

O BC como SaaS oferece duas formas de carregar e instalar extensões. Através de uma API, que permite automatizar o processo, ou manualmente na própria aplicação como exemplificado na Figura 20.

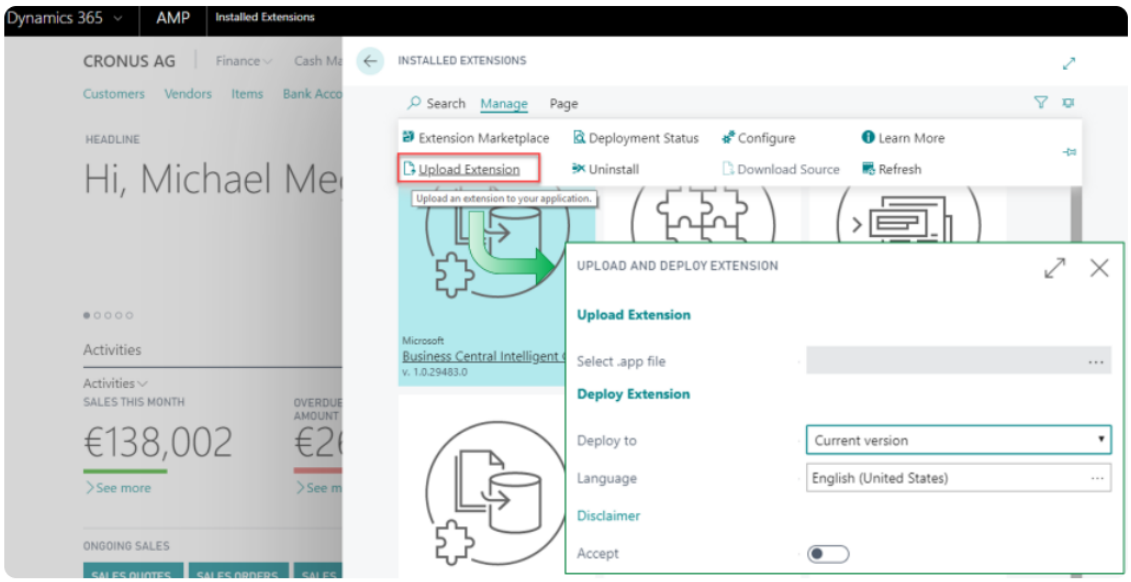


Figura 20: Instalação manual de uma extenção em BC (Megel, 2019c)

O carregamento manual é precisamente o que não é pretendido. O objetivo principal de uma metodologia CI/CD é automatizar processos.

Os agentes do Azure DevOps podem ser executados como um serviço local no ambiente destino. Quando o agente é iniciado, este conecta-se ao Azure DevOps e aguarda comandos.

Dependendo dos parâmetros de instalação, o agente é instalado numa *Agent Pool* ou numa *Deployment Pool*.

Agent Pools são normalmente utilizadas em pipelines de CI. As tarefas de build ou deploy devem ser adicionadas às tarefas do agente e serão executadas em qualquer agente do *pool* selecionado.

Após criar uma *Agent Pool*, é possível fazer download do software de agente e instalá-lo na máquina pretendida.

Um *Deployment Pool/Group* é um conjunto lógico de máquinas que possuem agentes instalados que irão realizar o processo de deploy. Os *Deployment Groups* representam os ambientes físicos, por exemplo, ambiente "Dev", "Test" ou "Production". Na prática, um *Deployment Group* não é mais do que um conjunto de agentes, à semelhança de um *Agent Pool,* com a diferença de que estes têm um propósito mais específico.

CI/CD Pipeline para Dynamics 365 Business Central

O Azure DevOps oferece várias tarefas/pipelines de raiz para executar diversas pipelines no contexto de build ou deploy. No entanto, pipelines ou tarefas pré-configuradas para o Microsoft Dynamics 365 Business Central não fazem parte desta lista.

Pipelines são criadas e definidas através de scripts de Powershell que podem ser executados diretamente ou a partir de ficheiros YAML, e estas podem ser criadas e customizadas para corresponder às necessidades de um determinado projeto. (Megel, 2019d)

Alternativamente, Megel recomenda a utilização da extensão para Azure DevOps, ALOps - que contém várias pipelines de CI/CD desenhadas especificamente para projetos de BC.

2.4.3.2 ALOps

Uma das dificuldades em montar um ambiente de CI/CD para projetos de BC é que o facto de ser um nicho muito específico na área de desenvolvimento de software, reflete-se na existência de menos informação técnica disponível sobre como fazer este tipo de implementações. Geralmente, programadores de AL não têm o vasto conhecimento de PowerShell necessário para uma implementação desta natureza. Da mesma forma, especialistas em PowerShell carecem dos conhecimentos técnicos acerca da arquitetura e processos inerentes ao Business Central também necessários.

Para colmatar esta dificuldade, a empresa Hodor NV desenvolveu uma extensão para Azure DevOps contendo diversos scripts de PowerShell necessários para gerir o fluxo de trabalho de um processo CI/CD no âmbito de projetos de Business Central. Permitindo criar e gerir pipelines de build e release sem ter de programar diretamente em PowerShell (Hodor NV, 2019).

Alguns dos benefícios que o ALOps oferece são:

- Flexibilidade total da pipeline: As pipelines são criadas de acordo com os parâmetros definidos e não estão "pré-construídas" de forma estática.
- Nenhum conhecimento de PowerShell é necessário para as pipelines de build e release mais complexas.
- A gestão e manutenção dos scripts de PowerShell são feitos pela Hodor.
- Suporte a implantações em ambientes OnPrem e SaaS sem a necessidade de agentes de compilação nos clientes.
- Compatibilidade com as diversas versões do Business Central existentes.
- Todo o output relativo à execução das pipelines é disponibilizado para análise.
- Várias tarefas específicas do Business Central (Hodor NV, 2019).

2.4.3.3 Al-Go for GitHub

AL-Go for GitHub é um conjunto de modelos e ações do GitHub, que podem ser usados para configurar e manter processos de DevOps profissionais para projetos de Business Central. Estas funcionalidades são instaladas adicionando um conjunto de ficheiros contendo scripts YAML e PowerShell ao repositório do projeto.

O objetivo é que os developers que criaram repositórios de GitHub com base nos modelos AL-Go, possam gerir esses repositórios mantê-los atualizados com apenas a execução de um fluxo

de trabalho que atualiza os repositórios. Isto inclui as alterações necessárias em scripts e workflows para lidar com novos recursos e funções no Business Central.

A maior vantagem do AL-Go for GitHub é precisamente essa, a atualização do repositório para a versão mais recente do AL-Go for GitHub é feita executando um fluxo de trabalho. Até que se execute esse fluxo, o repositório continua sendo executado na versão atual. (Freddy Kristiansen, 2022)

A adição de novas funcionalidades e atualizações regulares ao AL-Go for GitHub não interrompem pipelines existentes e podem ser aplicadas e removidas como qualquer outro Pull Request, permitindo uma maior estabilidade nos projetos. (Freddy Kristiansen, 2022)

Outra vantagem é que, ao contrário do ALOps, todos os scripts de PowerShell estão no próprio repositório e podem ser alterados e adaptados consoante as necessidades do projeto, no entanto, isto requere vasto conhecimento de PowerShell e de Business Central, pelo que não é recomendável na maior parte dos casos.

Por esta razão, o AL-Go for GitHub acaba por ser até menos flexível que o ALOps no sentido em que o ALOps permite bastante customização das pipelines sem necessidade de conhecimentos PowerShell, enquanto o AL-Go requere estes conhecimentos para poder explorar a flexibilidade de ter acesso ao código-fonte.

3 Análise de Valor

Este capítulo visa apresentar uma análise ao valor que a solução pode gerar à myPartner. Para isso, será feita uma análise aos cinco elementos do modelo NCD, seguido da proposta de valor e abordando temas como a dívida técnica.

3.1 New Concept Development Model

O modelo NCD divide-se em três áreas distintas:

- O *engine*, que impulsiona os cinco elementos de atividade de *front-end* que a organização controla.
- A *wheel,* referindo ao interior do modelo que define os cinco elementos de atividade:
    - **Identificação da oportunidade** – a organização identifica oportunidades, definindo o mercado ou área tecnológica;
    - **Análise da oportunidade** – avaliação da oportunidade através do estudo de mercado, dos concorrentes e do cliente, de forma a entender o potencial;
    - **Geração da ideia**: conceção, desenvolvimento e amadurecimento de uma ideia concreta, com recurso a processos formais, como *brainstorming,* ou não formais como um pedido de um utilizador;
    - **Seleção da ideia**: avaliação das ideias, e seleção da ideia com maior potencial.
    - **Definição de conceito**: Análise da viabilidade dos processos, dos requisitos técnicos e dos fatores económicos relacionados com o projeto.
- O *rim,* representa os fatores ambientais que influenciam o *engine* e moldam os cinco elementos da atividade. Isso inclui as capacidades organizacionais da

empresa, ameaças da concorrência, tendências mundiais e de clientes, mudanças regulatórias e a profundidade e força de habilitar ciências e tecnologias.

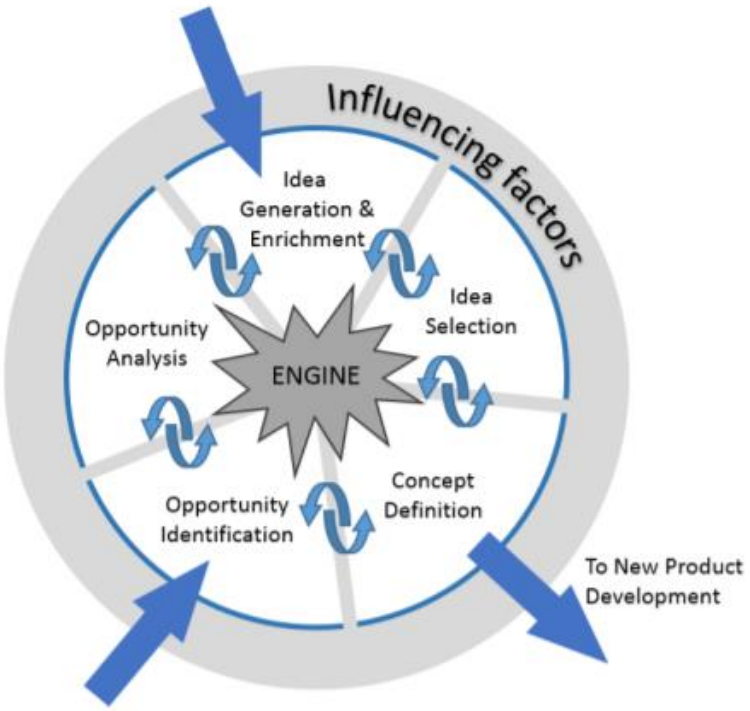


Figura 21: Modelo NCD (Koen et al., 2001)

Na Figura 21 está representado o modelo como descrito por Peter A. Koen. Em contraste com os processos lineares, encenados e controlados, o modelo é circular para indicar que as ideias fluem, circulam e iteram entre os cinco elementos. As setas que apontam para o modelo representam os pontos de partida para os projetos e indicam que os projetos podem começar tanto na identificação de oportunidades quanto na geração e enriquecimento de ideias. Os projetos saem entrando no processo de desenvolvimento de novos produtos ou tecnologia *Stage-Gate*. (Koen et al., 2001)

3.1.1 Identificação da oportunidade

Com o upgrade realizado pela Microsoft que converteu o antigo NAV no novo Business Central, e com todas as mudanças de paradigma associadas a um nível técnico para quem é developer deste ERP, surgem naturalmente desafios associados, especialmente de adaptação às novas ferramentas e metodologias.

No entanto, com mudança, surgem oportunidades, especificamente:

- Controlo sobre o armazenamento/organização do código na forma de extensões;
- Possibilidade de criação de pipelines de CI;

- Suporte para testes automatizados;
- Processos de deploy podem ser automatizados;

O intuito deste projeto é precisamente a exploração de todas estas oportunidades. E o desenvolvimento de uma abordagem sistemática para aplicação dos conceitos de CI/CD permitindo tirar partido destas oportunidades e melhorar os processos atuais da empresa, com impacto direto na produtividade e qualidade dos serviços prestados.

3.1.2 Análise da oportunidade

De modo a sustentar a oportunidade referida, é necessário analisar a mesma para perceber quais são as vantagens e desvantagens da solução. Para realizar um diagnóstico foi realizada uma análise SWOT, apresentada na Figura 22. As letras SWOT referem-se a *Strenghts* (pontos fortes), *Weaknesses* (pontos fracos), *Opportunities* (oportunidades) e *Threats* (ameaças).

Figura 22: Análise SWOT

3.1.3 Geração da ideia

Devido à mudança de paradigma já referida noutros capítulos, a necessidade de adotar novas práticas de desenvolvimento é inevitável. A implementação das práticas de CI/CD nos projetos BC da myPartner é uma solução que vem complementar as mudanças nos processos de desenvolvimento já adotados pela myPartner para projetos nas novas versões.

A plataforma Azure DevOps possibilita dar resposta às questões levantadas em 1.2. Sendo que soluções já existentes podem ser aplicadas no contexto da myPartner com pouca necessidade de adaptações.

3.1.4 Seleção da ideia

Uma vez que o BC é um ERP que faz parte do ecossistema da Microsoft, tanto uma solução em Azure DevOps como em GitHub, ambas plataformas que também integram o ecossistema da Microsoft, são respostas válidas para auxiliar na aplicação das novas metodologias. No entanto, O Azure DevOps já é utilizado pela myPartner em projetos de BC, reduzindo a necessidade de aprendizagem e adoção de uma plataforma nova, o que traria custos adicionais à empresa. Para além disto, a solução em Azure DevOps requere menos necessidade de conhecimentos especializados, nomeadamente em PowerShell para criação de pipelines, como foi abordado em 2.4.3.

Outras especificidades do problema, como a criação de pipelines, ou os modelos de branching, serão alvo de constante criação, desenvolvimento, adaptação e por vezes desistência de certas ideias até se atingir a maturidade da solução final.

3.1.5 Definição do conceito

Decidida a ideia, a ferramenta e a forma adequada de implementar a solução, será possível automatizar alguns dos processos que atualmente consomem tempo e abrem espaço para que erro humano aconteça. Irá também facilitar a gestão do código e a gestão técnica dos projetos, tendo todas as informações e ferramentas necessárias centralizadas numa só plataforma.

3.2 Proposta de Valor

Ao levar a cabo uma mudança tão radical nos processos de desenvolvimento atuais da empresa, é importante ter ideia do valor que deve ser criado com a adoção das novas metodologias e certificar de que este novo valor torna o desenvolvimento dos projetos melhor do que anteriormente e de que os benefícios superam os custos e valem os riscos.

O valor para o cliente corresponde a uma perceção pessoal, da vantagem que um cliente ganha com a oferta de uma organização e pode ocorrer através da redução de esforço e custos, benefícios e a combinação de ambos (Woodall, 2003). O *perceived value* tem por base as perceções do que é recebido e é o resultado da análise da utilidade de um produto (Zeithmal, 1988).

Desta forma, no âmbito deste projeto, é oportuno analisar os benefícios e esforços para o cliente, neste caso a myPartner e/ou outras empresas semelhantes que considerem implementar a solução a desenvolver.

Tabela 3: Benefícios e Esforços/Sacrifícios

Benefícios	**Esforços/Sacrifícios**
• Automatização de processos • Redução do risco de erro humano • Deteção de erros em fases mais iniciais (antes de chegar a ambiente de produção) • Exposição a inovação futura na área de CI/CD para projetos BC pela Microsoft • Maior facilidade na sistematização de processos dos projetos • Redução do esforço dos developers em processos que passam a estar automatizados • Aumento de produtividade • Melhoramento de qualidade dos serviços	• Necessária análise e design de uma abordagem que possa ser aplicada na empresa • Possível dificuldade/resistência numa fase inicial por parte dos developers • Processos definidos devem ser seguidos rigorosamente • Formação dos developers para que apliquem corretamente as novas práticas • Formação técnica necessária para criação e manutenção de pipelines no Azure DevOps

A Tabela 3 mostra a lista de esforços e sacrifícios identificados na análise de valor.

Os benefícios identificados resultam do entendimento da plataforma a ser utilizada, assim como dos processos atuais da myPartner, bem como a sua aplicabilidade ao nível de metodologias de desenvolvimento e competências técnicas e as principais vantagens e desafios em relação ao processo de desenvolvimento atual. Contudo, apesar destes benefícios, existem sacrifícios não monetários associados, uma vez que o entendimento e aplicação deste estudo exige um esforço no conhecimento das plataformas, conceitos e práticas a utilizar.

3.3 Modelo Canvas

Um modelo de negócios *Canvas* oferece uma visão abrangente e de alto nível dos vários detalhes estratégicos necessários para levar um produto ao mercado com sucesso. O caso de uso típico desta ferramenta é delinear os blocos de construção fundamentais de um negócio, mas também pode ser usado de forma eficaz para produtos internos (ProductPlan, 2021). As métricas exatas podem variar, mas abaixo seguem-se alguns dos componentes típicos incluídos:

- **Parceiros-chave** - Que outras empresas ou indivíduos fazem parte da estratégia de desenvolvimento e entrada no mercado?
- **Atividades-chave** - O que deve acontecer internamente para tornar possível a incepção do produto?
- **Recursos-chave** - Que pessoas, materiais e orçamentos são necessários?
- **Propostas de valor** – Que benefícios trará o produto para o cliente?
- **Relações com cliente** — De que forma o cliente interage com o produto?

- **Canais**—Como o produto será vendido ou distribuído?
- **Segmentos de clientes**—Quem vai usar este produto?
- **Estrutura de custos** — Quanto custará desenvolver, fabricar, distribuir e dar suporte ao produto?
- **Fontes de receita**—Como a empresa ganhará dinheiro com este produto?

A Figura 23 apresenta o modelo *Canvas* onde as métricas acima referidas são consideradas e apresentadas.

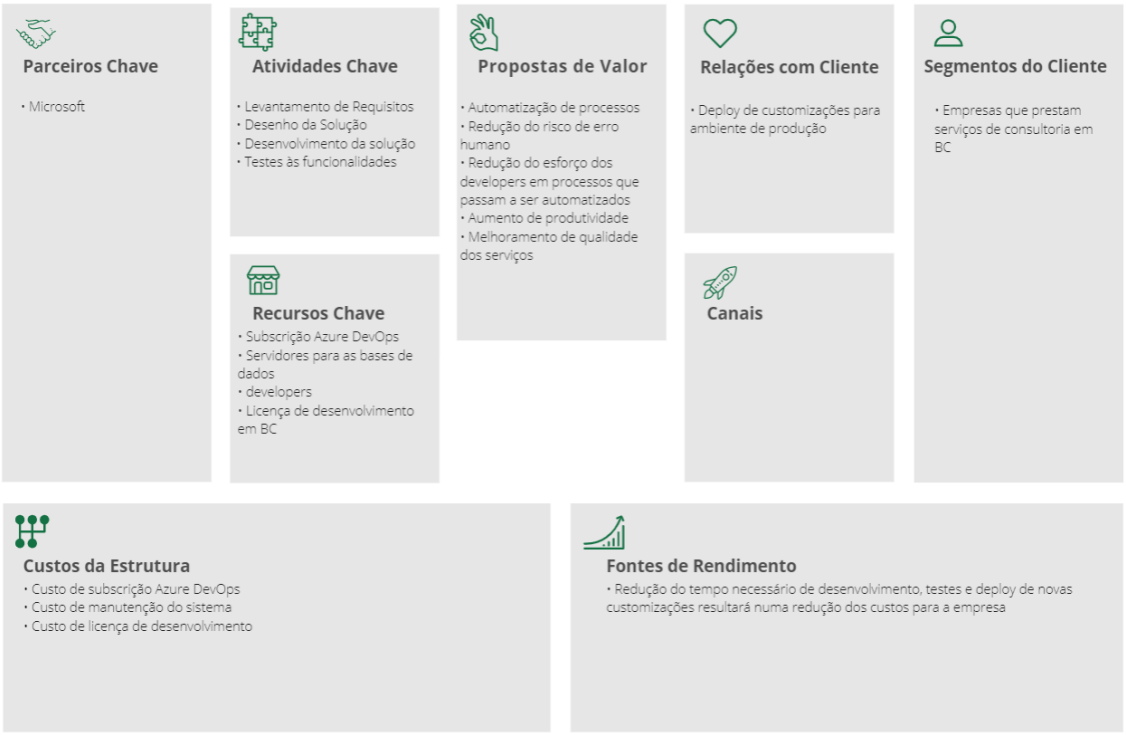


Figura 23: Modelo Canvas

3.4 Dívida Técnica

A Dívida Técnica (DT) ou *Technical Debt* é uma metáfora usada para representar soluções de projeto ou implementação abaixo do ideal que geram um benefício no curto prazo, mas tornam as mudanças mais caras ou mesmo impossíveis no médio e longo prazo (Lenarduzzi et al., 2020).

A DT é geralmente vista pela comunidade de desenvolvimento de software como fazer compromissos técnicos que são convenientes reduzindo tempos de implementação e custos no curto prazo, mas que criam um contexto técnico que aumenta a complexidade e o custo a longo prazo. Embora as raízes conceituais da dívida técnica impliquem um processo de tomada de decisão idealizado e deliberado e uma estratégia de reconstrução conforme necessário, a dívida

técnica geralmente é incorrida involuntariamente e apanha os desenvolvedores de software de surpresa. Por isso, é observada principalmente durante as fases de manutenção e evolução de projetos de software (Avgeriou et al., 2016).

No contexto de gestão de software, as organizações estão interessadas nas consequências económicas da DT e dos riscos de negócios que podem representar. Negligenciar a DT pode aumentar a ineficiência na execução de sistemas de software e criar dificuldades ao ampliá-los para dar suporte às operações de negócios. Os custos indiretos causados por tais problemas são considerados como *juros da dívida técnica*. Seguindo essa perspetiva económica, a noção de dívida técnica é definida como: o custo de reparar problemas de qualidade em sistemas de software para atingir um nível de qualidade ideal – em termos práticos, um nível de qualidade ideal representa o nível mais alto de qualidade alcançável definido num modelo de qualidade de software adotado por uma organização. Esse tipo de reparo envolve mudanças que nem sempre são desejadas pelas organizações, pois os problemas e os seus impactos não são visíveis no curto prazo. O montante da dívida dependerá da diferença entre o nível de qualidade atual e o ideal. A dívida técnica cresce ao longo do tempo se não for abordada desde o início porque a qualidade das mudanças realizadas em sistemas de baixa qualidade técnica tendem a acumular os defeitos no software (Eick et al,. 2002).

A noção correspondente de juros é definida como *o custo extra de manutenção gasto por não atingir o nível de qualidade ideal*. A manutenção engloba atividades como adicionar novas funcionalidades e correção de bugs. A manutenção difere da reparação de qualidade técnica, pois a última envolve mudanças visíveis e os impactos são imediatamente visíveis (por exemplo, correção de bugs ou adição de novos recursos). Esforço extra gasto na adaptação de novas funcionalidades ou correção de bugs são exemplos de juros da dívida técnica. Assim, os juros da dívida técnica não são os mesmos que os custos de manutenção. Sistemas sem problemas técnicos gastam sempre algum esforço em manutenção (Nugroho et al, 2011).

A Figura 24 ilustra estes conceitos e como estes afetam os custos de um projeto ao longo do tempo.

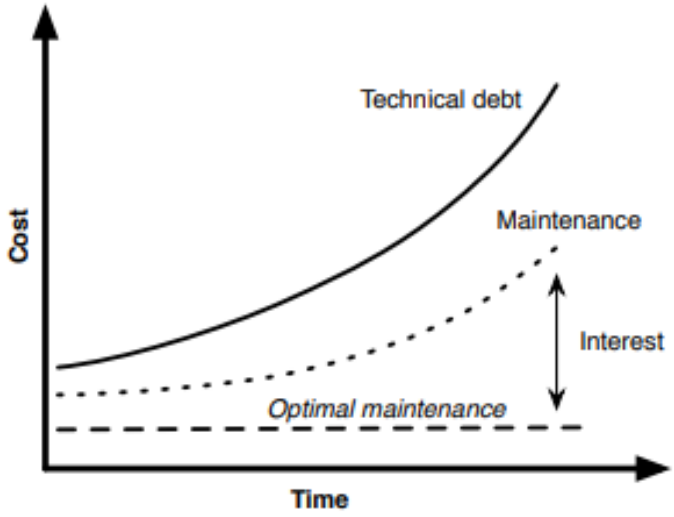


Figura 24: Dívida técnica e o crescimento dos juros ao longo do tempo (Nugroho et al, 2011)

Na abordagem ao desenvolvimento de software sob as metodologias de CI/CD a adotar para a myPartner sobre a qual encaixa o âmbito desta dissertação, é importante ter a DT em consideração para que os benefícios no curto prazo não sejam anulados pelos custos a longo prazo. Para isso, uma solução ideal deve requerer o mínimo de manutenção possível de modo a manter os juros da dívida técnica baixos, como demostra a Figura 24.

Neste contexto e no contexto do projeto em mãos, a manutenção do software relativo às metodologias de CI/CD a aplicar diz respeito essencialmente à manutenção das pipelines que serão criadas. Como foi abordado em 2.4.3, a utilização da extensão ALOps reduz a manutenção necessária às pipelines, diminuindo os juros da DT e reforçando a aposta na utilização do Azure DevOps.

3.5 Método AHP

O *Analytic Hierarchy Process* (AHP) é um método para organizar e analisar decisões complexas, usando matemática e psicologia. Foi desenvolvido por Thomas L. Saaty na década de 1970 e foi aperfeiçoado desde então. (Lin et al, 1999)

O Método AHP será aplicado para analisar qual das políticas de branching estudadas em 2.4.2 melhor se adequa aos processos da myPartner e aos projetos de implementação do BC em clientes.

Foram estudadas 4 soluções:

- Git Flow;
- GitHub Flow;
- GitLab Flow;

- Trunk-based Development (TBD).

Definidas as diferentes soluções a avaliar, é necessário, inicialmente, construir a árvore hierárquica de decisão que apresenta o objetivo da decisão, os critérios de decisão e as alternativas identificadas. Os critérios de decisão a considerar estão apresentados na Figura 25 e são os seguintes:

- **Tempo**: tempo estimado gasto na implementação e manutenção da solução;
- **Adequação**: relação e capacidade de resposta da solução aos requerimentos levantados;
- **Complexidade**: esforço e dificuldade previstos para a implementação da solução.

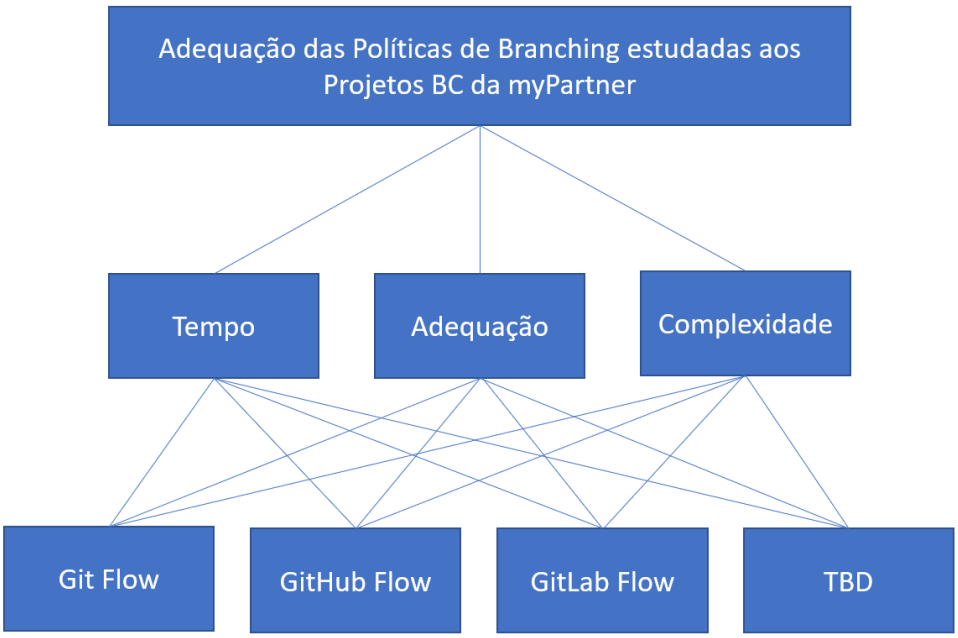


Figura 25: Árvore hierárquica de decisão

Para a atribuição de níveis de importância nas comparações seguir-se-á a Escala Fundamental - Níveis de importância e comparação de (Saaty, 1991).

Inicialmente, realiza-se a comparação entre os três critérios definidos, resultando na matriz de comparação apresentada na Tabela 4.

Tabela 4: Matriz de comparação de critérios

	Tempo	Adequação	Complexidade
Tempo	**1**	**1/2**	**2**
Adequação	**2**	**1**	**3**
Complexidade	**1/2**	**1/3**	**1**
Total	**7/2**	**11/6**	**6**

De seguida, normalizam-se os valores da matriz de comparação, dividindo-se o valor da matriz pelo total da sua coluna. Por último, calcula-se a média aritmética dos valores de cada linha da matriz, de forma a obter-se o denominado vetor de prioridades, tal como se verifica na Tabela 5.

Tabela 5: Matriz normalizada dos critérios

	Tempo	Adequação	Complexidade	Peso do Critério
Tempo	**2/7**	**3/11**	**2/6**	**29,3%**
Adequação	**4/7**	**6/11**	**3/6**	**54%**
Complexidade	**1/7**	**2/11**	**1/6**	**16,3%**

Em seguida, de forma a avaliar-se a consistência das prioridades relativas calcula-se a Razão de Consistência (RC), através da fórmula IC= (λmax-n)/(n-1), sendo n=3. O valor de λmax é obtido através dos seguintes cálculos: multiplicação da matriz normalizada com vetor de prioridades; cálculo da média dos valores resultantes da multiplicação com os valores do vetor de prioridades. Assim, o valor de λmax=3.05. Aplicando a fórmula obtém-se o resultado 0,025. Segue-se o cálculo de RC, sendo RC=IC/0.58=0.025/0.58=0.04. Como 0.04 < 0.1 conclui-se que os valores das prioridades relativas estão consistentes.

Na fase seguinte definem-se as matrizes de comparação para cada um dos critérios, tendo em conta cada uma das soluções estudadas. A Tabela 6, a Tabela 7 e s Tabela 8 apresentam as matrizes de comparação entre os diferentes critérios de seleção.

Tabela 6: Matriz de comparação para o critério Tempo

Tempo				
	Git Flow	**GitHub Flow**	**GitLab Flow**	**TBD**
Git Flow	1	1/4	1/2	1/4
GitHub Flow	4	1	2	1
GitLab Flow	2	1/2	1	1/2
TBD	4	1	2	1

Tabela 7: Matriz de comparação para o critério Adequação

Adequação				
	Git Flow	**GitHub Flow**	**GitLab Flow**	**TBD**
Git Flow	1	3	1/2	2
GitHub Flow	1/3	1	1/4	1/3
GitLab Flow	2	4	1	3
TBD	1/2	3	1/3	1

Tabela 8: Matriz de comparação para o critério Complexidade

Complexidade				
	Git Flow	**GitHub Flow**	**GitLab Flow**	**TBD**
Git Flow	1	1/4	1/2	1/4
GitHub Flow	4	1	2	1
GitLab Flow	2	1/2	1	1/2
TBD	4	1	2	1

Para a definição destas matrizes consideraram-se valores inversos para os critérios de Tempo e Complexidade, uma vez que nestes casos, quanto maior o valor relativo, menos adequada é a solução.

Definidas as matrizes, foram calculados os resultados com o auxílio da ferramenta open-source *Analytic Hierarchy Process (AHP) Tool* (Anonymous, 2022).

Na Figura 26 são apresentados os resultados da aplicação do método AHP para seleção de melhor solução de políticas de branching nos projetos de implementação de BC.

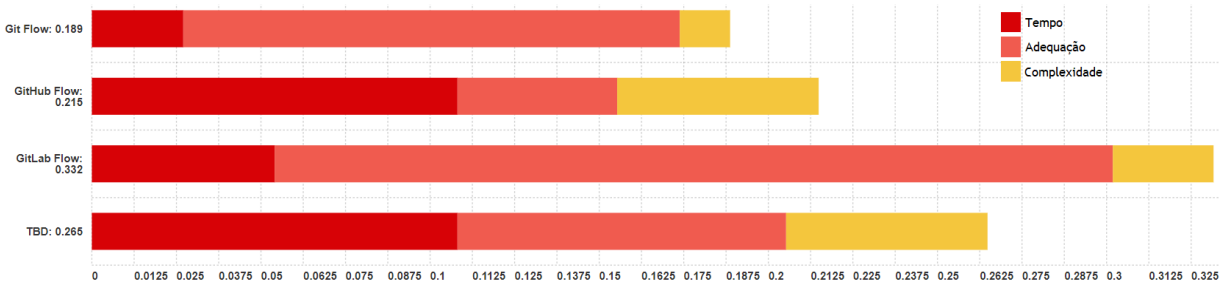


Figura 26: Adequação relativa de diferentes políticas de branching a projetos de implementação do BC.

Como mostra a Figura 26, os pesos relativos finais das 4 soluções são:

- Git Flow: 18,9% - com maior peso do critério Adequação.
- GitHub Flow: 21,5% - com maior peso do critério Tempo.
- GitLab Flow: 33,2% - com maior peso do critério Adequação.
- TBD: 26,5% - com maior peso do critério Tempo.

Conclui-se então que a solução que melhor se adequa a este projeto, seguindo as regras do método AHP, é o GitLab Flow. No entanto, esta solução apenas vence com um peso de 33,2%, o que não é muito convincente visto que apenas foram consideradas 4 soluções. Estes resultados são úteis e dão uma ideia do que melhor se adequa ao projeto, mas não devem ser tomados como verdade absoluta sem considerações adicionais.

4 Desenho

Nesta secção será descrita a arquitetura da solução proposta, assim como a definição de processos desde a fase de desenvolvimento das funcionalidades até à fase de deploy para ambiente de produção. É importante definir estes processos na fase de desenho porque a solução final dependerá dos procedimentos que serão adotados na implementação dos projetos de BC.

4.1 Definição de processos

Aqui serão descritos os novos procedimentos de desenvolvimento de projetos em BC. O objetivo é definir um workflow para o desenvolvimento de alterações e funcionalidades nos projetos.

4.1.1 Bases de dados

Começando pela estrutura de bases de dados, cada projeto terá, já à semelhança de alguns projetos atuais da myPartner, 4 bases de dados:

- DEV**:** Será sobre esta BD que as alterações e funcionalidades serão desenvolvidas. Será utilizada principalmente pelos developers.
- TEST**:** Quando os developers considerarem que a funcionalidade está pronta para os testes funcionais, será migrada para TEST. Esta BD será utilizada essencialmente pela equipa de consultoria que testa as funcionalidades.
- PrePROD**:** Após serem feitos todos os testes necessários para garantir a fiabilidade da funcionalidade, antes de a passar para produção, esta deverá ser sujeita a aprovação do cliente. Para tal será utilizada a BD de PrePROD, esta BD deverá ser atualizada periodicamente com os dados de PROD. Pode também ser utilizada tanto pelas equipas

de desenvolvimento como de consultoria para análise de problemas e bugs que acontecem em produção, permitindo replicar cenários mais complexos que não se conseguem replicar na BD de TEST.

- PROD: Após aprovação do cliente, a funcionalidade poderá passar para produção.

A Figura 27 mostra uma representação visual deste workflow. Este fluxograma apenas inclui processos não automáticos.

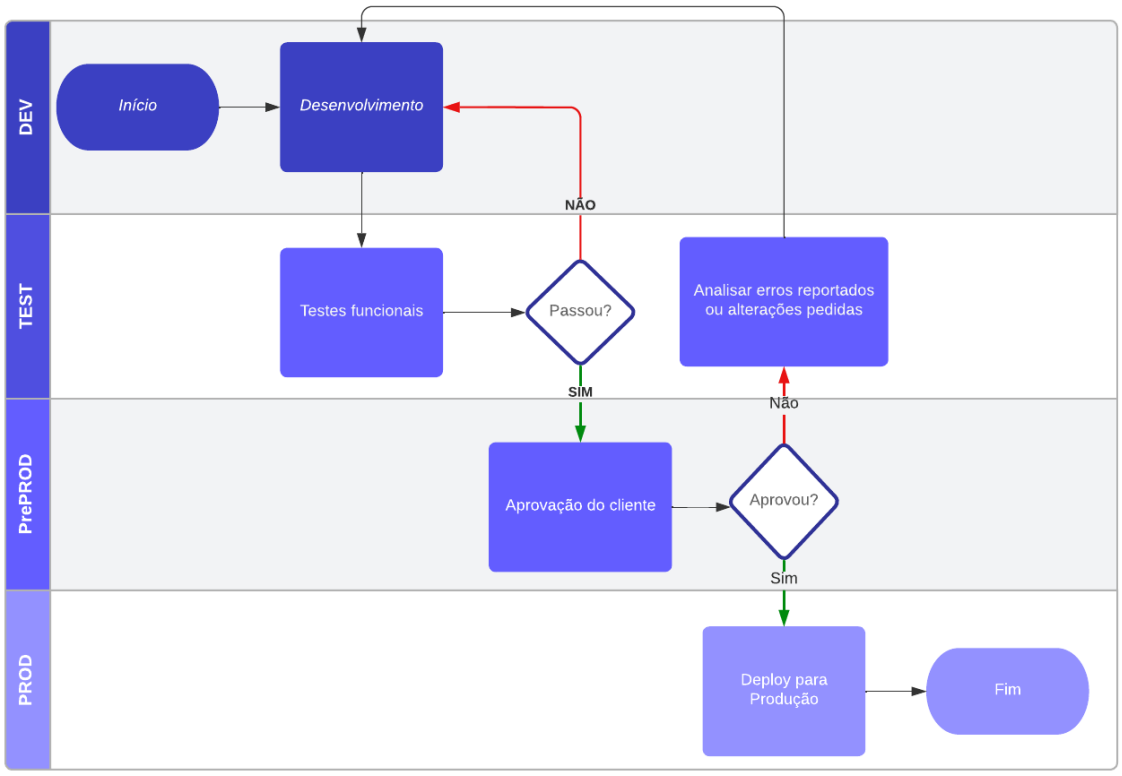


Figura 27: Fluxograma representando o workflow de desenvolvimento de customizações ao BC.

4.1.2 Políticas de branching

Como em qualquer projeto de desenvolvimento de software, a definição de um modelo de branching organizado e consistente é indispensável.

Para tal, serão definidos 2 branches permanentes que deverão existir em todos os projetos:

- **Main/master:** Contém todo o código relativo às funcionalidades que já estão instaladas em ambiente de produção ou alterações que já estão aprovadas tanto internamente como pelo cliente e estão prontas para serem colocadas em produtivo, todas as alterações puxadas para este branch serão sujeitas a revisão de código e aprovação (pull request).

- **Staging:** Contém código relativo às funcionalidades que já passaram a fase de testes internos e que estão prontas para validação do cliente, alterações puxadas para este branch serão também sujeitas a pull request.

Estes 2 branches irão conter o código de desenvolvimentos já em fase final ou terminados. Para a fase de desenvolvimento existirão 2 outros tipos de branches:

- **Feature branches:** Cada desenvolvimento terá um branch associado, sobre o qual serão feitas as alterações ao código.
- **Developer branches:** Uma vez que um desenvolvimento pode envolver vários developers simultaneamente, é boa prática que estes trabalhem em branches diferentes para evitar conflitos.

Periodicamente, pode surgir a necessidade de realizar intervenções urgentes em funcionalidades que já se encontram em produção, seja por bug, ou por necessidade do cliente.

Para tal existirá outro tipo de branch:

- **Hotfix:** Este branch estará ao mesmo nível que o Staging**,** sendo criado diretamente do Main. Como vários developers podem também estar a trabalhar simultaneamente num hotfix, deverão ser criados também developers branches abaixo deste.

A Figura 28 exemplifica a hierarquia desta estrutura de branching.

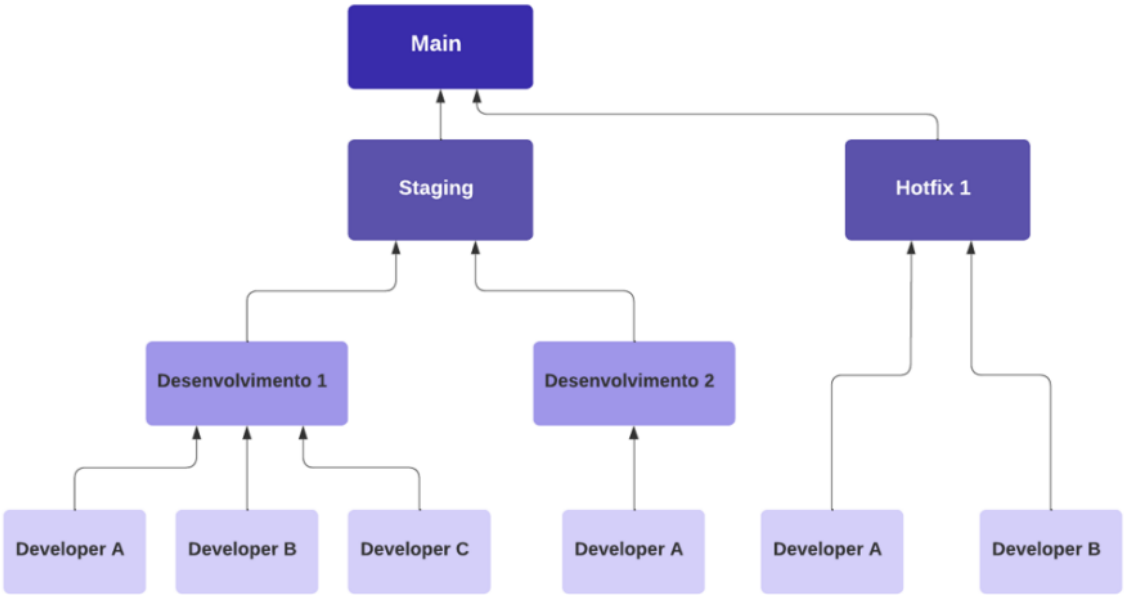


Figura 28: Exemplificação do modelo de *branching* a aplicar.

Neste exemplo, cada caixa representa um branch e as setas representam o fluxo das alterações ao código pelos diferentes branches. No topo da hierarquia encontra-se o master branch, todo o código desenvolvido deverá, eventualmente, terminar aqui.

Cada developer deverá criar um branch individual sempre que efetuar alterações relativas a um determinado desenvolvimento. Desta forma, se estiverem 3 developers a trabalhar no "Desenvolvimento 1", deverão existir 3 branches individuais abaixo do feature branch.

Quando um desenvolvimento é considerado como terminado, deverá ser feito o merge para Staging**.** Quando o desenvolvimento for validado pelo cliente, poderá então passar para master.

Em caso de necessidade de realizar hotfixes, existirá um branch dedicado a este efeito. Cada hotfix terá o seu branch individual que seguirá a mesma lógica dos *feature branches*. Definiu-se que os branches de hotfix ficam diretamente abaixo do master de modo a separar estas alterações de carácter urgente dos restantes desenvolvimentos. No entanto, é também imperativo validar os hotfixes em ambiente de pré-produção antes de os passar para produção, isto será feito normalmente de forma automática pelas pipelines de CD ao fazer deploy para Staging, como será explicado em 4.1.3. Mas nos casos de hotfixes, para já, isto será feito de forma manual (como é feito atualmente), ou seja, o developer instala a extensão em ambiente de pré-produção manualmente, onde esta será testada, e assim que seja aprovada, será feito o deploy para o *master branch*.

Este modelo utiliza aspetos de diferentes modelos analisados em 2.4.2:

- Uma vez que as extensões são instaladas com a opção de desativação no BC, aplica-se o aspeto *Feature Toggles* do modelo TBD.
- O branch *Staging* segue uma lógica semelhante ao branch *Develop* existente no modelo Git Flow, enquanto simultaneamente, servirá de branch de ambiente para Pré-Produção como no modelo GitLab Flow.
- Os branches de desenvolvimento seguirão também a mesma lógica dos *feature branches* recomendados nos modelos Git Flow e GitHub Flow.

4.1.3 Pipelines

As pipelines têm como objetivo automatizar muitos dos processos que seriam normalmente feitos manualmente, como a execução de testes automatizados e o *deploy* das extensões.

Estas pipelines vão ser executadas por um *self-hosted agent* alojado numa máquina virtual a correr nos servidores da myPartner ou do cliente.

Existirão 2 tipos de pipelines: pipelines de continuous Integration (CI Pipelines) e pipelines de continuous deployment (CD Pipelines).

4.1.3.1 CI Pipelines

Para implementação de projetos de BC, apenas será necessária uma CI Pipeline. Esta pipeline será acionada automaticamente pelo Azure DevOps sempre que é feito um merge para os branches Staging e master, garantindo que nenhuma alteração destrutiva chega a esta fase.

Para tal, a pipeline deverá compilar o código, validar alterações destrutivas (Ex: eliminação de campos de tabelas) e executar os testes automáticos associados às extensões incluídas no merge.

4.1.3.2 CD Pipelines

Serão criadas 2 pipelines de CD:

- **Release to PrePROD:** Executada **automaticamente** após cada build do branch Staging para publicar as extensões em ambiente de PrePROD.
- **Release to PROD:** Executada **manualmente** para publicar as extensões em ambiente de produção.

4.2 Requisitos não funcionais

É importante identificar os objetivos não-funcionais de modo a garantir que a solução é capaz de responder às necessidades da empresa:

- **Confiabilidade**: a solução deve ser estável, independentemente das condições que possam afetá-la.
- **Performance**: as execuções das pipelines não devem exceder 1 hora.
- **Funcionalidade**: deve ser possível monitorizar o comportamento das pipelines.
- **Suportabilidade**: a solução deve conseguir suportar a compilação e execução de testes a extensões de BC.
- **Escalabilidade**: deve ser possível fazer alterações e escalar as pipelines de forma não intrusiva conforme necessário.
- **Usabilidade**: a solução deverá poder ser utilizada por qualquer pessoa com as devidas permissões e deve ser de utilização simples e intuitiva.
- **Portabilidade**: é importante que seja possível aceder à solução a partir de vários dispositivos e redes.
- **Segurança**: a ferramenta deverá ter proteções contra ataques informáticos.

De modo a responder a todos pontos, é importante que as ferramentas e serviços escolhidos para implementação da solução tenham estes requisitos cobertos de raiz.

4.3 Diagrama de casos de uso

A Figura 29 apresenta o diagrama de casos de uso para o sistema de CI/CD desenhado.

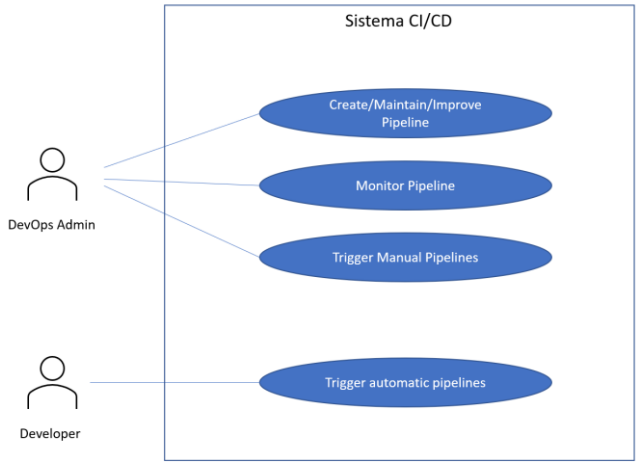


Figura 29: Diagrama de casos de uso do sistema CI/CD

O(s) administrador(es) do sistema serão responsáveis pela criação, manutenção e modificações às pipelines, assim como o monitoramento do seu comportamento e o despoletar das pipelines de ativação manual.

A equipa de desenvolvimento não deverá ter intervenção direta com este processo, no entanto, são as suas ações despoletarão a pipeline automáticas de CI. Sempre que são realizados commits os branches configurados.

4.4 Ferramentas escolhidas

Nesta secção serão apresentadas e justificadas as escolhas para as ferramentas a utilizar para implementação da solução.

4.4.1 Azure DevOps

Com base na análise a metodologias e processos existentes, decidiu-se que a ferramenta a utilizar para implementação das práticas de CI/CD em projetos de BC será o Azure DevOps da Microsoft. Esta ferramenta tem a vantagem de ser um serviço que já é utilizado pela myPartner para gestão de projetos, facilitando o processo de adoção e adaptação às novas metodologias. Para além disso, é a ferramenta de eleição da maior parte dos casos estudados de implementação de CI/CD em projetos BC, em grande parte devido à facilidade de integração com outros serviços e ferramentas da Microsoft, como o VS Code e o próprio ERP Business Central.

O Azure DevOps oferece também as garantias de que os requisitos não funcionais apresentados em 4.2 são satisfeitos.

4.4.2 VS Code

O VS Code é a ferramenta que os developers vão utilizar para desenvolver as extensões, esta componente não faz parte do escopo deste trabalho, no entanto, é também a partir do VS Code que os developers vão criar os seus branches e fazer os seus commits para o repositório alojado no Azure DevOps.

4.4.3 Docker

O Docker oferece a capacidade de montar e desmontar ambientes complexos de forma automatizada e sem recurso a programas externos.

Para a implementar solução proposta, é necessária esta capacidade porque o código-fonte tem de ser compilado e testado num ambiente de Business Central.

Uma vez que queremos automatizar este processo, será utilizado o Docker para criar este ambiente de forma automática sempre que é executada a pipeline de CI.

4.5 Arquitetura

Nesta secção serão descritos os componentes da solução e como estes estão interligados entre si.

4.5.1 Integração continua e implantação das extensões.

A Figura 30 mostra o workflow de CI/CD para projetos de BC da solução proposta.

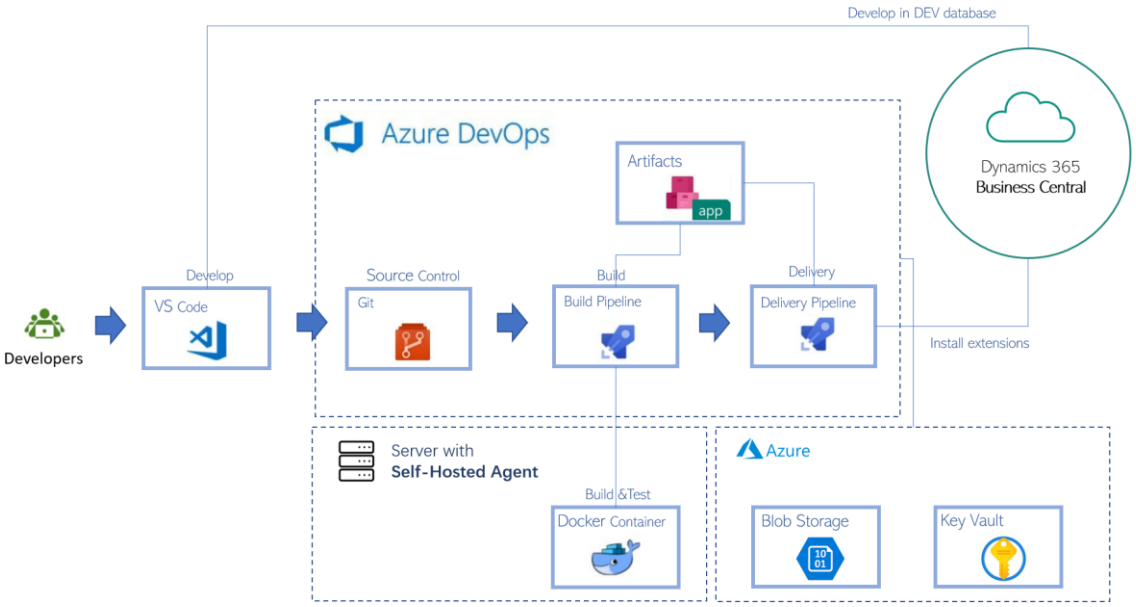


Figura 30: Workflow de CI/CD da solução proposta.

O processo começa na fase de desenvolvimento. Os developers utilizarão o VS Code para submeter as suas alterações para o repositório Git alojado no Azure DevOps.

A partir daqui, será executada a pipeline de CI na máquina que tem o respetivo *Self-Hosted Agent* instalado. Esta máquina deverá também ter o Docker instalado, é esta a ferramenta que a pipeline utilizará para montar o ambiente de BC de modo a poder compilar e executar os testes automáticos. Por fim, são gerados os artefactos resultantes pipeline. Estes artefactos incluem ficheiros ".app", pacotes de execução e os resultados dos testes.

Na fase seguinte, a pipeline de CD deverá instalar a extensão no ambiente de produção alojado na cloud.

Informações e dados sensíveis necessários à execução das pipelines estarão alojados em *Key Vaults* e *Azure Blob Storage*, aos quais o repositório do Azure DevOps deverá ter acesso. Este tema será abordado com mais detalhe no capítulo 5.3 *Segurança de informação sensível*.

4.5.2 Pipeline de CI

Serão criadas 2 pipelines de CI, ambas serão idênticas no seu comportamento, a única diferença sendo o branch onde operam. Uma será direcionada ao branch *Staging*, e outra para o *master*.

A Figura 31 mostra o diagrama de sequência dos processos executados pela pipeline de CI para compilar e testar as alterações.

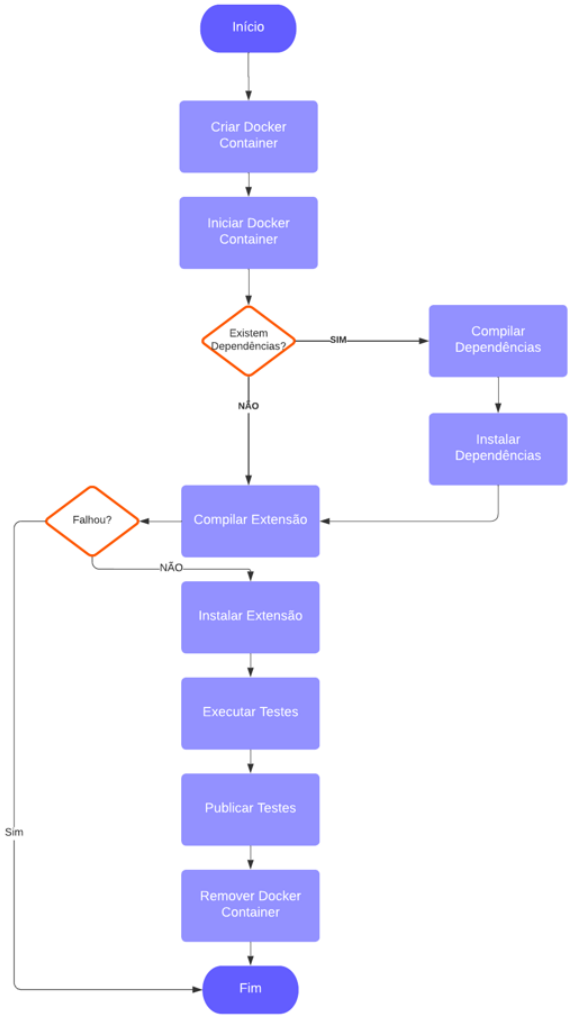


Figura 31: Diagrama de sequência da pipeline de CI

A primeira tarefa a ser executada pela pipeline de CI é a criação e arranque do contentor Docker. Este contentor será um ambiente sandbox do Microsoft Dynamics 365 Business Central.

Como já foi referido, este processamento ocorre na máquina que tem o *Self-Hosted Agent* usado pela pipeline instalado. Por esta razão, o Docker já deverá estar instalado na máquina para que a pipeline possa ser executada.

Com o contentor iniciado, a pipeline instalará as extensões às quais existem dependências, caso existam.

De seguida, a extensão será compilada e, caso este passo seja bem-sucedido, instalada no BC do contentor.

O próximo passo será a execução dos testes automáticos.

No final, o contentor será descartado para não ficar a consumir recursos da máquina.

4.5.3 Pipeline de CD

A Figura 32 mostra o diagrama de sequência dos processos executados pela pipeline de CD para publicar a extensão em ambiente de produção.

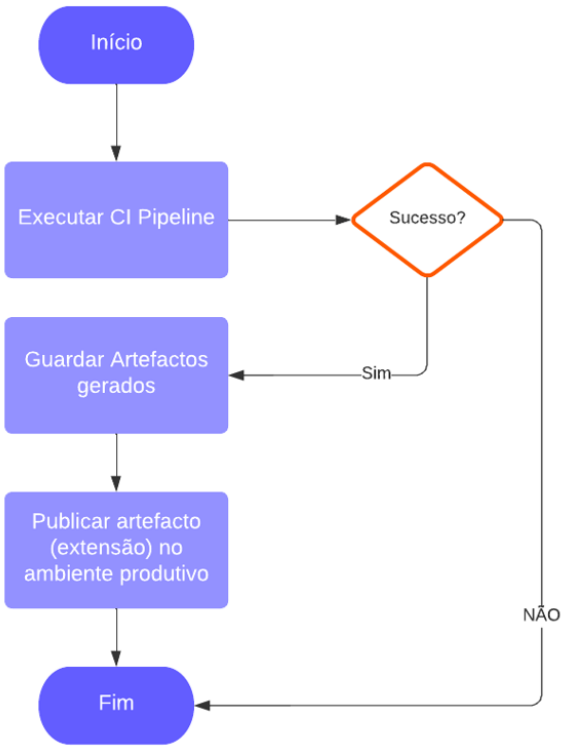


Figura 32: Diagrama de sequência da pipeline de CD

O primeiro processo executado é a pipeline de CI, que voltará a compilar os objetos e executar os testes automáticos.

Caso este passo seja bem sucedido, serão gerados os novos artefactos que incluem o ficheiro .app, este ficheiro que é a extensão no seu formato instalável num ambiente de BC.

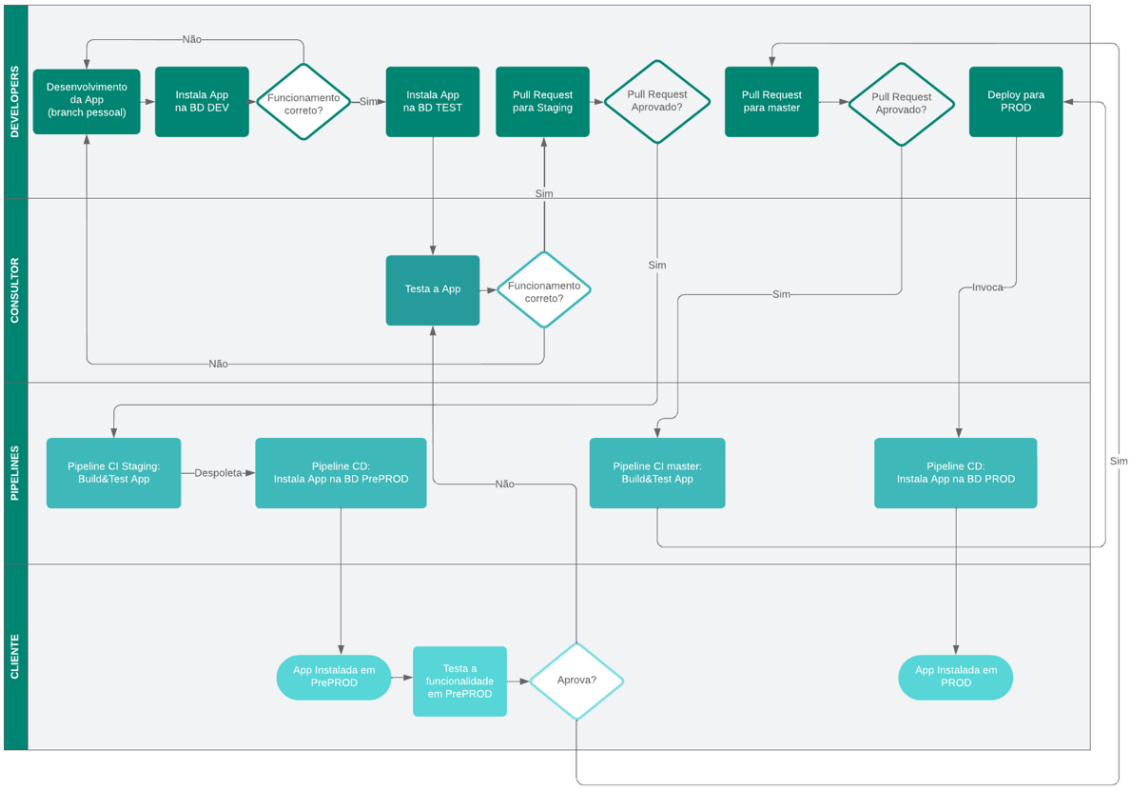
Por fim, o artefacto será publicado no ambiente produtivo de acordo com os parâmetros configurados na pipeline (servidor, base de dados, *tenant*, empresa).

4.6 Visão Geral

Nos subcapítulos anteriores falou-se da definição de processos e da arquitetura das pipelines a implementar de forma isolada. Aqui, será dada uma visão global de como e onde estes processos e pipelines se encaixam no workflow global do processo de desenvolvimento em projetos BC.

A Figura 33 ilustra o fluxo de tarefas e processos que constituem o plano desenhado para desenvolvimento de extensões de BC nos projetos de implementação da myPartner. O fluxo começa com o desenvolvimento da App num branch pessoal do developer, e termina com a instalação da funcionalidade em ambiente produtivo.

Figura 33: Modelo de fluxo do desenvolvimento em BC nos projetos de implementação da myPartner



O processo de desenvolvimento começa com o developer criando o seu branch pessoal onde serão feitas as suas alterações. O developer deve ir testando o funcionamento da aplicação à medida que a desenvolve, para isso, deve instala-la na BD de DEV do cliente e garantir que os testes mais básicos ao seu funcionamento foram feitos.

Quando o developer considerar que a funcionalidade está pronta para testes mais rigorosos, deve instala-la na BD de TEST para que o consultor possa aí avaliar se esta tem o comportamento correto.

Se for este o caso, o consultor dá indicação ao developer de que pode passar o desenvolvimento à próxima fase. Aqui, o developer deverá fazer um pull request para o branch *Staging*. Um outro developer deverá rever o código desenvolvido e, caso aprove as alterações, será despoletada a pipeline de CI que vai compilar e correr os testes automáticos.

Em caso de sucesso, será despoletada automaticamente uma pipeline de CD que instalará a aplicação na BD de PrePROD. Aqui, o cliente poderá testar e avaliar a aplicação desenvolvida.

Caso o cliente aprove, o developer poderá então passar o desenvolvimento para produtivo. Para isso, deverá fazer um novo pull request para o branch *master*. Quaisquer alterações neste branch, à semelhança do *Staging*, despoletarão a pipeline de CI.

Para completar o processo, o developer deverá despoletar manualmente a pipeline de CD para ambiente produtivo.

5 Implementação

Neste capítulo serão apresentados todos os processos e passos realizados para implementar a solução desenhada num repositório de protótipo.

Para este efeito, foi criado um projeto no Azure DevOps contendo o código fonte de uma extensão simples contendo testes automáticos.

Foi também necessário instalar a extensão ALOps no Azure DevOps.

5.1 Pré-requisitos

Para criar este protótipo de uma implementação de CI/CD, foi necessário montar alguns ambientes.

5.1.1 Projeto de Azure DevOps

O primeiro passo na realização deste protótipo foi a criação de um novo projeto no Azure DevOps. O projeto criado foi disponibilizado publicamente e pode ser acedido externamente através do link: https://dev.azure.com/bfalvesmypartner3/ALOps%20Thesis.

Neste projeto, foi adicionado código fonte correspondente a 2 extensões (ver Figura 34):

- Uma extensão denominada “Hello World App” que simplesmente mostra a mensagem “Hello World” no Business Central. Esta extensão está dentro da pasta *app*.
- Uma extensão "Hello World App Test" que simula testes à extensão “Hello World App”. Esta extensão está dentro da pasta *test*.

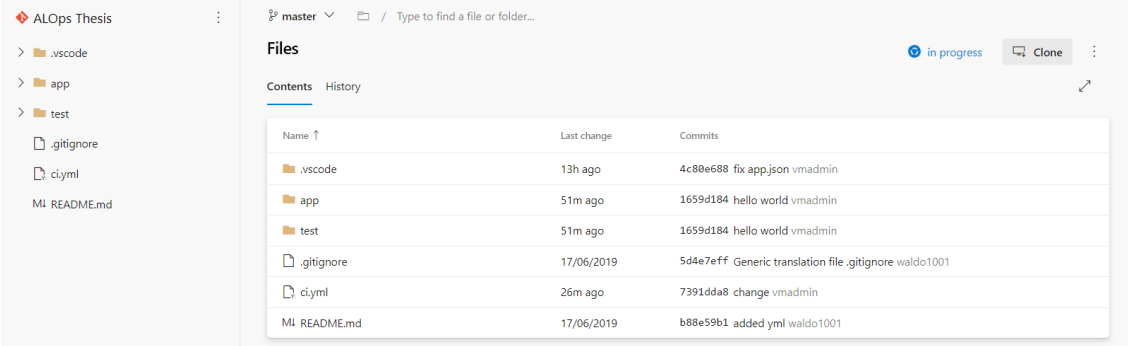


Figura 34: Repositório para o protótipo

Nesse repositório, criou-se o branch *Staging*. Este, juntamente com o branch *master* formarão a estrutura fixa de branching nos projetos de BC (apresentados na Figura 35) como explicado em 4.1.2. Outros branches poderão ser criados e removidos no decorrer dos projetos.

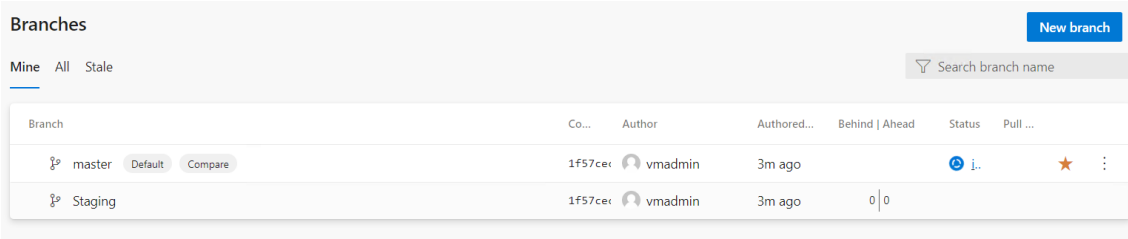


Figura 35: Estrutura fixa de branching para o protótipo

Quaisquer alterações puxadas para estes dois branches serão sujeitas a revisão de código e aprovação. Para tal definiu-se um número mínimo de 1 aprovador como mostra a Figura 36.

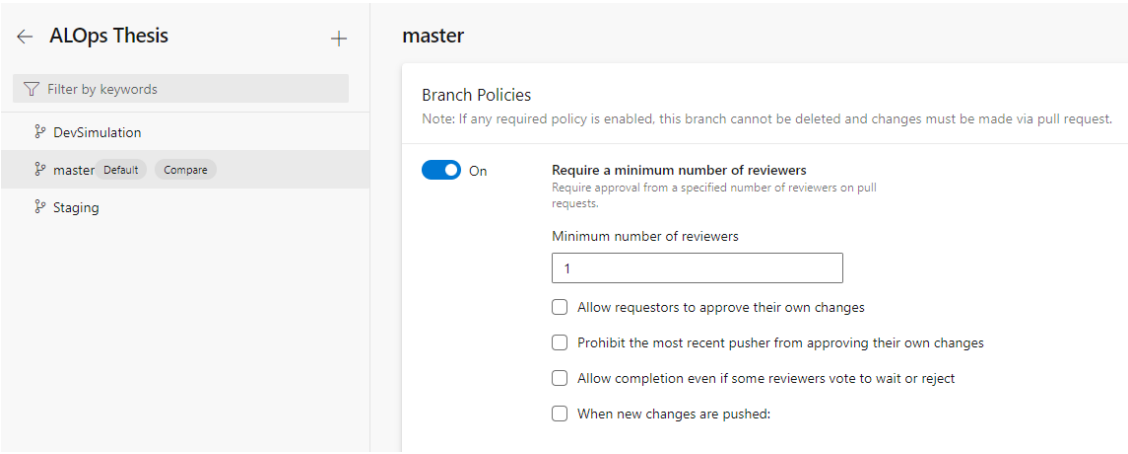


Figura 36: Definição do número mínimo de revisores de código

5.1.2 Ambiente de Sandbox do Business Central

Foram ainda criados 2 ambientes de Sandbox do Business Central (ver Figura 37):

- **PreProd**: Simulação de uma base de dados Pré-Produtiva em cliente.
- **Prod**: Simulação de uma base de dados Produtiva em cliente.

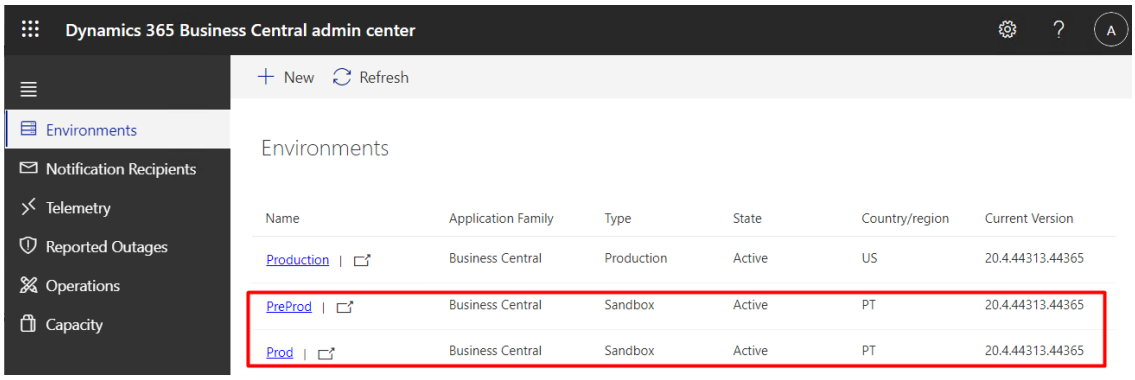


Figura 37: Ambientes de BC Sandbox criados

5.2 Configuração do agente

O agente pode ser alojado numa máquina virtual Azure, mantida pela Microsoft, ou num servidor local. Neste caso, optou-se por alojar o agente num servidor local de modo a ter mais controlo sobre a máquina.

Para tal, no Azure DevOps, criou-se uma *Agent Pool* do tipo *Self-Hosted*, e nela criou-se o agente que vai ser utilizado pelas pipelines.

Ao criar o agente, o Azure DevOps disponibiliza um ficheiro .zip para download que deverá ser instalado na máquina alojadora.

Na máquina, foi criada a pasta **C:\agent** e extraiu-se os conteúdos do ficheiro .zip para aqui.

A Figura 38 mostra os conteúdos desta pasta.

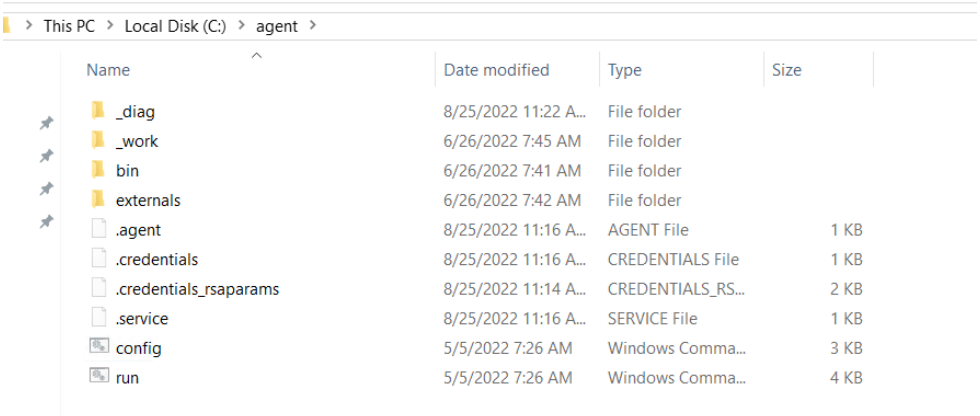


Figura 38: Pasta do agente na máquina alojadora

Para configurar e registar o agente, foi executado o ficheiro *config.cmd* via linha de comandos e, para ativá-lo, executou-se o ficheiro *run.cmd* como mostra a Figura 39.

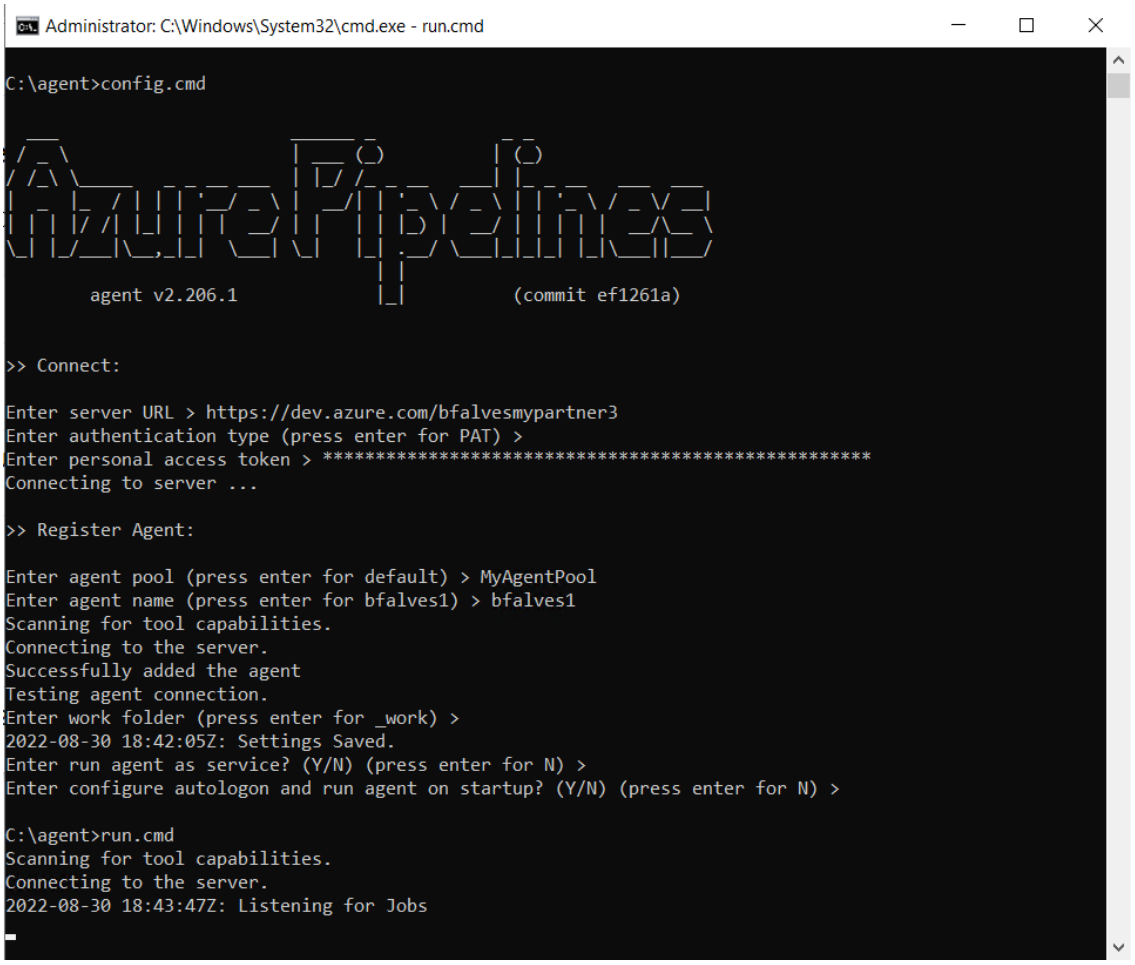


Figura 39: Configuração do agent no servidor

Após ter o agente ativado, o DevOps deteta que este já está disponível para utilização e mostra-o com o estado online.

A Figura 40 e a Figura 41 mostram a *Agent Pool* e o agente criado operacional.

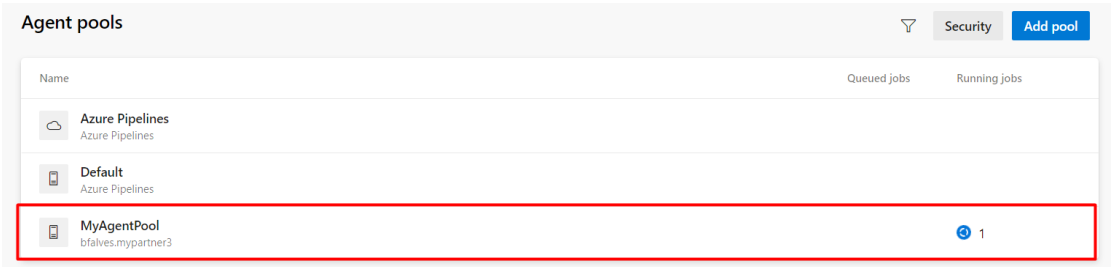


Figura 40: Visualização da Agent Pool criada

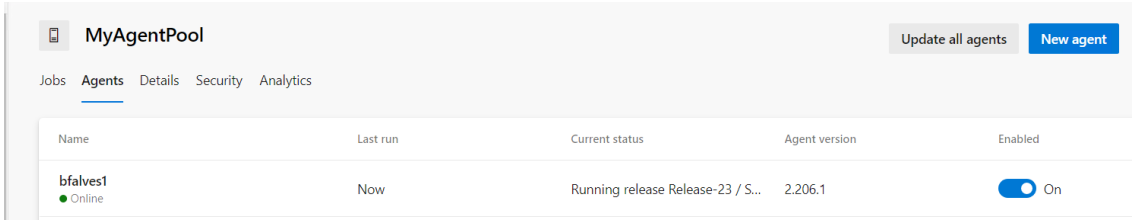


Figura 41: Visualização do agente instalado e operacional

5.2.1 Instalação do Docker

Após ter o agente instalado, é necessário instalar o Docker na máquina. Para tal foram executados os seguintes cmdlets.

```
Install-WindowsFeature Hyper-V, Containers -Restart
Install-PackageProvider -Name NuGet -MinimumVersion 2.8.5.201 -Confirm:$false -Force
Install-Module DockerProvider -Confirm:$false -Force
Install-Package Docker -ProviderName DockerProvider -Confirm:$false -Force
```

5.3 Segurança de informação sensível

De modo a garantir a segurança de dados sensíveis, foi criado um *Key vault* no Azure.

Um *Key vault* permite guardar *Secrets*. Um *Secret* é uma string de texto que pode representar informações sensíveis tais como tokens, códigos de autenticação ou links seguros, todas as informações sensíveis cujo acesso deve ser restringido a um grupo reduzido de pessoas com as devidas permissões foi colocada dentro do *Key Vault*.

A Figura 42 mostra o Key vault e os respetivos *Secrets* criados que serão usados pelas pipelines de build e release.

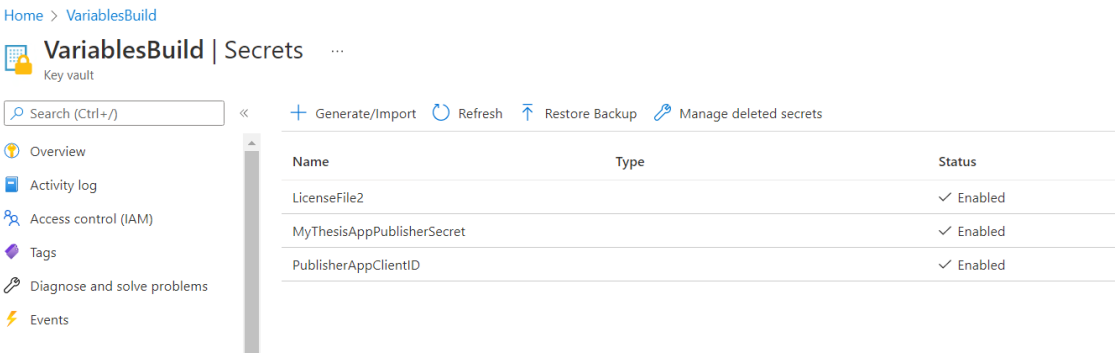


Figura 42: *Key vault* criado e os *secrets* contidos

Para garantir a segurança de ficheiros, utilizou-se a funcionalidade de *Blob Storage* do Microsoft Azure.

Blob Storage está otimizado para armazenar grandes quantidades de dados não estruturados. Dados não estruturados são dados que não aderem a um determinado modelo ou definição de dados, como texto ou dados binários. (Microsoft, 2022j)

Utilizou-se esta funcionalidade para guardar a licença de desenvolvimento em BC da myPartner, gerando uma *Shared Access Signature* (SAS) que foi guardada no *Key Vault* para que as pipelines tenham acesso ao ficheiro.

O URI para uma SAS consiste no URI para o recurso para o qual o SAS delegará acesso, seguido pelo token SAS.

O token SAS é a string de consulta que inclui todas as informações necessárias para autorizar uma solicitação. O token especifica o recurso que um cliente pode aceder, as permissões concedidas e o período de tempo durante o qual a assinatura é válida.

A SAS também pode especificar o endereço IP com suporte ou o intervalo de endereços do qual as solicitações podem ser originadas, o protocolo com suporte com o qual uma solicitação pode ser feita ou um identificador de política de acesso opcional associado à solicitação. (Microsoft,2022k)

A Figura 43 mostra o software *Microsoft Azure Storage Explorer*, utilizado para guardar a licença em *Blob Storage* e gerar a respetiva SAS.

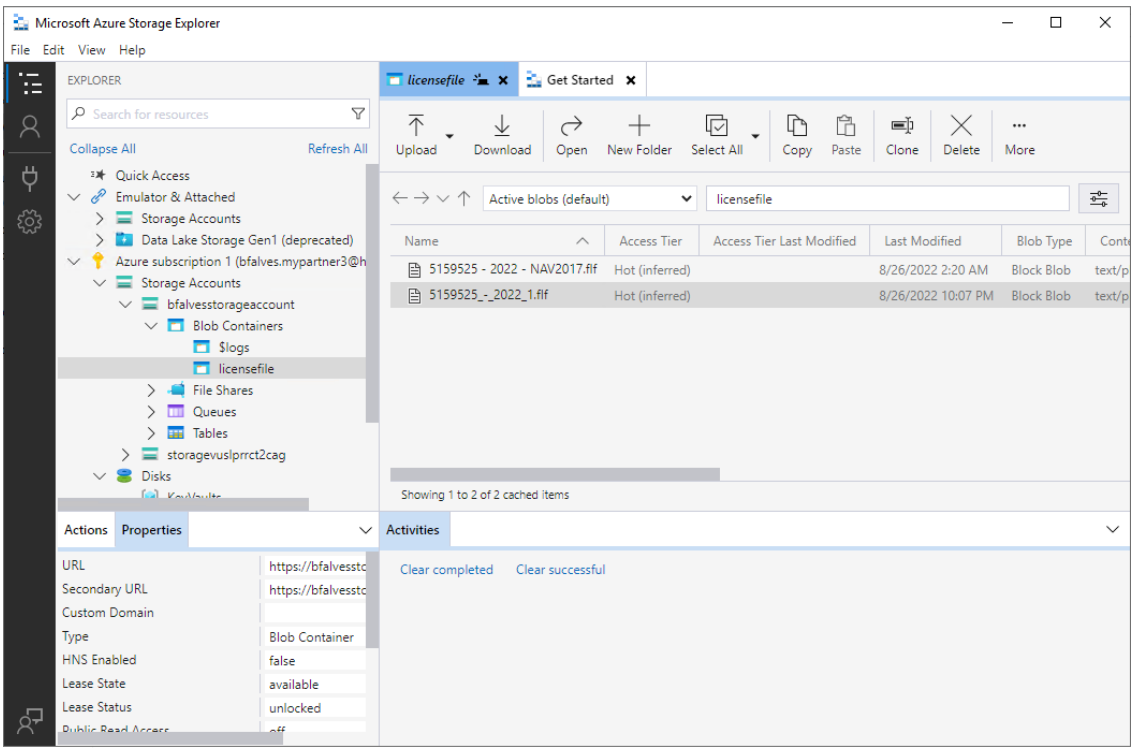


Figura 43: Licença de desenvolvimento Business Central da myPartner em *Blob Storage*

5.4 Pipelines

Neste subcapítulo será descrito o processo de implementação das pipelines de CI e de CD criadas para o protótipo.

5.4.1 Acesso a dados sensíveis

Antes de avançar com a criação das pipelines, é importante estabelecer uma conexão ao *Key vault* que contém os *Secrets* necessários à execução das mesmas, como por exemplo, a licença do BC.

Isto é feito através da criação de um grupo de variáveis conectado à subscrição Azure sobre a qual está criado o *Key Vault* e, de seguida, indicando os *Secrets* aos quais as pipelines deverão ter acesso.

A Figura 44 mostra a criação deste grupo de variáveis e a seleção dos *Secrets* definidos em 5.3.

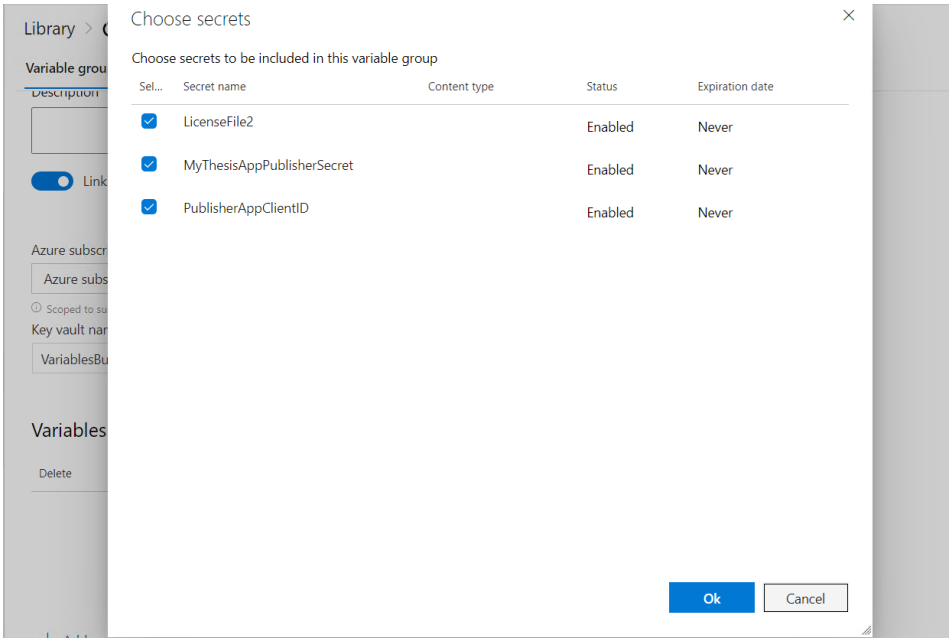


Figura 44: Criação do grupo de variáveis

5.4.2 Pipeline de Continuous Integration

A pipeline de CI está definida num ficheiro YAML (ci.yml) que invoca scripts de Powershell incluídos na extensão ALOps.

No ficheiro definem-se os vários passos que constituem a pipeline:

No cabeçalho, definem-se os branches que deverão despoletar esta pipeline, o grupo de variáveis que contém os *Secrets* necessários, e o agente que a vai executar:

```
name: $(Build.BuildId)

trigger:
- master

variables:
- group: 'VariablesBuild'

pool: MyAgentPool

steps:
- checkout: self
  clean: true
```

De seguida, descrevem-se os passos que a pipeline segue para realizar o processo de integração contínua:

1. Criação do Contentor Docker com a imagem de um ambiente Business Central;

```
- task: ALOpsDockerCreate@1
  displayName: 'ALOPS - Create Docker Image'
  inputs:
    artifacttype: Sandbox

- task: ALOpsDockerStart@1
  displayName: 'ALOPS - Start Docker Container'

- task: ALOpsDockerWait@1
  displayName: 'ALOPS - Wait for Docker Container to start'
  inputs:
    search_string: 'Ready for connections!'
```

2. Importação da licença do Business Central a partir do *Key Vault* criado;

```
- task: ALOpsLicenseImport@1
  displayName: 'ALOPS - License Import'
  inputs:
    usedocker: true
    license_path: '$(LicenseFile2)'
```

3. Importação extensão Test Toolkit, esta extensão é instalada no BC e é necessária para que os testes possam ser executados;

```
- task: ALOpsAppPublish@1
  displayName: 'ALOPS - Install AL TestTool'
  inputs:
    usedocker: true
    installaltesttool: true
    skip_verification: true
    install_al_app_names: |
     Tests-TestLibraries
     System Application Test
     System Application Test Library
     Any
     Library Assert
```

```
    Test Runner
    Permissions Mock
    Library Variable Storage
```

4. Compilação e instalação da extensão “Hello World App”. De cada vez que a extensão é compilada, é-lhe atribuída um novo número de versão. Isto é importante porque o artefacto gerado desta pipeline será usado pela pipeline de release que instala a aplicação no ambiente BC. Ao ter versões diferentes para cada execução da pipeline, é possível reverter facilmente qualquer alteração para a versão anterior com o mínimo impacto, como mostra a Figura 55.

```
- task: ALOpsAppCompiler@1
  displayName: 'ALOPS - Compile Extension: App'
  inputs:
    usedocker: true
    nav_app_version: '1.0.[yyyyWW].*'
    targetproject: 'app/app.json'
    failed_on_warnings: true
    app_file_suffix: '_APP'


- task: ALOpsAppPublish@1
  displayName: 'ALOPS - Publish Extension'

  inputs:
    usedocker: true
    nav_artifact_app_filter: '*.app'
    skip_verification: true
```

5. Compilação e instalação da extensão de testes no BC;

```
- task: ALOpsAppCompiler@1
  displayName: 'ALOPS - Compile Extension: App'
  inputs:
    usedocker: true
    nav_app_version: '1.0.[yyyyWW].*'
    targetproject: 'test/app.json'
    failed_on_warnings: true
    app_file_suffix: '_TEST'

- task: ALOpsAppPublish@1
  displayName: 'ALOPS - Publish Extension'
  inputs:
    usedocker: true
    nav_artifact_app_filter: '*.app'
    skip_verification: true
```

6. Execução dos testes automáticos;

```
- task: ALOpsAppTest@1
  displayName: 'ALOPS - Run TestSuite'
  inputs:
    usedocker: true
    import_action: "Skip"
    import_testtoolkit: false
```

```
    testpage: '130451'
    testsuite: 'DEFAULT'
    failed_test_action: 'Ignore'
  continueOnError: true

- task: PublishTestResults@2
  displayName: 'Publish Test Results **/TestResults.xml'
  inputs:
    testResultsFormat: XUnit
    testResultsFiles: '**/TestResults.xml'
    testRunTitle: 'BC Test Results: $(Build.BuildId)'
```

7. Remoção do contentor criado para execução da Build (para não ficar a consumir recursos da máquina).

```
- task: ALOpsDockerRemove@1
  displayName: 'ALOPS - Remove Docker Container'
  enabled: true
  condition: always()
```

Depois de construído, resta criar a pipeline a partir desse ficheiro, como mostra a Figura 45.

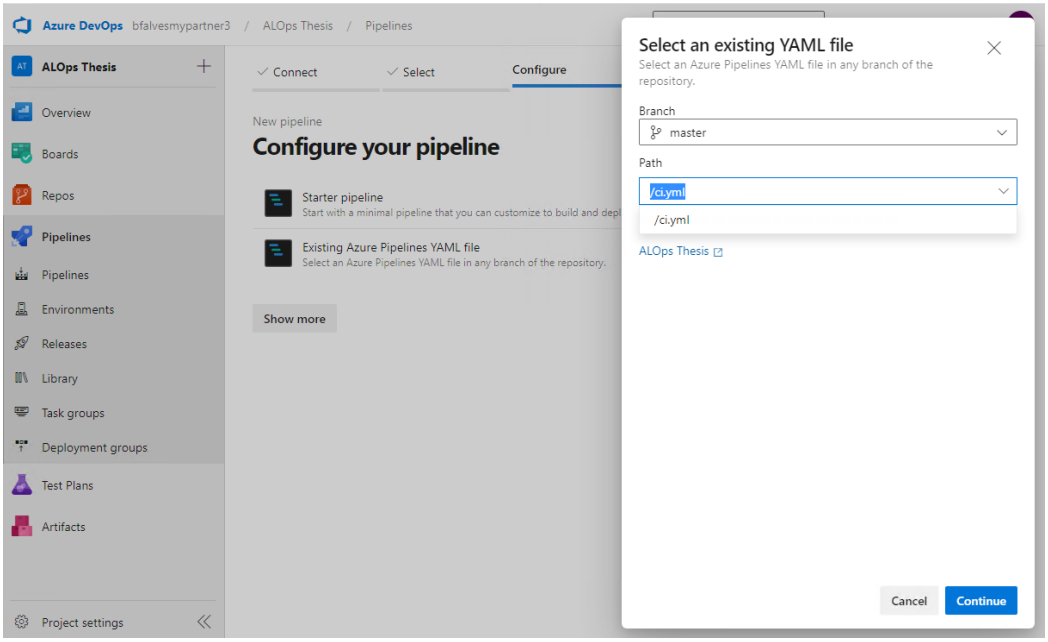


Figura 45: Criação da Pipeline de CI

Foram criadas 2 pipelines de CI (apresentadas na Figura 46):

- **CI_master**: despoletada sempre que são feitas alterações ao branch *master*. Gera o artefacto *_CI_master*.
- **CI_Staging**: despoletada sempre que são feitas alterações ao branch *Staging*. Gera o artefacto *_CI_Staging*.

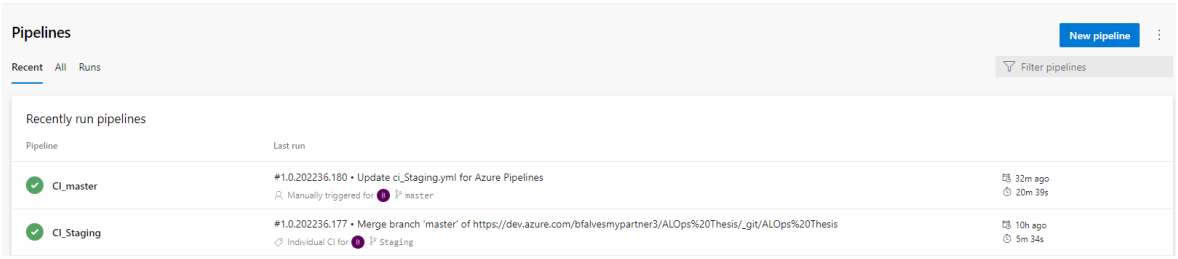


Figura 46: Pipelines de CI criadas

5.4.3 Pipelines de Continuous Deployment para PrePROD e PROD

A pipeline de release para a base de dados de PrePROD será despoletada automaticamente após a pipeline de *CI_Staging* terminar e gerar o artefacto *CI_Staging.app* (extensão "Hello World").

Este artefacto será então importado para o ambiente de PrePROD como extensão. Isto foi feito com recurso à funcionalidade de *Service-to-Service (S2S) Authentication* da Microsoft.

A autenticação S2S é adequada para cenários em que as integrações precisam de ser executadas sem nenhuma interação do utilizador. A autenticação S2S usa o fluxo OAuth 2.0 de credenciais do cliente. (Microsoft, 2022h).

OAuth é um padrão aberto para autorizar o acesso a serviços Web e APIs de clientes e sites nativos no Azure Active Directory (Azure AD). No Business Central, o OAuth é útil quando o deploy de uma extensão está configurado para autenticação do Azure AD, por meio da própria assinatura do Azure ou de uma assinatura do Microsoft 365. OAuth permite que os utilizadores entrem nos serviços Web do Business Central usando as suas credenciais do Microsoft 365 ou do Azure AD. Caso contrário, eles teriam que usar as suas credenciais da conta do Business Central (nome de utilizador e senha ou chave de acesso à web). (Microsoft, 2022i)

Para fazer esta autenticação, foi necessário efetuar os seguintes passos:

1. Criar um registo de aplicação Azure AD. Isto gera uma identidade (*App ID*) e um token de verificação (*App Secret*) que serão posteriormente utilizados pelas pipelines de release para obter autorização de acesso aos ambientes de BC (ver Figura 47).

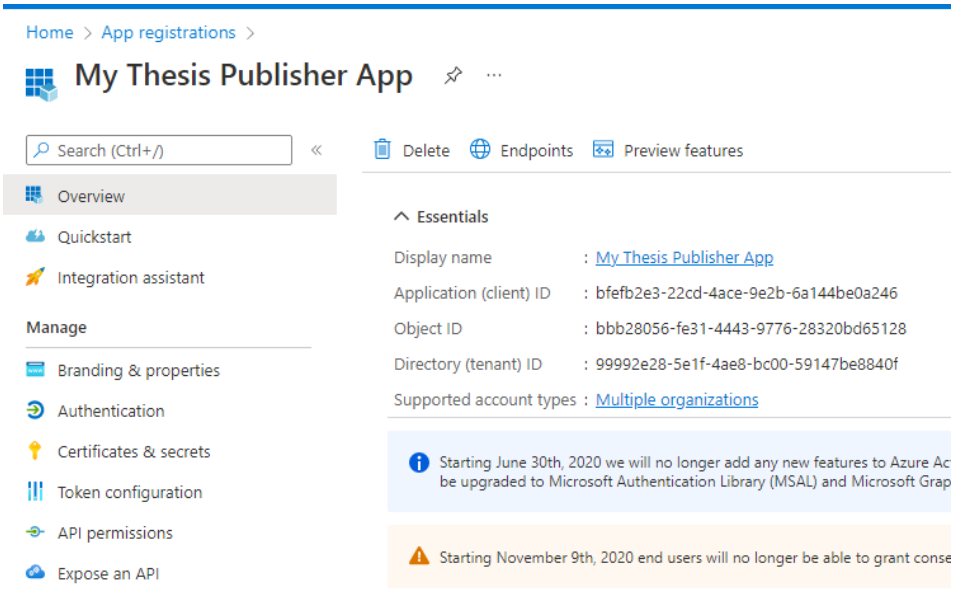


Figura 47: Registo da aplicação no Azure AD

2. Guardar o *App ID* e o *App Secret* no *Key Vault* criado em 5.3. A Figura 42 mostra o *Key Vault* com estes dados já incluídos.
3. No Business Central, registar a aplicação de Azure AD e dar permissões de automação e gestão de extensões (Figura 48), este passo deve ser feito nos 2 ambientes de BC (PrePROD e PROD).

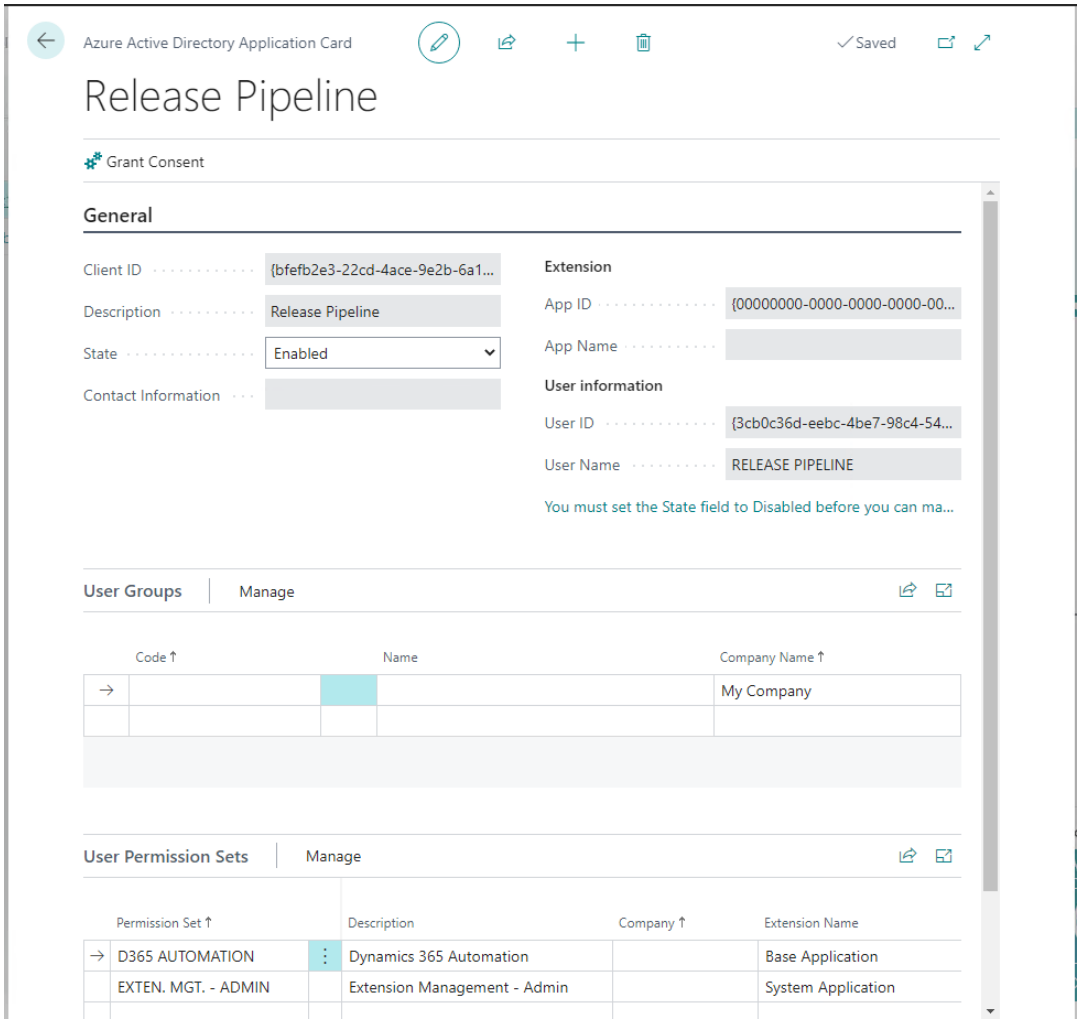


Figura 48: Registo da aplicação Azure AD no Business Central

Após terminada a configuração da autenticação no Business Central, construiu-se a pipeline de release para PrePROD.

O primeiro passo é a indicação do artefacto a ser utilizado pela pipeline que, neste caso, é o último artefacto gerado pela pipeline *CI_Staging*. A Figura 49 mostra a realização deste passo.

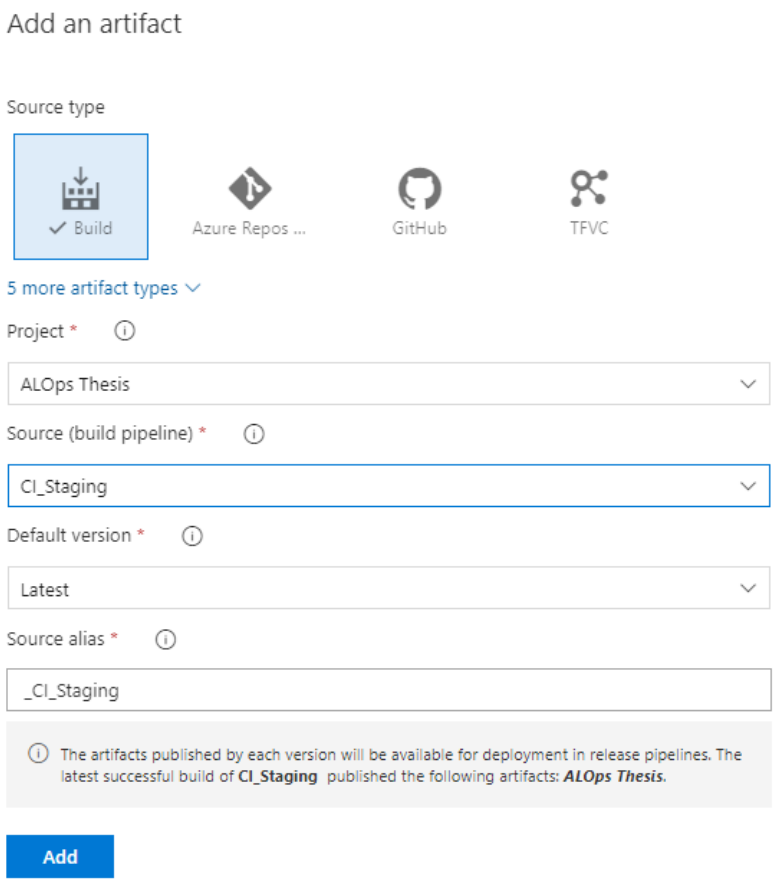


Figura 49: Indicação do artefacto de origem para a pipeline de release para PrePROD

De seguida, foi necessário indicar a tarefa que a pipeline vai realizar. Isto é feito através de um script de PowerShell, executado por um agente.

O script utiliza o módulo *BcContainerHelper* para invocar a função *Publish-PerTenantExtensionApps*. Esta função publica um artefacto de extensão num determinado ambiente de BC online. O script utilizado foi o seguinte:

```
Write-Host "Installing BcContainerHelper"
Install-Module BcContainerHelper -Force -AllowPrerelease
Write-Host "Publishing to tenant"
Publish-PerTenantExtensionApps -useNewLine `
 -ClientID "$(PublisherAppClientID)" `
 -ClientSecret "$(MyThesisAppPublisherSecret)" `
 -tenantId "$(TenantId)" `
 -environment "$(Environment)" `
 -companyName "$(CompanyName)" `
 -appFiles @(Get-Item "_CI_Staging\ALOps Thesis/*_APP.app" | % { $_.FullName })
```

Para que o script funcione, é necessário definir as variáveis *TenantID*, *Environment* e *CompanyName*. Além disso, é necessário dar acesso ao *Key Vault* que contém o *App ID* e o *App Secret* da aplicação Azure AD.

A Figura 50 mostra a definição destas variáveis.

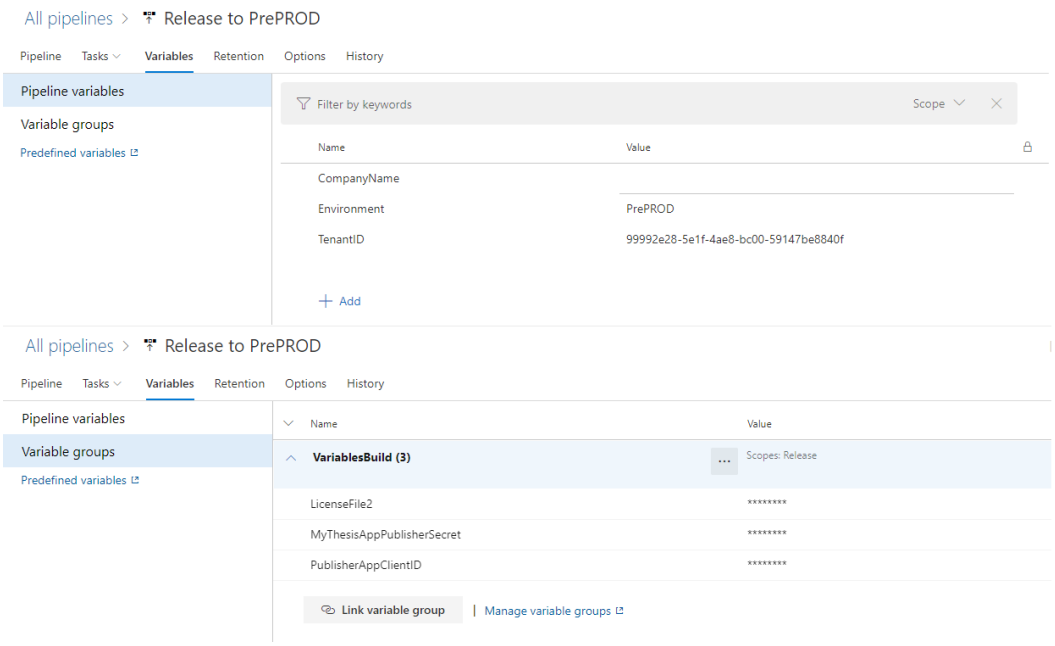


Figura 50: Definição das variáveis da pipeline de release

Para que a pipeline seja executada automaticamente sempre que o branch *master* é alterado, é necessário definir um trigger de Continuous deployment.

A Figura 51 mostra a visão geral da pipeline criada e a definição do trigger.

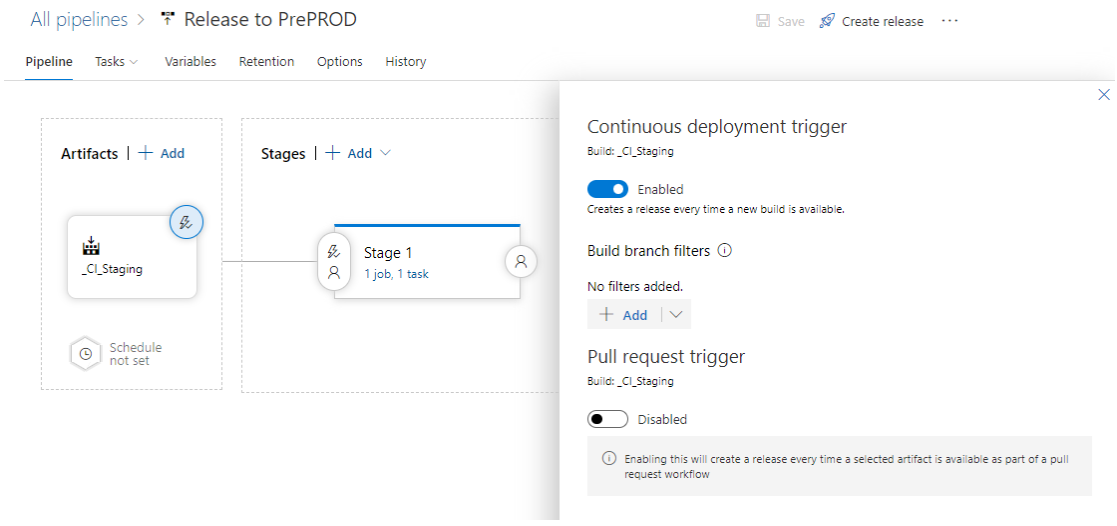


Figura 51: Visão geral da pipeline de release para PrePROD e definição do trigger de Continuous deployment

Para efeitos de protótipo, apenas criou-se 1 stage que faz a instalação da extensão no BC. Futuramente, poderão ser adicionados outros stages como a realização de testes automatizados.

Depois de criada a pipeline de release para PrePROD, repetiram-se os passos para a pipeline de release para PROD, as únicas diferenças são o artefacto de origem que, teste caso, é o CI_master, a variável *Environment*, que passa a ter o valor "PROD", e o facto de esta pipeline ser de ação manual significa que não se define o trigger de Continuous deployment.

A Figura 52 apresenta as 2 pipelines de release criadas para o protótipo.

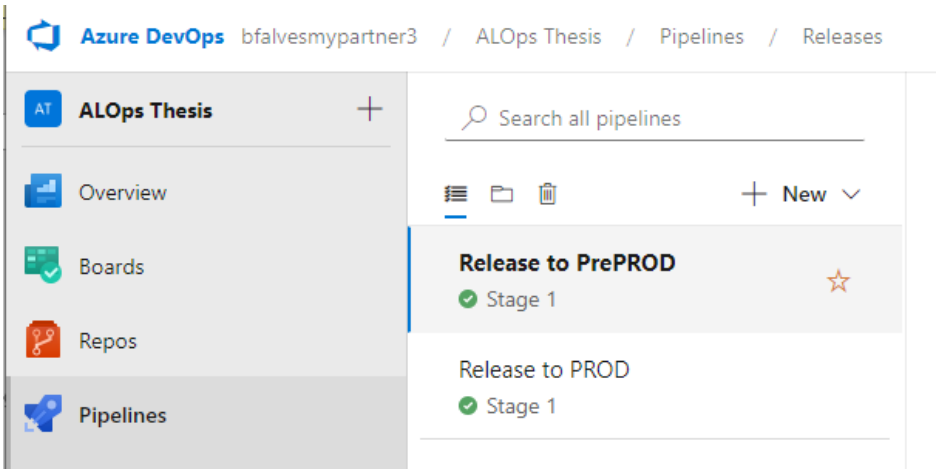


Figura 52: Pipelines de Release criadas

5.5 Resultados

As pipelines de *CI_Staging* e *CI_master* são executadas automaticamente a cada alteração nos branches *Staging* e *master* respetivamente.

A Figura 53 apresenta os resultados de uma execução da pipeline *CI_Staging*. Como é possível verificar, todas as tarefas foram realizadas com sucesso, e no final foi produzido um artefacto.

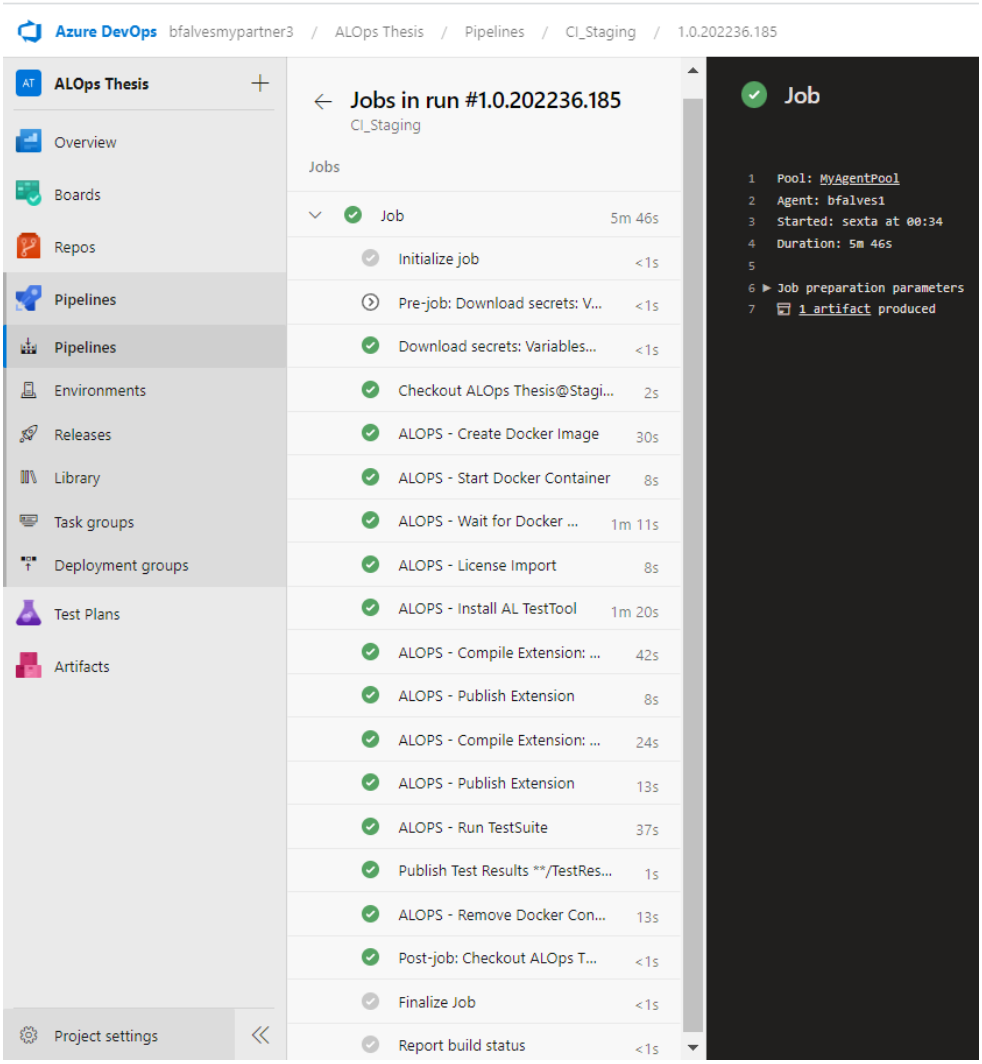


Figura 53: Resultados de uma execução da pipeline *CI_Staging*

A execução desta pipeline, por sua vez, despoleta a pipeline *Release to PrePROD*. Esta pipeline pega no artefacto gerado e publica-o no ambiente de PrePROD como extensão. A Figura 54 mostra os resultados da execução desta pipeline.

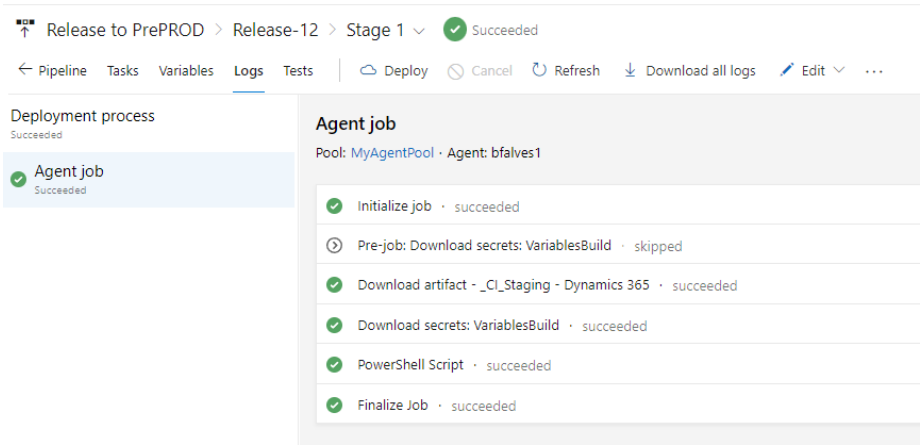
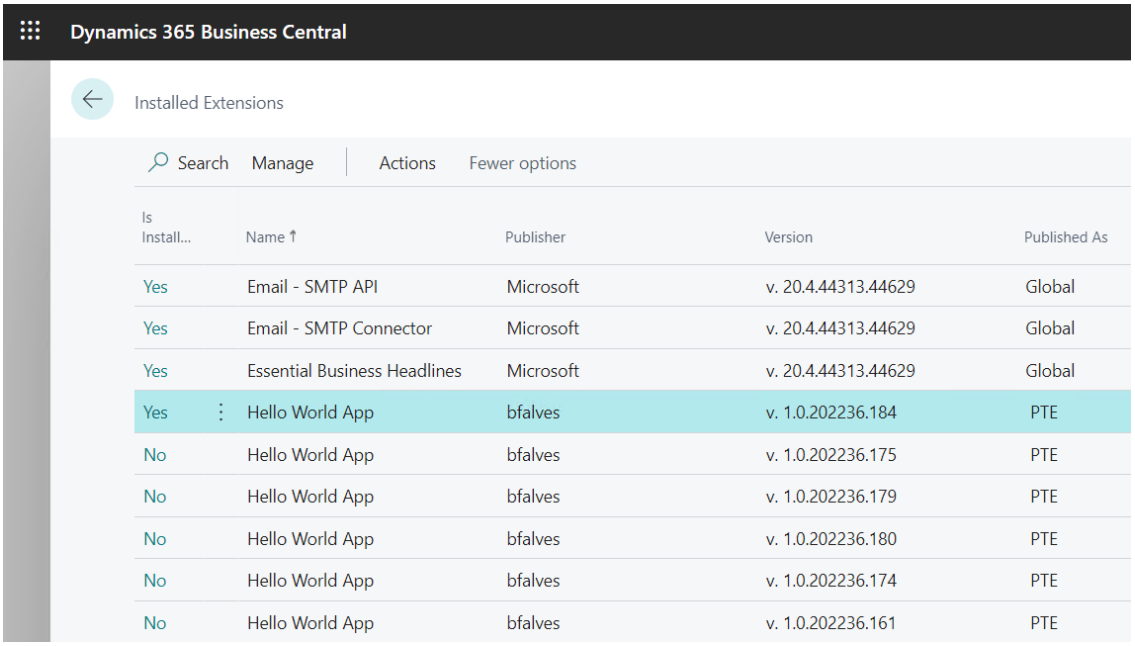


Figura 54: Resultados da execução da pipeline *Release to PrePROD*

Como resultado, no Business Central, é publicada e instalada a extensão. Caso esta já estivesse instalada, a versão anterior seria desativada. Este é um cenário de especial importância no ambiente produtivo porque permite que qualquer alteração com impacto negativo seja facilmente revertida. A Figura 55 mostra a nova extensão instalada e as várias versões anteriores desativas.



Is Install...	Name	Publisher	Version	Published As
Yes	Email - SMTP API	Microsoft	v. 20.4.44313.44629	Global
Yes	Email - SMTP Connector	Microsoft	v. 20.4.44313.44629	Global
Yes	Essential Business Headlines	Microsoft	v. 20.4.44313.44629	Global
Yes	Hello World App	bfalves	v. 1.0.202236.184	PTE
No	Hello World App	bfalves	v. 1.0.202236.175	PTE
No	Hello World App	bfalves	v. 1.0.202236.179	PTE
No	Hello World App	bfalves	v. 1.0.202236.180	PTE
No	Hello World App	bfalves	v. 1.0.202236.174	PTE
No	Hello World App	bfalves	v. 1.0.202236.161	PTE

Figura 55: Extensão instalada no ambiente BC

Quando o cliente aprovar as alterações, poderá então ser despoletada a pipeline *Release to PROD* manualmente.

6 Avaliação

De modo a fazer uma avaliação o mais objetiva possível, foi feita uma retrospetiva aos objetivos principais definidos no início do projeto. No entanto, porque ainda não havia uma ideia concreta de como a solução devia ser, os objetivos iniciais do projeto eram bastante subjetivos e pouco detalhados. Fazendo referência ao subcapítulo 1.3, lê-se: "*Estes objetivos não são particularmente detalhados devido ao facto da solução final pretendida não ter ainda um conjunto de requisitos delineados, sendo o objetivo final mais uma "prova de conceito" do que uma solução pronta a ser utilizada pela empresa*".

Por esta razão, foi feito um levantamento de alguns pontos de avaliação que fazem sentido no contexto da solução proposta e que simultaneamente podem servir de objetivos para o seu desenvolvimento futuro. Na Tabela 9 é feita uma avaliação geral à solução proposta de acordo com estes critérios.

Tabela 9: Critérios e avaliação da solução

Descrição do critério	**Foi cumprido?**
A solução permite compilar e executar testes automáticos através de uma pipeline de CI.	Sim
A solução permite realizar implantações em ambiente de produção automaticamente.	Sim
Em caso de necessidade, é possível reverter alterações colocadas em produção rapidamente com o mínimo de impacto?	Sim (ver 5.5)

A solução permite mais controlo sobre o histórico e ciclo de vida do desenvolvimento de extensões.		Sim
Os requisitos não funcionais (4.2) foram cumpridos.	Confiabilidade	Sim, garantida pelo Azure DevOps
	Performance	Sim, como é possível verificar no subcapítulo 5.5, a pipeline de CI executa em cerca de 5 minutos, e a pipeline de CD apenas leva cerca de 1 minuto e meio.
	Funcionalidade	Sim, garantida pelo Azure DevOps e a extensão ALOps
	Suportabilidade	Sim
	Escalabilidade	Sim
	Usabilidade	Sim, apesar de ser um critério subjetivo, considera-se que o Azure DevOps é uma plataforma de utilização intuitiva
	Portabilidade	Sim, garantida pelo Azure Devops
	Segurança	Sim, garantida pelo Azure Devops e pelas funcionalidades do Azure (*Key Vaults* e *Blob Storage*).
As metodologias de branching definidas fazem sentido no contexto de projetos da myPartner, e ajudam o controlo de qualidade dos serviços.		Sim. No entanto, de modo a poder avaliar este ponto de forma mais objetiva será necessário aplicar estas metodologias num projeto piloto.
Os developers estão satisfeitos com as novas metodologias de trabalho.		Esta questão só poderá ser respondida com fundamento após aplicar a solução num projeto piloto.
O tempo de desenvolvimento de uma nova funcionalidade é encurtado na maioria dos cenários.		Este ponto também só pode ser respondido objetivamente após a solução ser implementada e avaliada num projeto piloto.
A frequência da ocorrência de problemas e erros relacionados com a instalação de novas funcionalidades em ambientes de produção foi reduzida.		Só será possível responder a esta questão de forma objetiva após a utilização da solução proposta por um período mínimo de um ano.

Tendo em conta os critérios enumerados, fica evidente de que uma avaliação completa e objetiva da solução proposta só será possível após a sua implementação num projeto piloto envolvendo os vários intervenientes num projeto típico da myPartner, com uma equipa de desenvolvimento e de consultoria, assim como a supervisão do gestor de qualidade.

7 Conclusão

Nesta dissertação foi proposta uma solução de uma abordagem sistemática para implementação das práticas de Continuous Integration/Continuous Deployment (CI/CD) em projetos de implementação de Business Central (BC). A solução proposta consiste não só no desenvolvimento de pipelines de CI/CD, mas também na definição dos vários processos inerentes ao desenvolvimento de software no contexto de projetos de BC. Desde as políticas de branching, aos diferentes ambientes de desenvolvimento, testes e produção, assim como o ciclo de vida do código-fonte das extensões desenvolvidas e como todos estes conceitos se encaixam num workflow contínuo.

7.1 Objetivos alcançados

Como foi referido no capítulo 6 onde é feita a avaliação da solução, os objetivos iniciais eram pouco detalhados e subjetivos, no entanto, apesar de não serem bons indicadores para avaliação, é possível avaliar se estes foram efetivamente cumpridos ou não.

Desta forma, no subcapítulo 1.3 foram definidos os seguintes objetivos:

- Inicialmente, deverá ser realizado um levantamento do estado da arte dos conceitos e metodologias de CI/CD existentes aplicadas a projetos de BC;

No capítulo 2 é apresentado o estado da arte onde são apresentados os vários conceitos e termos relevantes para o projeto desta dissertação e são avaliadas diversas metodologias já existentes quer de projetos BC como de outros projetos de software mais genéricos.

- Com as informações obtidas deverá ser feita uma análise de como se podem adaptar estas soluções ao contexto da myPartner e/ou definir novos processos caso haja necessidades não correspondidas pelas soluções existentes.

Este objetivo foi também concretizado no capítulo 2 onde são apresentados os processos atuais utilizados pela myPartner e como algumas das metodologias estudadas encaixam nestes processos.

- Após esta análise, pretende-se que seja definida a abordagem que a myPartner deverá aplicar nos projetos BC que iniciar, tendo em consideração todas as questões acima enumeradas.

Na fase de Design, no subcapítulo 4.1, foram definidos os novos processos a aplicar nos projetos de implementação de BC da myPartner desde os vários ambientes que existirão às políticas de branching que deverão ser respeitadas pelos developers para tirar o maior partido das pipelines de CI e CD.

- Por fim pretende-se a criação de um projeto de protótipo no Azure DevOps com a estrutura definida na análise e onde deverão ser aplicadas as regras de branching estabelecidas e criadas as pipelines que ajudarão a automatizar os processos delineados.

Um dos objetivos principais do projeto era a preparação de um repositório de protótipo para um projeto que implementaria a solução proposta. No capítulo 5 está descrita esta implementação, as pipelines criadas, para além de validarem a compilação do código desenvolvido, implantam as alterações para os ambientes de produção e pré-produção sem necessidade de intervenção manual por parte dos developers.

7.2 Problemas/Limitações

Ao longo da realização deste projeto foram surgindo alguns obstáculos que, por vezes, obrigaram ao recuo e mudança na abordagem.

Uma destas ocorrências aconteceu na implementação das pipelines de CI. Inicialmente, não havia intenção de utilizar o ALOps devido ao facto de ser um serviço pago. Em vez disso as pipelines seriam construídas manualmente através de scripts de PowerShell e ficheiros YAML adicionados ao próprio projeto. Estes scripts de PowerShell não foram desenvolvidos de raiz, mas eram scrips disponibilizados pela própria Microsoft num workshop de implementação de pipelines em projetos de BC realizado em 2019, e adaptados para o projeto em mãos.

No entanto, após abrir um *issue* no GitHub devido a um problema com a execução da pipeline de CI, foi comunicado que já não era dado suporte àquela versão e que deveria utilizar ou AL-Go for GitHub, ou ALOps caso pretenda continuar com o Azure DevOps. Como o objetivo era implementar a solução no Azure DevOps optou-se pelo ALOps.

Além de problemas que surgiram durante a implementação, foram identificadas algumas limitações da solução proposta.

Uma destas limitações é o facto de o agente utilizado (subcapítulo 5.2) correr num servidor local e de apenas poder executar 1 job de cada vez. Isto implica que quantos mais projetos

utilizarem este agente e quantos mais developers envolver, maior o risco da fila de espera para executar uma pipeline crescer descontroladamente e eventualmente requerer intervenção manual para cancelar algumas execuções. Isto pode ser colmatado com a criação de vários agentes, no entanto, se forem todos alojados no mesmo servidor poderá sobrecarregar o sistema. Outra opção é a utilização de *Azure Pipelines Agents*, que são alojados pela Microsoft mas que são um serviço pago. Ou então, e talvez a melhor opção, será utilizar o servidor do cliente para alojar o agente que executa tarefas do projeto desse cliente.

Uma outra limitação/problema que poderá surgir e que não foi abordada neste projeto, é o facto de alguns dos clientes da myPartner requererem o uso de Virtual Private Networks (VPNs) para permitir o acesso externo aos seus ambientes de ERP. Isto apresenta um obstáculo à implantação contínua nos ambientes de produção e pré-produção se não existir forma de ligar automaticamente a essas VPNs.

7.3 Trabalho futuro

Como já foi referido, este projeto teve como objetivo principal analisar e avaliar a possibilidade de implementar as práticas de CI/CD em projetos de implementação de Business Central. A componente de implementação realizada, apesar de funcional, tem ainda muito espaço para melhoramentos, e mesmo os processos definidos deverão ser reavaliados ao longo do tempo. Tal como em projetos de implementação do BC, muitas questões e problemas só surgem depois do arranque do projeto em produção. A teoria é importante, mas é na prática que se avalia realmente a adequação dos processos e soluções desenvolvidas.

Por esta razão, é importante implementar a solução desenvolvida num projeto piloto que empregue todos os processos definidos e desenvolvidos.

Com este projeto, obter-se-á feedback muito importante para melhor adaptar a solução à realidade e às necessidades da myPartner.

Após avaliar o comportamento da solução num projeto real, poder-se-ão fazer ajustes aos processos definidos, como as políticas de branching ou workflow de deploy para os ambientes de PrePROD e PROD. Também poder-se-á incorporar o repositório do projeto com a gestão de requisitos do projeto que já é feita no Azure DevOps pela myPartner, isto pode ser feito obrigando todos os commits a terem um caso de uso associado, facilitando o rastreio de alterações.

Outros trabalhos futuros podem incluir a compatibilidade das pipelines de deploy com possíveis VPNs requeridas por alguns dos clientes para acesso aos seus ambientes, a automatização do processo de deploy de hotfixes em ambiente de pré-produção (atualmente está definido em 4.1.2 que este processo será manual), e a adição de novas funcionalidades às pipelines de CI,

como a validação de *breaking changes* (alterações que causam incompatibilidade com código antigo), ou adição de novos stages nas pipelines de CD.

7.4 Considerações finais

A solução desenvolvida oferece um ponto de partida robusto e completo para que a myPartner possa melhorar a qualidade dos seus projetos de implementação de BC enquanto simultaneamente reduz a dependência nos developers para processos importantes como o deploy em ambientes de produção, reduzindo o risco de erro humano e o tempo gasto nestes processos, e facilitando a reversão de alterações erróneas.

A solução representa também a adoção de processos de trabalho mais metódicos e organizados, facilitando a coordenação e gestão dos projetos.

Esta dissertação proporcionou uma oportunidade de aprofundar muitos dos conhecimentos base adquiridos ao longo do meu percurso académico na área da engenharia de software, nomeadamente na área CI/CD, assim como a aquisição de experiência importante em tecnologias e ferramentas como o Azure DevOps, Git, Business Central, Docker, e também conceitos como pipelines, políticas de branching e processos de release.

Referências

(Active BS, 2022) Active BS, What is Business Central Extensions? (n.d.)., obtido a 20 fevereiro 2022 de https://www.activebs.com/en/news/2019/what-is-business-central-extensions

(Veldkamp, 2020) Veldkamp, M. (2020, December 7). Meer Dan 10.000 bedrijven werken met Microsoft Dynamics 365 Business Central Cloud. SucceedIT. Retrieved October 8, 2022, from https://www.succeedit.nl/meer-dan-10-000-bedrijven-werken-met-microsoft-dynamics-365-business-central-cloud/

(Microsoft, 2017) Microsoft. (19 de Julho de 2017). Development Environment (C/SIDE). Obtido de MSDN: https://msdn.microsoft.com/en-us/library/hh165383(v=nav.70).aspx

(Amazon, 2022) Amazon. What is Business Central Extensions? (n.d.)., obtido a 20 Fevereiro 2022 https://www.activebs.com/en/news/2019/what-is-business-central-extensions

(Chcomley, 2022) Chcomley. (n.d.). What is azure devops? - azure DevOps. Azure DevOps | Microsoft Docs. Obtido a 20 fevereiro, 2022, de https://docs.microsoft.com/en-us/azure/devops/user-guide/what-is-azure-devops?view=azure-devops

(Megel, 2019) Megel, M. (2019, September 3). CI / CD for Dynamics 365 Business Central - Part I. LinkedIn. Retrieved March 5, 2022, from https://www.linkedin.com/pulse/ci-cd-dynamics-365-business-central-part-i-michael-megel/

(Megel, 2019b) Megel, M. (2019, September 9). CI / CD for Dynamics 365 Business Central - Part II. LinkedIn. Retrieved March 5, 2022, from https://www.linkedin.com/pulse/ci-cd-dynamics-365-business-central-part-ii-michael-megel/

(Megel, 2019c) Megel, M. (2019, September 22). CI / CD for Dynamics 365 Business Central - Part III. LinkedIn. Retrieved March 5, 2022, from https://www.linkedin.com/pulse/ci-cd-dynamics-365-business-central-part-iii-michael-megel/

(Megel, 2019d) Megel, M. (2019, October 8). CI / CD for Dynamics 365 Business Central - Part IV. LinkedIn. Retrieved March 5, 2022, from https://www.linkedin.com/pulse/ci-cd-dynamics-365-business-central-part-iv-michael-megel/

(Red Hat, 2018) Red Hat (2018), What is CI/CD? Obtido a 20 de Fevereiro de 2020 de https://www.redhat.com/en/topics/devops/what-is-ci-cd

(Gillis, A. S., 2022) Gillis, A. S. (2022, February 1). What is an artifact in software development? SearchSoftwareQuality. Retrieved February 20, 2022, from https://searchsoftwarequality.techtarget.com/definition/artifact-software-development

(Techopedia, 2015) Techopedia. (2015, September 15). What is merge? - definition from Techopedia. Techopedia.com. Retrieved February 20, 2022, from https://www.techopedia.com/definition/1217/merge

(Schiestl, B., 2022) Schiestl, B. (n.d.). Code branching definition - what is a branch? Perforce Software. Retrieved February 20, 2022, from https://www.perforce.com/blog/vcs/branching-definition-what-branch

(Merron, D., 2020) Merron, D. (2020, May 13). Deployment Pipelines (CI/CD) in software engineering. BMC Blogs. Retrieved February 20, 2022, from https://www.bmc.com/blogs/deployment-pipeline/

(Techopedia, 2011) Techopedia. (2011, September 6). What is build? - definition from Techopedia. Techopedia.com. Retrieved February 20, 2022, from https://www.techopedia.com/definition/3759/build

(Dvernytskyi, 2020) Dvernytskyi, V. (2020). Visual studio code - useful extensions for Al Development. Visual Studio Code - Useful extensions for AL development. Retrieved February 26, 2022, from https://vld-nav.com/useful-extensions-for-al-development

(Koen et al., 2001) Koen P, Ajamian G, Burkart R, Clamen A, Davidson J, D'Amore R, Elkins C, Herald K, Incorvia M, Johnson A, Karol R, Seibert R, Slavejkov A, Wagner K (2001). Providing Clarity and a Common Language to the 'Fuzzy Front End'. Research Technology Management; 44, 2: 46-55.

(Woodall, 2003) Woodall, T., 2003. Conceptualising “Value for the Customer”: An Attributional, Structural and
Dispositional Analysis.

(Zeithmal, 1988) Zeithmal, V., 1988. Consumer Perceptions of Price, Quality, and Value: A Means-End Model and Synthesis of Evidence.

(ProductPlan, 2021) Business model canvas. ProductPlan. (2021, January 7). Retrieved February 27, 2022, from https://www.productplan.com/glossary/business-model-canvas/

(Saaty, 1991) Saaty, T., 1991. The Analytic Hierarchy Process.

(Microsoft, 2022) Sdwheeler. (n.d.). What is PowerShell? - powershell. PowerShell | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/powershell/scripting/overview?view=powershell-7.2

(Microsoft, 2022b) Sdwheeler. (n.d.). About modules - powershell. about Modules - PowerShell | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/powershell/module/microsoft.powershell.core/about/about_modules?view=powershell-7.2

(Microsoft, 2022c) KevSch. (n.d.). Use Docker in your business central development process - training. Training | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/learn/modules/use-docker-business-central/

(Microsoft, 2022d) KevSch. (n.d.). Work with Docker images and containers - training. Training | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/learn/modules/use-docker-business-central/2-docker-images-containers

(Microsoft, 2022e) Jswymer. (n.d.). Multitenant Deployment Architecture - Business Central. Multitenant Deployment Architecture - Business Central | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/deployment/multitenant-deployment-architecture

(Microsoft, 2022f) SusanneWindfeldPedersen. (n.d.). Developing extensions in Al - Business Central. Developing Extensions in AL - Business Central | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/developer/devenv-dev-overview

(Microsoft, 2022g) Tpetchel. (n.d.). What is github? - training. Training | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/learn/modules/introduction-to-github/2-what-is-github

(Microsoft, 2022h) Henrikwh. (n.d.). Using service to service authentication - business central. Using Service to Service Authentication - Business Central | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/administration/automation-apis-using-s2s-authentication

(Microsoft, 2022i) Jswymer. (n.d.). Using OAuth to authenticate business central web services (OData and soap) - business central. Using OAuth to Authenticate Business Central Web Services (OData and SOAP) - Business Central | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/dynamics365/business-central/dev-itpro/webservices/authenticate-web-services-using-oauth

(Microsoft, 2022j) Tamram. (n.d.). Introduction to blob (object) storage - azure storage. Introduction to Blob (object) storage - Azure Storage | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction

(Microsoft,2022k) Tamram. (n.d.). Create a service SAS - Azure Storage. Azure Storage | Microsoft Learn. Retrieved October 5, 2022, from https://docs.microsoft.com/en-us/rest/api/storageservices/create-service-sas#permissions-for-a-directory-container-or-blob

(Freddy Kristiansen, 2020) Kristiansen, F. (2020, October 8). Run-alpipeline. Freddys blog. Retrieved September 5, 2022, from https://freddysblog.com/2020/10/08/run-alpipeline/

(Freddy Kristiansen, 2022) Kristiansen, F. (2022, April 27). Al-go for github. Freddys blog. Retrieved September 6, 2022, from https://freddysblog.com/2022/04/26/al-go-for-github/

(NavisionPlanet,2022) Dynamics nav PowerShell. Navision Planet. (2020, July 28). Retrieved August 20, 2022, from https://www.navisionplanet.com/dynamics-nav-powershell/

(Maureen O'Gara, 2013) O'Gara, M. (2013, July 26). Maureen O'Gara. Ben Golub, Who Sold Gluster to Red Hat, Now Running dotCloud | Maureen O'Gara. Retrieved September 1, 2022, from https://web.archive.org/web/20190913100835/http://maureenogara.sys-con.com/node/2747331

(Jurgen et al., 2016) Cito, Jürgen & Ferme, Vincenzo & Gall, Harald. (2016). Using Docker Containers to Improve Reproducibility in Software and Web Engineering Research. 609-612. 10.1007/978-3-319-38791-8_58.

(Scolati et al., 2019) Scolati, Remo & Fronza, Ilenia & El Ioini, Nabil & Samir, Areeg & Pahl, Claus. (2019). A Containerized Big Data Streaming Architecture for Edge Cloud Computing on Clustered Single-Board Devices. 10.5220/0007695000680080.

(Waldo, 2021) Waldo. (n.d.). Al extension pack - visual studio marketplace. Retrieved June 11, 2022, from https://marketplace.visualstudio.com/items?itemName=waldo.al-extension-pack

(Philips et al., 2011) Shaun Phillips, Jonathan Sillito, and Rob Walker. 2011. Branching and merging: an investigation into current version control practices. In Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE '11). Association for Computing Machinery, New York, NY, USA, 9–15. https://doi.org/10.1145/1984642.1984645

(Loeliger et al., 2012)	Loeliger, J., & McCullough, M. (2012). Version control with git: Powerful tools and techniques for collaborative software development. O'Reilly.
(Knóldus, 2021)	Mansi. (2021, May 31). Introduction to git flow. Knoldus Blogs. Retrieved July 5, 2022, from https://blog.knoldus.com/introduction-to-git-flow/
(Hanna, 2022)	Hanna, K. T. (2022, March 17). What is a software release? SearchSoftwareQuality. Retrieved August 18, 2022, from https://www.techtarget.com/searchsoftwarequality/definition/release
(Plutora, 2022)	What is a software release? (all that you need to know). Plutora. (2022, March 1). Retrieved August 5, 2022, from https://www.plutora.com/software-release-management/software-release
(Babbar, 2021)	Babbar, M. (2021, June 20). Introduction to git flow. Medium. Retrieved August 17, 2022, from https://levelup.gitconnected.com/introduction-to-git-flow-3ad331d097fa
(Haddad, 2022)	Haddad, R. (2022, March 8). Git branching strategies: Gitflow, Github Flow, trunk based... Flagship.io. Retrieved October 5, 2022, from https://www.flagship.io/git-branching-strategies/
(GitLab, 2022)	Introduction to gitlab flow. GitLab. (n.d.). Retrieved October 5, 2022, from https://docs.gitlab.cn/14.0/ee/topics/gitlab_flow.html
(Sharma, 2020)	Sharma, V. (2020, September 16). Comparing git branching strategies. DEV Community. Retrieved October 5, 2022, from https://dev.to/arbitrarybytes/comparing-git-branching-strategies-dl4
(Hodor NV, 2019)	DevOps for Al Made Easy. ALOps. (n.d.). Retrieved October 5, 2022, from https://www.alops.be/#about
(Lenarduzzi et al., 2020)	Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2020, October 14). *A systematic literature review on technical debt prioritization: Strategies, processes, factors, and Tools*. Journal of Systems and Software. Retrieved September 17, 2022, from https://www.sciencedirect.com/science/article/pii/S016412122030220X
(Avgeriou et al., 2016)	Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016, October 12). *Managing technical debt in software engineering (dagstuhl seminar 16162)*. Dagstuhl Reports. Retrieved September 18, 2022, from https://drops.dagstuhl.de/opus/volltexte/2016/6693/
(Eick et al,. 2002)	S. Eick, T. Graves, A. Karr, J. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. Software Engineering, IEEE Transactions on, 27(1):1–12, 2002.
(Nugroho et al, 2011)	Nugroho, A., Visser, J., & Kuipers, T. (2011, May). *An empirical model of technical debt and interest*. Retrieved September 17, 2022, from https://www.researchgate.net/publication/228684782_An_Empirical_Model_of_Technical_Debt_and_Interest
(Lin et al, 1999)	Lin, Z.-C., & Yang, C.-B. (1999, February 16). *Evaluation of Machine Selection by the AHP Method*. Journal of Materials Processing Technology. Retrieved September 18, 2022, from https://www.sciencedirect.com/science/article/pii/0924013695020764
(Anonymous, 2022)	Analytic Hierarchy Process (AHP) tool. https://comcastsamples.github.io/ahp-tool/