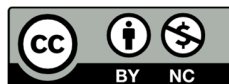


Tutorial de Introdução ao Azure DevOps

Da Configuração Inicial à Automação do Ciclo de Desenvolvimento



Luís Simões da Cunha (2025)



Índice

Capítulo 1 – Introdução ao Azure DevOps	6
Da Ideia ao Software Funcional: Como o Azure DevOps Facilita o Processo	6
1.1 O que é o Azure DevOps?	6
1.2 Benefícios do Azure DevOps para equipas de desenvolvimento	6
1.3 Principais componentes do Azure DevOps	7
1.4 Conceitos básicos de DevOps	8
Resumo do Capítulo 1	8
O que vem a seguir?	9
Capítulo 2 – Configuração Inicial e Primeiros Passos	10
Criando a Conta e Explorando o Ambiente do Azure DevOps	10
2.1 Criar uma Conta no Azure DevOps	10
Passo a passo para criar a conta:	10
2.2 Criar um Novo Projeto e Configurar Permissões Básicas	11
Criar um novo projeto	11
Configurar permissões básicas	11
2.3 Visão Geral do Ambiente do Azure DevOps	12
2.4 Interface Principal e Navegação	12
1 Barra superior	12
2 Menu lateral (Esquerda)	13
Menu "Repos" (Repositórios de código)	13
Menu "Boards" (Gestão de tarefas)	13
Resumo do Capítulo 2	13
Capítulo 3 – Fundamentos de Controle de Versões com Git (para quem nunca usou Git)	15
Gerindo Código de Forma Inteligente no Azure DevOps	15
3.1 O que é Git e por que usá-lo?	15
3.2 Conceitos básicos do Git: Repositórios, Commits, Branches e Merges	16
Repositório (Repo)	16
Commit	16
Branch (Ramificação)	16
Merge	16
3.3 Configuração inicial do Git no Azure Repos	17

3.4 Clonar um Repositório do Azure DevOps para o Computador.....	18
3.5 Criar, Modificar e Enviar Código para o Repositório.....	18
3.6 Resolvendo Conflitos de Merge Básicos	19
Como resolver conflitos?	19
Resumo do Capítulo 3	20
Capítulo 4 – Gerindo o Desenvolvimento com Azure Repos	21
Organizando e Colaborando no Código de Forma Eficiente	21
4.1 Trabalhando com Branches no Azure DevOps	21
✦ O que são Branches?	21
✦ Criando uma Branch no Azure DevOps	21
4.2 Estratégias de Branching: Feature Branches e GitFlow.....	22
✦ 1 Feature Branches (Mais Simples)	22
✦ 2 GitFlow (Mais Completo).....	23
4.3 Criando e Revisando Pull Requests.....	23
✦ Criando um Pull Request no Azure DevOps.....	23
✦ Revisando um Pull Request	24
4.4 Implementação de Políticas de Branch	24
✦ Como Configurar Políticas no Azure DevOps?	24
Resumo do Capítulo 4	25
Capítulo 5 – Gestão de Projetos Ágeis com Azure Boards	26
Organizando o Trabalho e Acompanhando o Progresso de Forma Eficiente.....	26
5.1 Criando e Gerindo Backlogs e Sprints	26
✦ O que é um Backlog?	26
✦ O que é um Sprint?	27
Criando um Sprint no Azure Boards.....	27
5.2 Trabalhando com Work Items: Epics, Features e User Stories	28
Criando Work Items no Azure Boards	28
5.3 Criando Boards e Personalizando Fluxos de Trabalho	29
Criando um Board no Azure DevOps.....	29
Personalizando Fluxos de Trabalho.....	29
5.4 Utilizando Queries para Rastrear Progresso	30
Resumo do Capítulo 5	30
Capítulo 6 – Introdução a CI/CD com Azure Pipelines	31
Automatizando Builds e Deploys para um Desenvolvimento Eficiente	31

6.1 O que é CI/CD e por que é importante?	31
6.2 Criando o Primeiro Pipeline de Build	32
Criando um Pipeline no Azure DevOps	32
✦ Exemplo de um pipeline básico para um projeto .NET:	33
6.3 Executando Testes Automáticos no Pipeline	33
✦ Exemplo de Pipeline com Testes (.NET Core)	33
6.4 Criando um Pipeline de Release para Deploy Automatizado	34
Criando um Pipeline de Release no Azure DevOps	34
✦ Exemplo de Deploy para Azure App Service (Web App)	34
Resumo do Capítulo 6	35
Capítulo 7 – Testes e Qualidade com Azure Test Plans (Introdução Básica)	36
Garantindo Qualidade com Testes Manuais e Automatizados	36
7.1 Criando Planos de Teste Manuais	36
✦ O que são Test Plans?	36
Criando um Plano de Teste no Azure DevOps	36
✦ Criando um Test Case (Caso de Teste)	37
7.2 Automatizando Testes dentro do Azure Pipelines	37
✦ Tipos de Testes Automatizados	38
Criando Testes Automatizados no Azure Pipelines	38
Resumo do Capítulo 7	40
Capítulo 9 – Melhores Práticas e Próximos Passos	41
Elevando a Eficiência no Uso do Azure DevOps	41
9.1 Boas Práticas para Equipas que Usam Azure DevOps	41
✦ Gestão de Código e Git	41
✦ Gestão de Tarefas com Azure Boards	42
✦ Pipelines e Automação	42
✦ Segurança e Permissões	42
9.2 Recursos para Aprofundamento e Documentação Oficial	42
✦ Documentação Oficial	43
✦ Cursos Online e Vídeos	43
9.3 Próximos Desafios: Infraestrutura como Código, Monitorização e Segurança	43
✦ Infraestrutura como Código (IaC)	43
✦ Monitorização e Logs	44
✦ Segurança no Azure DevOps	44

Resumo do Capítulo 9	45
Conclusão do Tutorial 🚀	45

Capítulo 1 – Introdução ao Azure DevOps

Da Ideia ao Software Funcional: Como o Azure DevOps Facilita o Processo

1.1 O que é o Azure DevOps?

Imagine que uma equipa de desenvolvimento de software está a trabalhar num novo projeto. Para que tudo funcione bem, eles precisam de:

- **Um local para armazenar o código-fonte**, garantindo que todos os programadores têm a versão mais atualizada do projeto.
- **Ferramentas para organizar tarefas**, como definir quais funcionalidades serão desenvolvidas e acompanhar o progresso.
- **Mecanismos para testar e publicar o software automaticamente**, sem precisar de instalar manualmente em cada servidor.

O **Azure DevOps** é uma plataforma da Microsoft que **centraliza todas essas funcionalidades num único lugar**. Ele é como uma "**fábrica digital**" onde as equipas podem planear, desenvolver, testar e lançar software de forma eficiente.

1.2 Benefícios do Azure DevOps para equipas de desenvolvimento

A razão pela qual as empresas e programadores usam o Azure DevOps é simples: **ele ajuda a entregar software de forma mais rápida, organizada e confiável**. Eis alguns benefícios:

- ✅ **Colaboração eficiente** – Permite que todos os membros da equipa trabalhem juntos no mesmo projeto, sem perder versões antigas do código.
- ✅ **Automação** – Processos repetitivos, como testes e publicações, podem ser automatizados.
- ✅ **Flexibilidade** – Funciona com diferentes linguagens de programação e tipos de projetos.
- ✅ **Rastreamento e organização** – Ferramentas para acompanhar tarefas e garantir que nada fica esquecido.
- ✅ **Integração com Git** – Ajuda a gerir o código-fonte e diferentes versões do software.

Exemplo prático:

Imagina que uma equipa está a criar uma aplicação web. O Azure DevOps permite que:

- O código esteja sempre disponível para todos os programadores.
- O progresso seja acompanhado num quadro visual.
- Cada vez que alguém faz uma alteração no código, ele seja testado automaticamente.
- Quando a aplicação estiver pronta, seja publicada automaticamente sem complicações.

1.3 Principais componentes do Azure DevOps

O Azure DevOps é composto por **cinco ferramentas principais**, cada uma com uma função específica.

- ◆ **Azure Repos** – Gestão do código-fonte com Git.
- ◆ **Azure Pipelines** – Automação de testes e deploy (CI/CD).
- ◆ **Azure Boards** – Gestão ágil de tarefas e projetos.
- ◆ **Azure Test Plans** – Planeamento e execução de testes.
- ◆ **Azure Artifacts** – Gestão de dependências e pacotes.

Analogia simples:

Pensa no Azure DevOps como uma **grande caixa de ferramentas** para programadores. Cada ferramenta tem um propósito, mas todas funcionam juntas.

Ferramenta	O que faz?	Exemplo prático
Azure Repos	Armazena e controla versões do código.	Um programador faz alterações no código sem perder a versão anterior.
Azure Pipelines	Automatiza testes e publicação do software.	Sempre que há uma alteração no código, os testes são executados automaticamente.
Azure Boards	Ajuda a organizar tarefas e acompanhar progresso.	A equipa define o que deve ser feito nesta semana e acompanha o andamento.
Azure Test Plans	Criação e gestão de testes manuais e automáticos.	Um gestor quer garantir que todas as funcionalidades foram testadas antes de publicar a aplicação.
Azure Artifacts	Gestão de bibliotecas e dependências.	Uma equipa usa pacotes de código reutilizáveis para não precisar programar tudo do zero.

1.4 Conceitos básicos de DevOps

O nome "Azure DevOps" vem da junção de duas palavras: **Desenvolvimento (Dev) + Operações (Ops)**.

DevOps é uma **cultura de trabalho** que une programadores e administradores de sistemas para que o software seja **desenvolvido, testado e publicado rapidamente, sem erros e sem atrasos**.

Antes do DevOps:

- O programador escrevia o código e passava para outra equipa testar.
- A equipa de operações só publicava o software depois de muito tempo, criando atrasos.
- Pequenos erros atrasavam todo o processo.

Com DevOps:

- O código é testado e publicado automaticamente.
- Pequenas alterações são feitas e testadas sem grandes complicações.
- O software chega aos utilizadores muito mais rápido e com menos erros.

Exemplo do mundo real:

Imagina um **banco** que está a criar uma nova funcionalidade para a sua app. Se não usar DevOps, a equipa pode levar **semanas ou meses** até lançar a atualização. Com DevOps, o código pode ser escrito, testado e publicado **em poucos dias**, garantindo que os clientes recebam as novidades rapidamente.

Resumo do Capítulo 1

- O **Azure DevOps** é uma plataforma que centraliza ferramentas essenciais para desenvolvimento de software.
- Ele ajuda as equipas a **colaborar, automatizar e entregar software com qualidade**.
- Suas principais ferramentas incluem:
 - **Azure Repos** (código-fonte),
 - **Azure Pipelines** (automação),
 - **Azure Boards** (gestão de projetos),
 - **Azure Test Plans** (testes) e
 - **Azure Artifacts** (gestão de pacotes).

- O conceito de **DevOps** permite que as equipas desenvolvam e publiquem software de forma mais rápida e confiável.
-

O que vem a seguir?

Agora que já compreendemos o que é o Azure DevOps e por que ele é útil, o **Capítulo 2** ensinará **como criar uma conta no Azure DevOps e configurar o primeiro projeto**. Vamos colocar a teoria em prática! 🚀

Capítulo 2 – Configuração Inicial e Primeiros Passos

Criando a Conta e Explorando o Ambiente do Azure DevOps

Agora que já compreendes o que é o **Azure DevOps** e para que serve, está na hora de colocares as mãos na massa! 🚀

Neste capítulo, vamos:

- ✓ Criar uma conta no Azure DevOps.
- ✓ Criar o primeiro projeto.
- ✓ Explorar a interface e os principais menus.

Se nunca usaste ferramentas da Microsoft antes, não te preocupes! Vamos guiar-te passo a passo.

2.1 Criar uma Conta no Azure DevOps

Para começares, precisas de uma **conta da Microsoft**. Se já tens uma (como um e-mail do Outlook, Hotmail ou uma conta do Office 365), podes usá-la. Caso contrário, poderás criar uma nova gratuitamente.

Passo a passo para criar a conta:

1. Abre um navegador e vai para <https://azure.microsoft.com/pt-pt/products/devops/>
2. Clica no botão "**Iniciar gratuitamente**".
3. Escolhe "**Começar gratuitamente**" na opção **Azure DevOps Services**.
4. Se já tiveres uma conta Microsoft, faz login. Se não, clica em "**Criar conta**" e segue as instruções para criar uma nova.

👉 **Dica:** O plano gratuito do Azure DevOps já oferece o suficiente para projetos individuais e pequenas equipas!

Após o login, serás direcionado para a página principal do Azure DevOps. Agora, vamos criar um projeto!

2.2 Criar um Novo Projeto e Configurar Permissões Básicas

Um **projeto** no Azure DevOps é como uma **pasta organizadora** para tudo o que estiver relacionado ao teu software: código, tarefas, testes, automação, etc.

Criar um novo projeto

1. Depois de fazer login, clica em "**Criar novo projeto**".
2. Escolhe um nome para o projeto, por exemplo, "**Meu Primeiro Projeto**".
3. Escolhe uma descrição (opcional).
4. No campo "**Visibilidade**", podes escolher entre:
 - **Privado** (só tu e a tua equipa podem ver).
 - **Público** (qualquer pessoa pode ver, útil para projetos open-source).
5. No campo "**Controle de Versão**", escolhe **Git**.
6. Em "**Processo**", deixa a opção padrão "**Basic**" (mais fácil para quem está a começar).
7. Clica em "**Criar**".

Configurar permissões básicas

Se estiveres a trabalhar sozinho, não precisas de configurar permissões. Mas se quiseres adicionar colegas à equipa:

1. No menu lateral esquerdo, clica em "**Project Settings**" (**Configurações do Projeto**).
2. Em "**Permissions**" (**Permissões**), clica em "**Users**" (**Utilizadores**).
3. Adiciona os e-mails dos teus colegas e define as permissões:
 - **Leitor**: pode ver o código, mas não alterar nada.
 - **Colaborador**: pode contribuir com código.
 - **Administrador**: pode alterar configurações do projeto.

👉 **Dica:** Para projetos pequenos, podes simplesmente adicionar os teus colegas como **Colaboradores**.

Agora que o projeto está criado, vamos explorar o ambiente!

2.3 Visão Geral do Ambiente do Azure DevOps

A interface do Azure DevOps pode parecer complexa no início, mas na verdade segue uma lógica muito clara. Assim que entras no teu projeto, verás um menu lateral com as principais secções:

Secção	O que é?	Para que serve?
Overview	Visão geral do projeto	Ver o status e informações gerais.
Boards	Gestão de tarefas	Criar e organizar o trabalho da equipa.
Repos	Repositórios Git	Guardar e gerir o código-fonte.
Pipelines	Automação CI/CD	Criar processos automáticos de build e deploy.
Test Plans	Testes de qualidade	Criar e executar testes (manual e automático).
Artifacts	Gestão de pacotes	Guardar bibliotecas e dependências.

👉 **Dica:** Se estiveres a começar, os menus mais importantes para ti são **Repos** (para código) e **Boards** (para organização do trabalho).

2.4 Interface Principal e Navegação

Agora que já conheces os menus, vamos ver **como navegar dentro do Azure DevOps**.

1 Barra superior

Na parte superior da página, encontras três elementos importantes:

- **Organização:** Aqui podes alternar entre diferentes organizações (caso tenhas mais de uma).
- **Projetos:** Lista os projetos disponíveis.
- **Pesquisa** 🔍: Para encontrar rapidamente repositórios, tarefas ou membros da equipa.

2 Menu lateral (Esquerda)

Este menu muda conforme a secção que estiveres a usar. Vamos ver os dois mais importantes para quem está a começar:

Menu "Repos" (Repositórios de código)

Aqui vais encontrar as opções para gerir o teu código-fonte.

- **Files:** Mostra os ficheiros do projeto.
- **Commits:** Regista todas as alterações feitas no código.
- **Branches:** Gerencia diferentes versões do código.
- **Pull Requests:** Onde podes sugerir alterações ao código e pedir revisões.

Menu "Boards" (Gestão de tarefas)

Aqui podes gerir o trabalho de forma organizada.

- **Work Items:** Lista de tarefas atribuídas à equipa.
- **Boards:** Quadro visual estilo *Kanban* para gerir tarefas.
- **Sprints:** Planeamento semanal ou quinzenal das tarefas.

Exemplo prático:

Imagina que tens um projeto para criar uma aplicação. No **Boards**, adicionas tarefas como "**Criar página de login**", "**Desenvolver API**", etc. Depois, no **Repos**, adicionas e geres o código dessas funcionalidades.

Resumo do Capítulo 2

- Criámos uma conta no **Azure DevOps**.
- Criámos um **projeto** e configurámos permissões básicas.
- Explorámos a **interface do Azure DevOps**, com foco em **Repos (código-fonte)** e **Boards (gestão de tarefas)**.
- Aprendemos como navegar entre os menus principais.

Agora que já temos o ambiente configurado, no **próximo capítulo** vamos aprender **os fundamentos do Git**, para que possas começar a trabalhar com código no Azure DevOps sem complicações! 🚀

Capítulo 3 – Fundamentos de Controle de Versões com Git

(para quem nunca usou Git)

Gerindo Código de Forma Inteligente no Azure DevOps

Agora que já criaste o teu projeto no Azure DevOps, chegou a hora de aprender **Git**, a ferramenta mais utilizada para controlo de versões.

Se nunca usaste Git antes, não te preocupes! Este capítulo foi feito para ti, com explicações simples e exemplos práticos.

3.1 O que é Git e por que usá-lo?

Git é um **sistema de controle de versões distribuído**. Mas o que isso significa na prática?

Imagina que estás a escrever um documento no Word. Se quiseses guardar versões diferentes (por exemplo, uma versão inicial, outra com revisões e outra final), provavelmente vais criar ficheiros como:

- documento_v1.docx
- documento_final_corrigido.docx
- documento_final_corrigido_revisado.docx

Agora imagina que estás a trabalhar num projeto de software com centenas de ficheiros. Criar cópias para cada alteração seria um caos!

Aqui entra o **Git**, que permite:

- ✓ Guardar automaticamente todas as versões do código sem criar ficheiros duplicados.
- ✓ Reverter para qualquer versão anterior quando necessário.
- ✓ Trabalhar em equipa sem que um programador sobrescreva o código do outro.
- ✓ Criar "ramificações" (*branches*) do código para desenvolver novas funcionalidades sem mexer no código principal.

♦ Exemplo do mundo real:

Pensa num livro que está a ser escrito por várias pessoas. Cada autor pode trabalhar num capítulo (numa *branch*

diferente), e depois juntar tudo no documento final (*merge*). Se um erro for encontrado, podes recuperar uma versão anterior sem perder tudo!

3.2 Conceitos básicos do Git: Repositórios, Commits, Branches e Merges

Agora que já entendes a utilidade do Git, vamos aos conceitos essenciais.

Repositório (Repo)

Um **repositório** é como uma pasta que guarda todos os ficheiros e as suas versões.

- ♦ No **Azure DevOps**, cada projeto tem um repositório Git associado no **Azure Repos**.

Commit

Um **commit** é uma "fotografia" do código num determinado momento.

- Sempre que fazes uma alteração no código, podes criar um commit para guardar essa versão.
- Cada commit tem uma **mensagem descritiva** que explica a alteração feita.

♦ Exemplo:

Se estiveres a corrigir um bug, podes fazer um commit com a mensagem:

✚ Corrigido erro no formulário de login

Branch (Ramificação)

Uma **branch** é uma cópia do código onde podes fazer alterações sem afetar o código principal.

♦ Exemplo:

Se quiseres desenvolver uma nova funcionalidade chamada "Exportação de Relatórios", podes criar uma branch chamada `feature-exportacao-relatorios` e só juntar essa funcionalidade ao código principal quando estiver pronta.

Merge

O **merge** é o processo de juntar uma branch ao código principal.

- Quando terminas de desenvolver uma funcionalidade numa branch, fazes um merge para integrar essas alterações.
- Se duas pessoas fizeram alterações no mesmo ficheiro, pode haver um **conflito** que precisa ser resolvido.

♦ **Analogia:**

Pensa numa receita de bolo.

- O código principal é a receita original.
 - Criar uma branch significa experimentar um novo ingrediente.
 - Se gostares do resultado, fazes um merge e adicionas a nova versão à receita original.
-

3.3 Configuração inicial do Git no Azure Repos

Agora que já compreendes os conceitos, vamos usar o Git no **Azure DevOps**.

1 Abrir o Azure Repos

1. Acede ao teu projeto no Azure DevOps.
2. No menu esquerdo, clica em **Repos** → **Files**.

2 Configurar Git na tua máquina

Se ainda não tens o Git instalado no computador, precisas de o instalar:

- ♦ **Windows:** [Baixar Git para Windows](#)
- ♦ **Mac:** [Baixar Git para Mac](#)
- ♦ **Linux:** Usa o comando `sudo apt install git` (Ubuntu) ou `sudo dnf install git` (Fedora).

Depois de instalar, abre o **terminal (cmd/powershell)** e executa:

```
git --version
```

Se aparecer algo como `git version 2.x.x`, significa que o Git está instalado corretamente.

Agora, configura o teu nome e e-mail (necessário para identificar os teus commits):

```
git config --global user.name "O Teu Nome"
git config --global user.email "teu@email.com"
```

3.4 Clonar um Repositório do Azure DevOps para o Computador

Agora vamos **clonar** (baixar uma cópia) do repositório para o teu computador.

- 1 No Azure DevOps, vai a **Repos** → **Files**.
- 2 Clica no botão **Clone** e copia o link HTTPS.
- 3 No terminal, navega até a pasta onde queres guardar o código e executa:

```
git clone URL_DO_REPO
```

(Substitui URL_DO_REPO pelo link copiado).

Isto criará uma pasta com todos os ficheiros do projeto.

♦ **Exemplo:**

Se o repositório se chama meu-projeto, será criada uma pasta meu-projeto no teu computador.

3.5 Criar, Modificar e Enviar Código para o Repositório

Agora que tens o repositório no teu computador, podes começar a trabalhar nele!

- 1 **Criar uma nova branch** (para evitar mexer diretamente na principal):

```
git checkout -b minha-nova-branch
```

- 2 **Criar ou modificar um ficheiro**

Abre qualquer ficheiro no editor de código e faz uma alteração.

- 3 **Adicionar as alterações ao Git**

```
git add .
```

- 4 **Criar um commit com uma mensagem descritiva**

```
git commit -m "Adicionei nova funcionalidade X"
```

5 Enviar as alterações para o repositório do Azure DevOps

```
git push origin minha-nova-branch
```

Agora, podes ver a nova branch no **Azure Repos**.

3.6 Resolvendo Conflitos de Merge Básicos

Às vezes, duas pessoas alteram o mesmo ficheiro e o Git não sabe qual versão manter. Isso gera um **conflito de merge**.

♦ Exemplo de conflito:

Se a branch `feature-x` e a `main` alteram o mesmo código, o Git vai pedir que escolhas qual versão manter.

Como resolver conflitos?

1. Faz o merge da `main` para a tua branch:

```
git pull origin main
```

2. Se houver um conflito, o Git marca as diferenças assim:

```
<<<<<<< HEAD
Código na main
=====
Código na feature-x
>>>>>>> feature-x
```

3. Edita manualmente o ficheiro para manter a versão correta.
4. Depois, faz um novo commit:

```
git add .
git commit -m "Resolvi o conflito no ficheiro X"
git push origin minha-nova-branch
```

Agora podes fazer o merge normalmente!

Resumo do Capítulo 3

- ✓ O **Git** é essencial para controlar versões do código e trabalhar em equipa.
- ✓ Os principais conceitos são: **Repositório, Commit, Branch e Merge**.
- ✓ O **Azure Repos** é onde o código é armazenado no Azure DevOps.
- ✓ Aprendemos a **clonar, modificar e enviar código** para o repositório.
- ✓ Vimos como **resolver conflitos** quando duas pessoas editam o mesmo ficheiro.

No **próximo capítulo**, vamos aprender a **trabalhar com branches no Azure DevOps e criar Pull Requests!**



Capítulo 4 – Gerindo o Desenvolvimento com Azure Repos

Organizando e Colaborando no Código de Forma Eficiente

Agora que já sabes os fundamentos do Git e como usar o **Azure Repos**, chegou a hora de aprender a **gerir o fluxo de desenvolvimento** de forma profissional.

Neste capítulo, vais aprender a:

- ✓ Criar e gerir **branches** corretamente.
- ✓ Usar estratégias eficientes como **Feature Branches** e **GitFlow**.
- ✓ Criar e rever **Pull Requests**.
- ✓ Definir **políticas para proteger o código**.

Vamos lá! 🚀

4.1 Trabalhando com Branches no Azure DevOps

📌 O que são Branches?

Uma **branch** é uma linha de desenvolvimento separada do código principal. Permite que possas trabalhar numa nova funcionalidade sem afetar o código existente.

♦ Exemplo:

Imagina que a tua equipa precisa adicionar uma funcionalidade de exportação de relatórios. Em vez de alterar o código principal (**main**), podes criar uma nova branch chamada `feature-exportacao` e fazer as mudanças lá.

Quando a funcionalidade estiver pronta, essa branch pode ser **mesclada (merge) de volta** ao código principal.

📌 Criando uma Branch no Azure DevOps

- 1 Acede ao **Azure Repos** no teu projeto.
- 2 No menu lateral, clica em **Branches**.
- 3 No canto superior direito, clica em **New Branch**.

- 4 Escolhe um nome para a branch (exemplo: feature-nova-funcionalidade).
- 5 Define a branch de origem (**main** ou outra que fizer sentido).
- 6 Clica em **Create Branch**.

👉 Alternativa via Terminal (Git CLI):

Se preferires criar a branch no teu computador antes de enviá-la para o Azure DevOps, usa:

```
git checkout -b feature-nova-funcionalidade  
git push origin feature-nova-funcionalidade
```

Agora podes trabalhar na nova branch sem afetar o código principal!

4.2 Estratégias de Branching: Feature Branches e GitFlow

Para manter um fluxo de trabalho organizado, as equipas usam diferentes **estratégias de branching**. Vamos ver as duas mais populares.

🔑 1 Feature Branches (Mais Simples)

- Cada nova funcionalidade é desenvolvida numa **branch separada**.
- Quando a funcionalidade está pronta, a branch é **mesclada na main**.
- Depois, a branch pode ser eliminada.

♦ Exemplo:

1. Criar feature-login.
2. Desenvolver e testar o código.
3. Fazer um merge na main através de um Pull Request.
4. Apagar a branch feature-login após a integração.

👉 Ideal para projetos pequenos ou individuais.

2 GitFlow (Mais Completo)

O **GitFlow** é um fluxo mais estruturado que usa várias branches para gerir diferentes fases do desenvolvimento.

Branch	Para que serve?
main	Versão estável em produção.
develop	Código em desenvolvimento.
feature/*	Desenvolvimento de novas funcionalidades.
release/*	Preparação para um lançamento.
hotfix/*	Correção de bugs críticos na produção.

♦ Fluxo do GitFlow:

1. Criar uma branch feature-nova-funcionalidade a partir de develop.
2. Desenvolver e testar a funcionalidade.
3. Fazer merge em develop.
4. Criar uma release para preparar o lançamento.
5. Fazer merge na main e gerar uma versão estável.

👉 **Vantagens do GitFlow:** Melhor organização para equipas grandes.

👉 **Desvantagem:** Pode ser excessivo para projetos pequenos.

✳️ **Conclusão:** Se estás a começar, usa **Feature Branches**. Se o projeto crescer, podes adotar o **GitFlow**.

4.3 Criando e Revisando Pull Requests

Depois de terminares o desenvolvimento numa branch, precisas de **integrar o código na main**. Isso deve ser feito através de um **Pull Request (PR)** para garantir qualidade e revisão.


Criando um Pull Request no Azure DevOps

- 1 Vais a **Azure Repos** → **Pull Requests**.
- 2 Clica em **New Pull Request**.
- 3 Escolhe a branch de origem (exemplo: feature-login) e a branch de destino (main).

- 4 Escreve uma descrição clara das alterações feitas.
- 5 Podes atribuir revisores (outros membros da equipa).
- 6 Clica em **Create** para criar o Pull Request.

Revisando um Pull Request

- Os revisores podem **comentar**, **pedir alterações** ou **aprovar** o código.
- O autor do PR pode fazer mais commits para corrigir sugestões.
- Quando tudo estiver certo, o PR pode ser **mesclado na main**.

 **Dica:** Sempre revisa o código antes do merge para evitar erros em produção!

4.4 Implementação de Políticas de Branch

Para manter a qualidade do código, o Azure DevOps permite configurar **políticas de branch**.

♦ Exemplos de políticas úteis:

- ✓ **Exigir aprovação de Pull Requests** antes do merge.
- ✓ **Impedir commits diretos na main**, forçando o uso de PRs.
- ✓ **Executar testes automáticos antes de aprovar um merge**.

Como Configurar Políticas no Azure DevOps?

- 1 Vai a **Azure Repos** → **Branches**.
- 2 Passa o rato sobre a branch main e clica em ..." → **Branch Policies**.
- 3 Ativa as opções que quiseres, como:
 - ✓ **"Require a minimum number of reviewers"** (exigir revisores).
 - ✓ **"Check for linked work items"** (garantir que cada PR tem uma tarefa associada).
 - ✓ **"Check for comment resolution"** (exigir que todos os comentários sejam resolvidos antes do merge).

 **Dica:** Estas políticas ajudam a manter um código mais limpo e seguro.

Resumo do Capítulo 4

- ✓ Aprendemos a **criar e gerir branches no Azure DevOps**.
 - ✓ Vimos **estratégias de branching**, incluindo **Feature Branches e GitFlow**.
 - ✓ Criámos e revisámos **Pull Requests** para integrar código de forma segura.
 - ✓ Configurámos **políticas de branch** para manter a qualidade do código.
- ✦ **Próximo passo:** No **Capítulo 5**, vamos aprender a usar **Azure Boards** para gerir tarefas e organizar o trabalho da equipa! 🚀

Capítulo 5 – Gestão de Projetos Ágeis com Azure Boards

Organizando o Trabalho e Acompanhando o Progresso de Forma Eficiente

Agora que já sabes como gerir código no **Azure Repos**, chegou a hora de aprender a organizar as tarefas e acompanhar o progresso do desenvolvimento no **Azure Boards**.

Se já usaste **Scrum** ou **Kanban**, este capítulo será fácil de entender. Se não, não te preocupes! Vou explicar os conceitos básicos para que possas tirar o máximo proveito do Azure Boards.

Neste capítulo, vais aprender a:

- ✓ Criar e gerir **backlogs e sprints**.
- ✓ Organizar tarefas usando **Work Items** como **Epics, Features e User Stories**.
- ✓ Criar **Boards personalizados** para visualizar o trabalho.
- ✓ Usar **queries** para rastrear progresso e gerar relatórios.

Vamos lá! 🚀

5.1 Criando e Gerindo Backlogs e Sprints

O **Azure Boards** é uma ferramenta para organizar e acompanhar tarefas de desenvolvimento. Ele suporta **metodologias ágeis**, como **Scrum** e **Kanban**.

Mas antes de falarmos de metodologias, precisamos entender dois conceitos essenciais:

📌 O que é um Backlog?

O **backlog** é uma lista de tudo o que precisa ser feito no projeto. Funciona como uma **to-do list**, onde cada item representa uma funcionalidade, melhoria ou correção que precisa ser desenvolvida.

♦ Exemplo:

Se estivermos a criar uma aplicação de e-commerce, o backlog pode incluir:

- Criar página de login
- Implementar carrinho de compras
- Adicionar integração com pagamentos

No Azure Boards, o backlog é gerido dentro da secção **Boards** → **Backlogs**.

O que é um Sprint?

Um **Sprint** é um período fixo (geralmente 1 a 4 semanas) em que um conjunto de tarefas do backlog será concluído.

♦ Analogia:

Pensa no backlog como uma **prateleira cheia de tarefas** e no sprint como uma **cesta onde colocamos apenas as tarefas que vamos resolver agora**.

👉 No Azure DevOps, podes configurar sprints no menu:

📌 **Boards** → **Sprints** → **New Sprint**

Criando um Sprint no Azure Boards

- 1 Vai a **Boards** → **Sprints**.
- 2 Clica em **New Sprint**.
- 3 Define um **nome** e um **período (data de início e fim)**.
- 4 Adiciona tarefas do backlog a este sprint.

Agora a equipa sabe exatamente no que trabalhar nas próximas semanas!

5.2 Trabalhando com Work Items: Epics, Features e User Stories

No Azure Boards, as tarefas são organizadas em diferentes níveis de abstração, chamados de **Work Items**.

Work Item	Para que serve?	Exemplo
Epic	Representa um grande objetivo do projeto.	"Criar Plataforma de E-commerce"
Feature	Um conjunto de funcionalidades dentro de um Epic.	"Implementar Carrinho de Compras"
User Story	Uma funcionalidade específica que um utilizador precisa.	"Permitir que o utilizador adicione produtos ao carrinho"
Task	Uma tarefa específica a ser concluída dentro de uma User Story.	"Criar botão de adicionar ao carrinho"

♦ Analogia:

Pensa num **Epic** como um grande projeto, uma **Feature** como um capítulo desse projeto, uma **User Story** como uma secção desse capítulo e uma **Task** como um parágrafo dentro dessa secção.

Criando Work Items no Azure Boards

- 1 Vais a **Boards** → **Work Items**.
- 2 Clica em **New Work Item** e escolhe **Epic**, **Feature**, **User Story** ou **Task**.
- 3 Preenche os detalhes (nome, descrição, prioridade).
- 4 Associa a um Sprint ou Epic, se necessário.
- 5 Atribui a um membro da equipa.
- 6 Guarda e acompanha o progresso.

✦ **Dica:** Sempre liga **User Stories** às **Features** e **Features** aos **Epics** para manter a hierarquia organizada.



5.3 Criando Boards e Personalizando Fluxos de Trabalho

Agora que já temos tarefas no backlog, precisamos de um **quadro visual** para organizar o fluxo de trabalho.

O Azure Boards permite criar **Boards Kanban**, onde cada tarefa pode estar em diferentes estados, como:

- **To Do** (A Fazer)
- **In Progress** (Em Progresso)
- **Done** (Concluído)

♦ Exemplo de um Board Kanban:

 Tarefa	 Estado
Criar login	Em progresso
Desenvolver API	A fazer
Implementar checkout	Concluído

Criando um Board no Azure DevOps

- 1 Vai a **Boards** → **Boards**.
- 2 Escolhe um projeto e vê as tarefas organizadas.
- 3 Arrasta e solta tarefas entre as colunas para atualizar o progresso.

Personalizando Fluxos de Trabalho

Se quiseres mais controlo, podes personalizar os estados do board:

- 1 Vai a **Project Settings** → **Boards** → **Process**.
- 2 Seleciona um processo e clica em **Customize**.
- 3 Adiciona novos estados (exemplo: "Em Revisão", "Testes").
- 4 Guarda e volta ao board para ver as mudanças.

🚩 **Dica:** Adiciona **limites de WIP (Work In Progress)** para evitar que muitas tarefas fiquem "Em Progresso" ao mesmo tempo.

5.4 Utilizando Queries para Rastrear Progresso

À medida que o projeto cresce, torna-se essencial rastrear o progresso com **queries**.

O Azure Boards permite criar **consultas personalizadas** para responder perguntas como:

- **Quais tarefas estão bloqueadas?**
- **Quais histórias de utilizador ainda não foram concluídas?**
- **Quem tem mais tarefas atribuídas?**

♦ Criando uma Query no Azure Boards

- 1 Vai a **Boards** → **Queries** → **New Query**.
 - 2 Define filtros (exemplo: "State = In Progress").
 - 3 Clica em **Run Query** para ver os resultados.
 - 4 Guarda e partilha a query com a equipa.
-

Resumo do Capítulo 5

- ✓ Aprendemos a **criar e gerir backlogs e sprints** para organizar o trabalho.
- ✓ Entendemos os diferentes **Work Items** (Epics, Features, User Stories e Tasks).
- ✓ Criámos **Boards Kanban** para visualizar o progresso das tarefas.
- ✓ Usámos **queries** para rastrear o estado do projeto e gerar relatórios.

🚩 **Próximo passo:** No **Capítulo 6**, vamos mergulhar no **Azure Pipelines** e aprender como automatizar builds e deploys! 🚀

Capítulo 6 – Introdução a CI/CD com Azure Pipelines

Automatizando Builds e Deploys para um Desenvolvimento Eficiente

Agora que já organizaste o código com **Azure Repos** e geriste as tarefas com **Azure Boards**, chegou a hora de dar o próximo passo: **automatizar o processo de build, testes e deploy com Azure Pipelines!** 🚀

Neste capítulo, vais aprender a:

- ✓ Compreender os conceitos de **CI/CD** e por que são essenciais.
- ✓ Criar o **primeiro Pipeline de Build** para compilar e validar o código.
- ✓ Adicionar **testes automáticos** ao pipeline para garantir qualidade.
- ✓ Criar um **Pipeline de Release** para automatizar o deploy.

Vamos lá! 🚀

6.1 O que é CI/CD e por que é importante?

CI/CD são duas práticas fundamentais no desenvolvimento moderno:

- ◆ **CI (Continuous Integration – Integração Contínua)**
 - Cada vez que um programador faz uma alteração no código, o sistema compila e executa testes automaticamente.
 - Isso evita problemas ao integrar diferentes partes do software.
- ◆ **CD (Continuous Deployment/Delivery – Entrega/Deploy Contínuo)**
 - O código é automaticamente enviado para um ambiente de teste ou produção após ser validado.
 - Evita deploys manuais e erros humanos.

🚩 Benefícios do CI/CD:

- ✅ **Detecção precoce de erros** → Builds e testes automáticos ajudam a evitar bugs.
- ✅ **Agilidade no desenvolvimento** → Pequenas alterações podem ser enviadas rapidamente.
- ✅ **Menos trabalho manual** → O processo de deploy é automático e confiável.

💡 Exemplo prático:

Imagina que uma empresa lança atualizações para a sua aplicação toda semana.

Com CI/CD, sempre que um programador faz um commit, o código é testado e, se estiver tudo certo, enviado automaticamente para produção!

Agora vamos ver como implementar isso no **Azure Pipelines**!

6.2 Criando o Primeiro Pipeline de Build

O **Pipeline de Build** é responsável por:

- ✓ Compilar o código.
- ✓ Executar testes básicos.
- ✓ Gerar artefatos (ficheiros prontos para deploy).

Criando um Pipeline no Azure DevOps

- 1 No menu lateral, clica em **Pipelines** → **Pipelines**.
 - 2 Clica em **New Pipeline**.
 - 3 Escolhe onde o código está armazenado (exemplo: **Azure Repos Git**).
 - 4 Seleciona o repositório do projeto.
 - 5 Escolhe um template de build ou clica em **Starter Pipeline** (para começar do zero).
 - 6 Edita o ficheiro `azure-pipelines.yml` para definir as etapas da build.
-

🔗 Exemplo de um pipeline básico para um projeto .NET:

trigger:

- main # O pipeline será executado sempre que houver alterações na branch main

pool:

vmImage: 'windows-latest' # Máquina virtual que executará a build

steps:

- task: UseDotNet@2
 - inputs:
 - packageType: 'sdk'
 - version: '6.x' # Define a versão do .NET
- script: dotnet build --configuration Release
 - displayName: 'Compilando a aplicação'

Depois de configurar o pipeline, clica em **Save and Run** para testá-lo.

🔴 **Dica:** Se estiveres a usar outro ambiente (Node.js, Python, Java), o processo será semelhante, apenas mudando os comandos de build.

6.3 Executando Testes Automáticos no Pipeline

Agora que temos um pipeline de build, vamos adicionar **testes automáticos** para garantir que o código está a funcionar corretamente.

🔗 Exemplo de Pipeline com Testes (.NET Core)

steps:

- script: dotnet test --configuration Release
 - displayName: 'Executando Testes Automatizados'

🔴 **Se houver falhas nos testes, o pipeline falha e impede que código com bugs chegue à produção!**

♦ **Outros exemplos de testes em diferentes tecnologias:**

✓ **Node.js:** `npm test`

✓ **Python:** `pytest`

✓ **Java:** `mvn test`

Executa o pipeline novamente e verifica os resultados dos testes!

6.4 Criando um Pipeline de Release para Deploy Automatizado

Agora que o código já está a ser testado automaticamente, vamos criar um **Pipeline de Release** para automatizar o deploy.

✦ O **Pipeline de Release** pega os ficheiros gerados pelo Pipeline de Build e envia-os para produção.

Criando um Pipeline de Release no Azure DevOps

- 1 No menu lateral, clica em **Pipelines** → **Releases**.
 - 2 Clica em **New Pipeline**.
 - 3 Escolhe **Empty Job** para criar um pipeline personalizado.
 - 4 Clica em **Add an artifact** e selecciona o pipeline de build como origem.
 - 5 Adiciona um **Stage** (Exemplo: "Deploy para Teste").
 - 6 Dentro do stage, adiciona tarefas como:
 - Deploy para um **Servidor Web** (IIS, Azure App Service, etc.).
 - Publicação num **Docker Container**.
-

✦ Exemplo de Deploy para Azure App Service (Web App)

- 1 Adiciona a tarefa **Azure App Service Deploy**.
- 2 Escolhe a tua conta Azure e o serviço onde queres publicar.

3 Define o ficheiro ZIP gerado no **Pipeline de Build**.

4 Guarda e executa o pipeline! 🚀

♦ Agora, sempre que fizeres uma alteração no código, ele será automaticamente testado e publicado!

Resumo do Capítulo 6

✓ Entendemos o que é **CI/CD** e por que é essencial no desenvolvimento moderno.

✓ Criámos um **Pipeline de Build** para compilar o código.

✓ Adicionámos **testes automáticos** ao pipeline para garantir qualidade.

✓ Criámos um **Pipeline de Release** para automatizar o deploy.

✂ No próximo capítulo, vamos explorar **Azure Test Plans** para garantir ainda mais qualidade com testes manuais e automáticos! 🚀

Capítulo 7 – Testes e Qualidade com Azure Test Plans

(Introdução Básica)

Garantindo Qualidade com Testes Manuais e Automatizados

Agora que já aprendeste a criar pipelines para compilar e publicar código, é fundamental garantir que o software funciona corretamente. **O Azure Test Plans ajuda a planejar, executar e acompanhar testes para melhorar a qualidade do código antes de ir para produção.**

Neste capítulo, vais aprender a:

- ✓ Criar e gerir **planos de teste manuais**.
- ✓ Automatizar testes dentro do **Azure Pipelines**.

Vamos lá! 🚀

7.1 Criando Planos de Teste Manuais

Nem todos os testes podem ser automatizados. Algumas funcionalidades requerem **testes manuais**, como verificar a experiência do utilizador ou testar cenários complexos.

O que são Test Plans?

Os **Test Plans** são conjuntos de testes organizados dentro do **Azure Test Plans**. Permitem:

- Criar **casos de teste** passo a passo.
- Acompanhar quais testes passaram/falharam.
- Repetir testes em diferentes versões do software.

Criando um Plano de Teste no Azure DevOps

- 1** Vai a **Test Plans** → **New Test Plan**.
- 2** Dá um nome ao plano de testes e associa a um sprint.
- 3** Clica em **Create**.

Agora, dentro do **Test Plan**, podemos adicionar **Test Suites** e **Test Cases**.


Criando um Test Case (Caso de Teste)

Um **Test Case** é um conjunto de passos que um testador deve seguir para validar uma funcionalidade.

1 Dentro do **Test Plan**, clica em **New Test Case**.

2 Preenche os seguintes campos:

- **Título:** "Verificar Login com Credenciais Válidas"
- **Passos:** Lista os passos para o testador seguir.
- **Resultado esperado:** Descreve o que deveria acontecer.


 **Exemplo de Test Case:**

Passo	Ação	Resultado Esperado
1	Abrir a página de login	A página carrega corretamente
2	Inserir um utilizador válido	O campo aceita a entrada
3	Inserir a password correta	O campo aceita a entrada
4	Clicar no botão "Login"	O utilizador é autenticado e redirecionado

3 Guarda e associa este teste ao sprint atual.

4 Quando o testador executar o teste, pode marcar se **Passou ou Falhou**.

- ◆ **Dica:** Se o teste falhar, cria um **Bug** diretamente no Azure Boards para que a equipa resolva o problema.

Agora que aprendemos a criar **Test Plans**, vamos automatizar testes para ganhar eficiência! 

7.2 Automatizando Testes dentro do Azure Pipelines

Testes manuais são úteis, mas para garantir que todas as alterações não introduzem falhas, precisamos de **testes automatizados**.

Tipos de Testes Automatizados

- ◆ **Testes Unitários:** Testam partes individuais do código.
- ◆ **Testes de Integração:** Verificam se diferentes partes do sistema funcionam juntas.
- ◆ **Testes de UI (Selenium, Playwright):** Automatizam interações do utilizador.

Criando Testes Automatizados no Azure Pipelines

Agora vamos integrar testes automatizados no **Pipeline de Build**.

Criar Testes Automatizados

Se estás a usar .NET, certifica-te de que tens testes unitários no teu projeto, por exemplo:

Exemplo de um teste unitário simples (.NET)

```
using Xunit;

public class CalculadoraTests
{
    [Fact]
    public void Soma_DeveRetornarValorCorreto()
    {
        var resultado = Calculadora.Soma(2, 3);
        Assert.Equal(5, resultado);
    }
}
```

Para outras tecnologias:

- **Node.js:** Usa Jest (`npm test`).
- **Python:** Usa Pytest (`pytest`).
- **Java:** Usa JUnit (`mvn test`).

Adicionar Testes ao Pipeline

Agora vamos modificar o `azure-pipelines.yml` para executar testes automaticamente.

🚩 Exemplo de Pipeline com Testes Automatizados (.NET Core)

```
trigger:
- main

pool:
  vmImage: 'ubuntu-latest'

steps:
- task: UseDotNet@2
  inputs:
    packageType: 'sdk'
    version: '6.x'

- script: dotnet build --configuration Release
  displayName: 'Compilar a Aplicação'

- script: dotnet test --configuration Release
  displayName: 'Executar Testes'
```

Se algum teste falhar, o pipeline será interrompido.

🚩 Exemplo para Node.js:

```
steps:
- script: npm install
- script: npm test
```

🚩 Exemplo para Python:

```
steps:
- script: pip install -r requirements.txt
- script: pytest
```

Verificando os Resultados no Azure DevOps

- 1** Vai a **Pipelines** → **Runs** e seleciona a execução mais recente.
- 2** Clica em **Tests** para ver o relatório.
- 3** Se houver falhas, revisa o código antes de aprovar um merge.

Agora os testes são executados automaticamente em todas as alterações no código! 🚀

Resumo do Capítulo 7

- ✓ Criámos **Planos de Teste Manuais** para validar funcionalidades importantes.
 - ✓ Definimos **Test Cases** para guiar os testadores no Azure Test Plans.
 - ✓ Automatizámos testes unitários dentro do **Azure Pipelines**.
 - ✓ Configurámos **relatórios de testes** para acompanhar falhas e garantir qualidade.
- ✦ **No próximo capítulo**, vamos explorar **Azure Artifacts** para gerir pacotes e dependências no projeto! 🚀

(capítulo 8 omitido por tratar de temas muito específicos que não deves precisar para já 😊)

Capítulo 9 – Melhores Práticas e Próximos Passos

Elevando a Eficiência no Uso do Azure DevOps

Agora que já aprendeste os fundamentos do **Azure DevOps**, desde **gestão de código e tarefas** até **automação de testes e deploys**, é essencial conhecer as **melhores práticas** para otimizar o uso da plataforma.

Neste capítulo, vais aprender a:

- ✓ Aplicar boas práticas para manter a organização e eficiência no Azure DevOps.
- ✓ Consultar recursos para continuar a evoluir no domínio da ferramenta.
- ✓ Explorar temas avançados como **Infraestrutura como Código, Monitorização e Segurança**.

Vamos lá! 🚀

9.1 Boas Práticas para Equipas que Usam Azure DevOps

Trabalhar com **Azure DevOps** de forma eficiente exige **disciplina, organização e colaboração**. Aqui estão as melhores práticas para equipas:

🔗 Gestão de Código e Git

- ✓ Usar **Feature Branches** ou **GitFlow** para manter o código organizado.
- ✓ Criar **Pull Requests** antes de integrar código na main (nunca fazer commits diretos!).
- ✓ Configurar **Políticas de Branch** para evitar erros e garantir revisões de código.
- ✓ Escrever **mensagens de commit descritivas** (evita Update code ou Fix bug).

♦ Exemplo de uma boa mensagem de commit:

[Login] Corrigido erro de autenticação ao inserir credenciais inválidas

Gestão de Tarefas com Azure Boards

- ✓ **Manter o backlog sempre atualizado** com prioridades claras.
 - ✓ **Associar cada commit ou Pull Request a uma User Story ou Task.**
 - ✓ **Definir sprints realistas**, evitando sobrecarga da equipa.
 - ✓ **Utilizar Queries** para monitorizar progresso e identificar bloqueios.
 - ◆ **Dica:** Usa etiquetas e comentários nos Work Items para facilitar o acompanhamento.
-

Pipelines e Automação

- ✓ **Executar testes automáticos em todos os Pull Requests** antes de aprovar merges.
 - ✓ **Criar Pipelines de Build eficientes**, sem etapas desnecessárias.
 - ✓ **Configurar notificações para falhas** para resolver problemas rapidamente.
 - ✓ **Utilizar variáveis de ambiente** para evitar armazenar senhas diretamente no código.
-

Segurança e Permissões

- ✓ **Definir permissões corretamente** (evitar que todos tenham acesso de admin).
 - ✓ **Usar Service Connections Seguras** ao integrar com outros serviços (exemplo: Azure, AWS).
 - ✓ **Habilitar Logs e Monitorização** para rastrear falhas e atividades suspeitas.
 - ◆ **Dica:** Ativa a autenticação multifator (MFA) para proteger a conta do Azure DevOps.
-


9.2 Recursos para Aprofundamento e Documentação Oficial

Agora que já tens uma base sólida, podes explorar **documentação oficial e cursos avançados** para te tornares um especialista em Azure DevOps.

Documentação Oficial

 [Azure DevOps Docs](#) – Guia completo da Microsoft.

Cursos Online e Vídeos

 [Microsoft Learn](#) – Azure DevOps – Treinamentos gratuitos da Microsoft.

 [Pluralsight](#) – Azure DevOps – Cursos técnicos detalhados.

 YouTube – Há diversos canais com demonstrações práticas (pesquisar por "Azure DevOps tutorial").

9.3 Próximos Desafios: Infraestrutura como Código, Monitorização e Segurança

Se quiseses levar os teus conhecimentos ao **próximo nível**, podes explorar **tópicos avançados** no Azure DevOps.

Infraestrutura como Código (IaC)

Em vez de configurar servidores manualmente, podemos definir **toda a infraestrutura como código** usando ferramentas como:

- **Terraform** (terraform.io)
- **Azure Resource Manager (ARM)**

◆ Exemplo de um ficheiro Terraform para criar uma VM no Azure:

```
resource "azurerm_virtual_machine" "vm1" {  
  name                = "minha-maquina"  
  location            = "West Europe"  
  resource_group_name = "meu-grupo"  
  vm_size             = "Standard_DS1_v2"  
}
```

Por que aprender IaC?

 Evita configurações manuais.

- ✅ Permite reverter mudanças facilmente.
 - ✅ Garante consistência entre ambientes.
-

Monitorização e Logs

Mesmo depois de lançar uma aplicação, precisas garantir que está a funcionar corretamente. O Azure oferece ferramentas como:

- **Azure Monitor** – Para visualizar métricas e logs.
- **Application Insights** – Para monitorizar aplicações web e APIs.

♦ Exemplo de query no Azure Monitor para encontrar falhas:

exceptions

| where timestamp > ago(24h)

| order by timestamp desc

✦ Por que aprender Monitorização?

- ✅ Identifica falhas rapidamente.
 - ✅ Melhora a experiência dos utilizadores.
 - ✅ Reduz tempo de inatividade da aplicação.
-

Segurança no Azure DevOps

A segurança deve ser sempre uma prioridade. Algumas práticas recomendadas incluem:

- ✅ **Usar Azure Key Vault** para armazenar senhas e chaves API.
- ✅ **Implementar varredura de vulnerabilidades nos pipelines** com ferramentas como **SonarQube**.
- ✅ **Configurar RBAC (Role-Based Access Control)** para garantir que cada utilizador só tem acesso ao que precisa.

♦ Exemplo de configuração de uma Secret no Azure Key Vault:

```
az keyvault secret set --vault-name "MeuCofre" --name "SenhaBD" --value "SuperSecreta123"
```


Por que aprender Segurança?

- ✓ Protege dados sensíveis.
 - ✓ Evita acessos indevidos.
 - ✓ Previne falhas de segurança no código.
-

Resumo do Capítulo 9

- ✓ Aplicámos **boas práticas** para equipas que usam Azure DevOps.
 - ✓ Descobrimos **recursos oficiais e cursos** para aprofundamento.
 - ✓ Explorámos temas avançados como **Infraestrutura como Código, Monitorização e Segurança**.
-

Conclusão do Tutorial

 **Parabéns!** Agora tens uma base sólida para trabalhar com **Azure DevOps**, desde o **controlo de código e gestão de projetos** até **automação de builds, testes e deploys**.

◆ **Próximos passos recomendados:**

- ✓ Experimenta usar **Terraform ou ARM** para criar infraestrutura automaticamente.
- ✓ Aprende a monitorizar aplicações com **Azure Monitor e Application Insights**.
- ✓ Implementa políticas de segurança para proteger os teus projetos.

 **Agora é contigo! Começa a explorar e a aplicar o que aprendeste!** 