

MBA
USP
ESALQ

DevOps

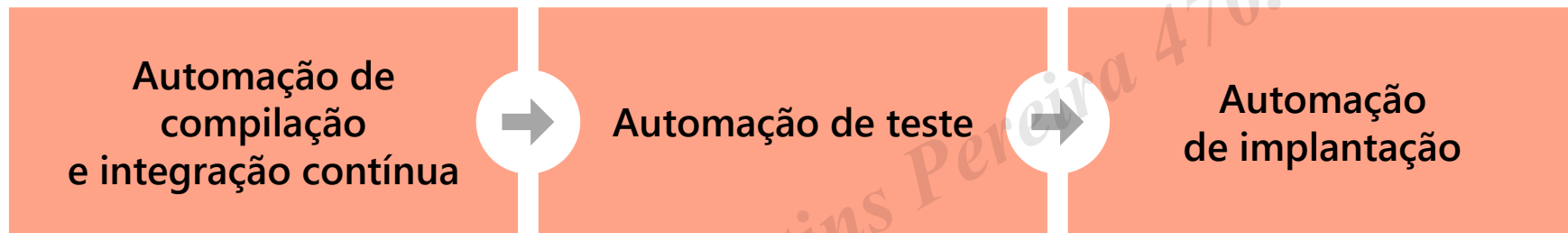
Rogério Rodrigues

*A responsabilidade pela idoneidade, originalidade e licitude dos conteúdos didáticos apresentados é do professor.

Proibida a reprodução, total ou parcial, sem autorização. Lei nº 9610/98

Azure Pipelines

Explorar o conceito de pipelines no DevOps



- O pipeline possibilita um fluxo constante de alterações na produção por meio de uma linha de produção de software automatizada
- Os pipelines criam um processo reproduzível, confiável e de melhoria incremental para levar o software da fase conceitual ao cliente
- Os pipelines exigem infraestrutura, que terá um impacto direto na eficácia deles

Descrever o Azure Pipelines

1

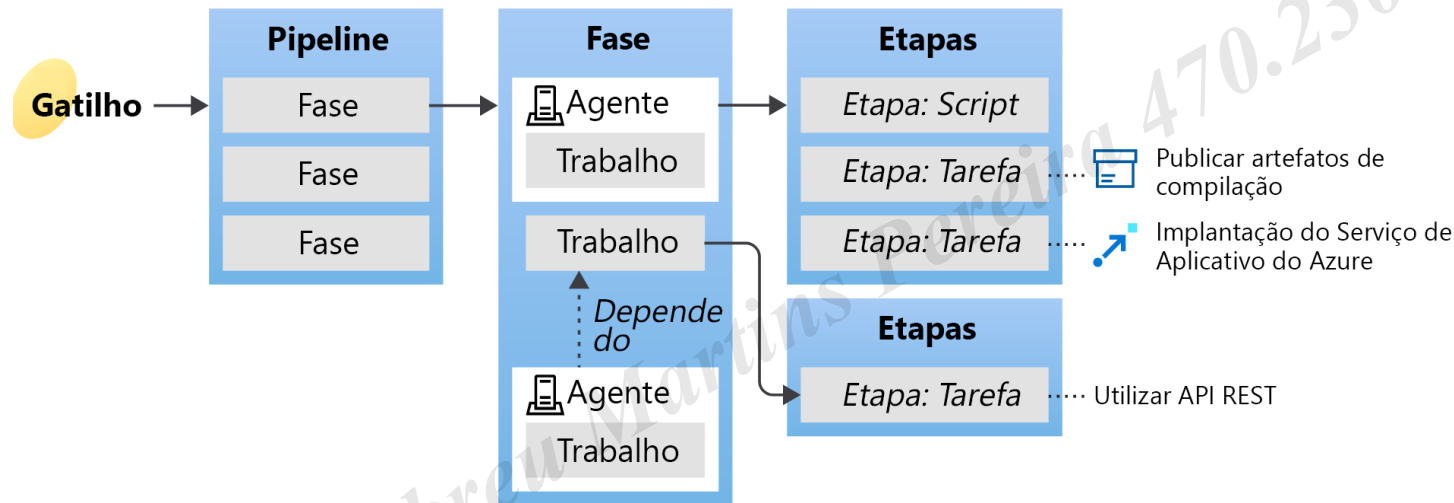
O Azure Pipelines é um serviço de nuvem que você pode usar para criar e testar automaticamente o projeto de código e disponibilizá-lo para outros usuários.

2

É ideal para a integração contínua e entrega contínua:

- Trabalhe com qualquer linguagem ou plataforma – Python, Java, PHP, Ruby, C# e Go
- Implante em diferentes tipos de destinos ao mesmo tempo
- Integre com implantações do Azure – Registros de contêiner, máquinas virtuais, serviços do Azure ou qualquer destino local ou na nuvem (Microsoft Azure, Google Cloud ou AWS)
- Crie em computadores Windows, Linux ou macOS
- Integrar com o GitHub
- Trabalhe com projetos de código aberto

Entender os principais termos do Azure Pipelines



CD (Entrega contínua)

CI (Integração contínua)

Destino de implantação

Pools e agentes do Azure Pipelines

Escolher entre agentes hospedados pela Microsoft versus auto-hospedados

- Para criar seu código ou implantar um projeto, geralmente você precisa de pelo menos um agente.
- Um agente é um software instalável que executa um trabalho de build ou implantação por vez.

Há dois tipos de agentes:

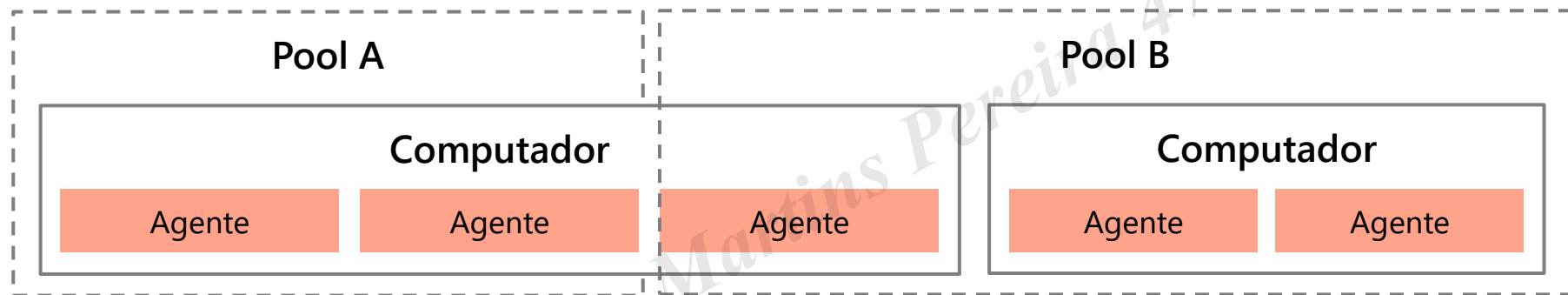
- 1** **Agentes hospedado pela Microsoft**– Manutenção e atualizações são feitas automaticamente. Cada vez que um pipeline é executado, uma nova máquina virtual (instância) é fornecida. Há limites de tempo para trabalhos executados nesses agentes.
- 2** **Agentes auto-hospedados** – você cuida da manutenção e das atualizações. Tenha mais controle para instalar o software dependente necessário. Pode ser instalado em computadores Linux, macOS, Windows ou em um contêiner do Docker Linux. Não há limites de tempo para trabalhos executados nesses agentes.

Explorar tipos de trabalho

- Há quatro tipos de tarefas:

- 1** **Trabalhos do pool de agentes**– Trabalhos executados em um agente de um pool de agentes.
- 2** **Trabalhos de contêiner**– Trabalhos executados em um contêiner em um agente de um pool de agentes.
- 3** **Trabalhos em grupos de implantação**– Trabalhos executados em sistemas em um grupo de implantação
- 4** **Trabalhos sem agente**– Trabalhos executados no Azure DevOps Server (também chamados de trabalhos de servidor)

Introdução aos pools de agentes



Você pode organizar os agentes em pools de agentes.

Um pool de agentes define o limite de compartilhamento.

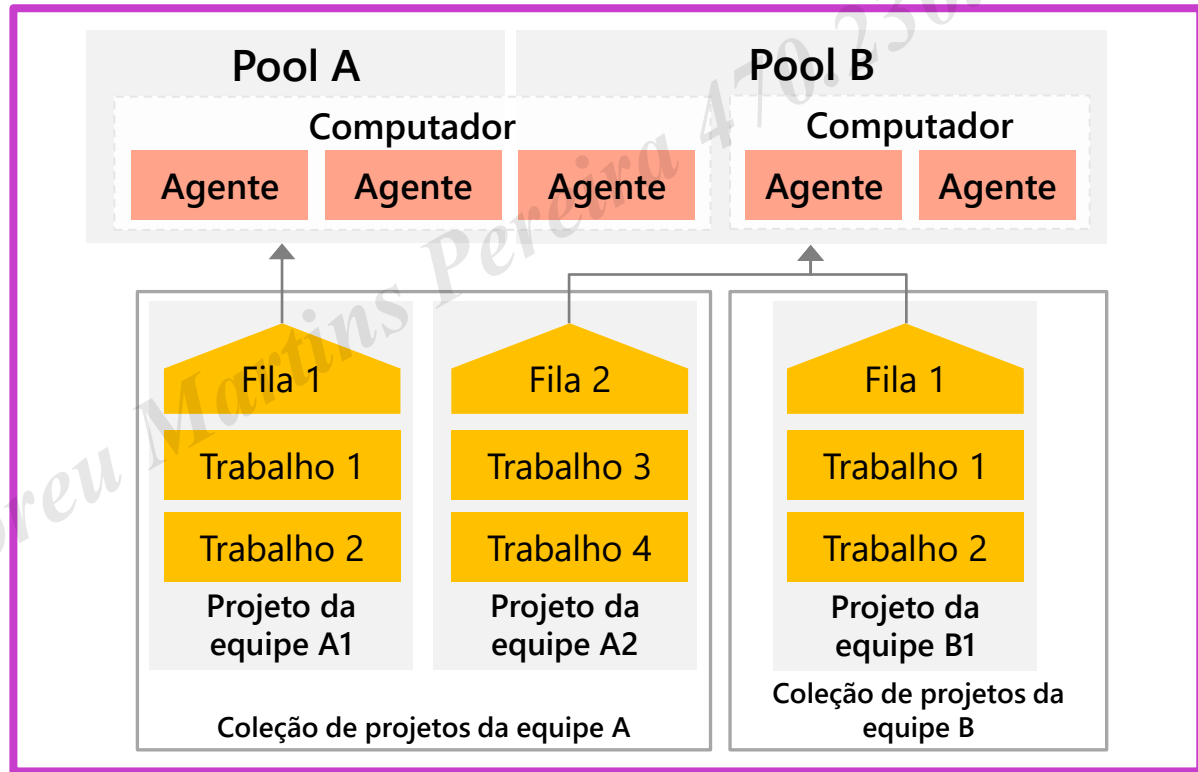
No Azure Pipelines, os pools de agentes têm como escopo a organização do Azure DevOps, de modo que é possível compartilhar um pool de agentes entre projetos.

Explorar pool de agentes predefinido

Imagem	Objetivo	Rótulo da imagem de VM do YAML
Windows Server 2022 com Visual Studio 2022	Windows-2022	Windows mais recente ou windows-2022
Windows Server 2019 com Visual Studio 2019	Windows-2019	Windows-2019
Ubuntu 22.04	Ubuntu-22.04	Ubuntu-mais recente OU ubuntu-22.04
Ubuntu 20.04	Ubuntu-20.04	Ubuntu-20.04
macOS 12 Monterey	MacOS-12	MacOS mais recente OU macOS-12
macOS 11 Big Sur	MacOS-11	MacOS-11

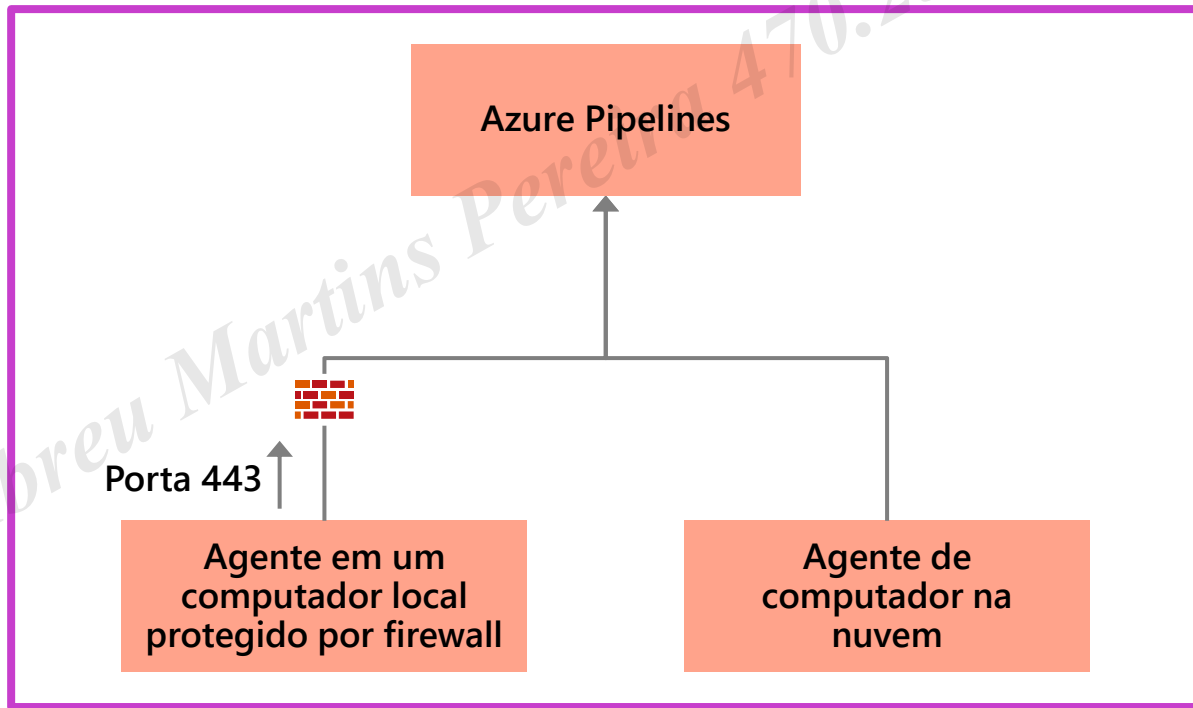
Entender as situações típicas para pools de agentes

- Você participa de um projeto e quer usar um conjunto de computadores da equipe para executar trabalhos de compilação e implantação.
- Você participa da equipe de infraestrutura e gostaria de configurar um pool de agentes para uso em todos os projetos.
- Você quer compartilhar um conjunto de computadores de agente com vários projetos, mas não todos.



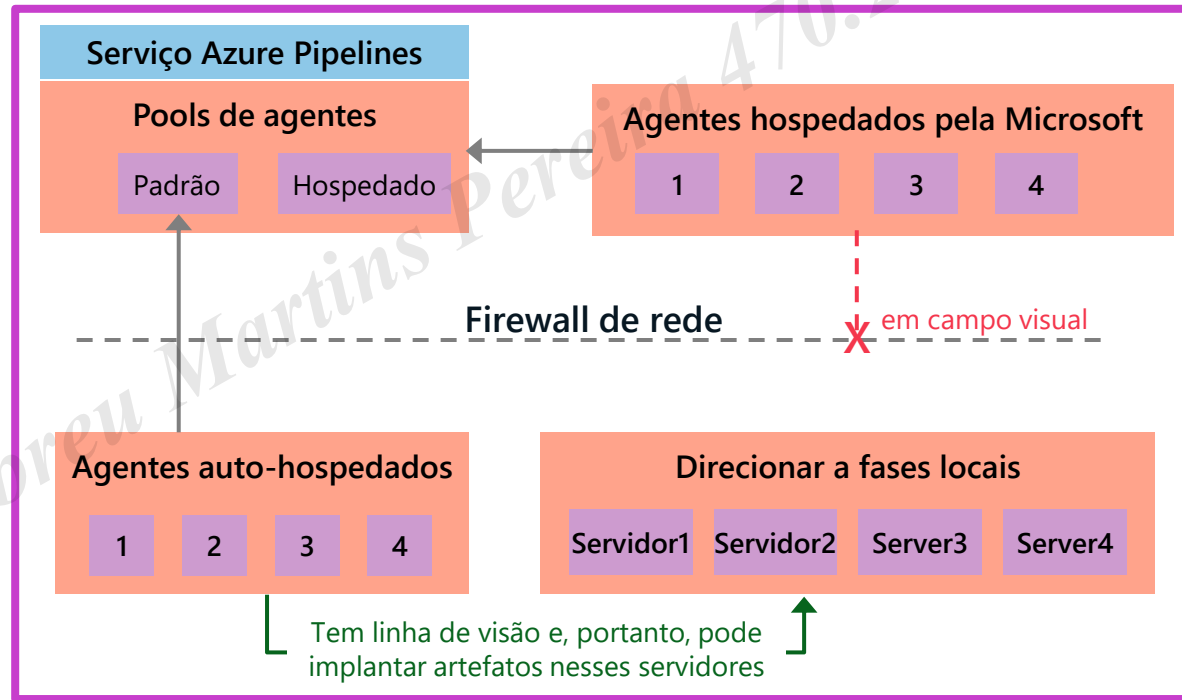
Comunicar-se com o Azure Pipelines

- O agente determina qual trabalho ele precisa executar, relatar os logs e o status.
- A comunicação é sempre iniciada pelo agente.
- Todas as mensagens do agente para o Azure Pipelines são enviadas por HTTPS.



Comunicar-se para implantar em servidores de destino

- O agente deve ter conectividade de "linha de visão" com os servidores.
- Por padrão, os pools de agentes hospedados pela Microsoft têm conectividade com sites e servidores do Azure em execução no Azure.
- Você precisará configurá-lo manualmente.



Examinar outras considerações

- 1 Autenticação
- 2 Tokens de acesso pessoal
- 3 Processos interativos versus de serviço
- 4 Versão e atualizações do agente

Descrever a segurança de pools de agentes

Função	Objetivo
Leitor	Os membros dessa função podem visualizar o pool de agentes da organização e os agentes. Normalmente, você usa isso para adicionar operadores responsáveis por monitorar os agentes e a integridade deles.
Conta de serviço	Os membros dessa função podem usar o pool de agentes da organização para criar um pool de agentes de projeto em um projeto. Se você seguir as diretrizes acima para criar pools de agentes de projeto, normalmente não será preciso adicionar nenhum membro aqui.
Administrador	Além de todas as permissões acima, os membros dessa função podem registrar ou cancelar o registro de agentes do pool de agentes da organização. Também podem consultar o pool de agentes da organização ao criar um pool de agentes de projeto. Por fim, também podem gerenciar a associação para todas as funções do pool de agentes da organização. O usuário que criou o pool de agentes da organização é adicionado automaticamente à função Administrador.

Pipelines e a Parallels Jobs

Módulo 03: descrever pipelines e a simultaneidade

Estimar trabalhos paralelos

- Determine quantos trabalhos paralelos são necessários.
- Estimativas simples versus estimativas detalhadas
- Você pode exibir todas as builds e versões.
- Os trabalhos paralelos são comprados no nível da organização e compartilhados por todos os projetos.

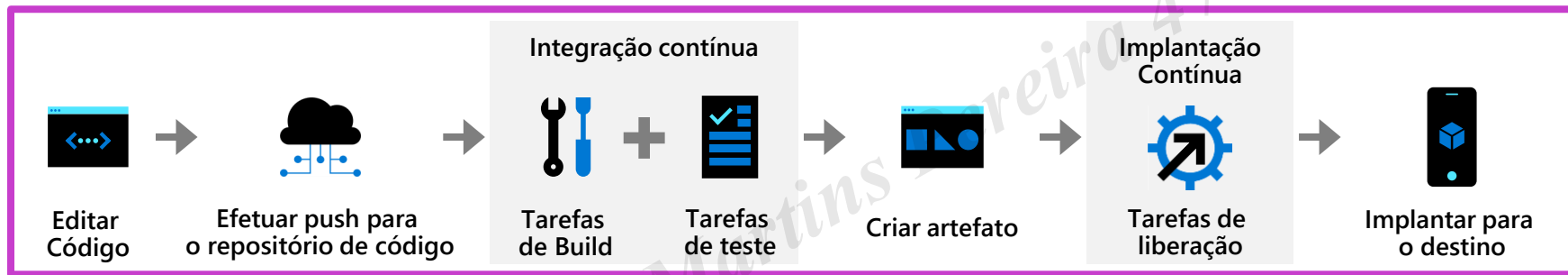
The screenshot displays the 'Organization Settings' page in Azure DevOps. The left sidebar contains a navigation menu with categories: Global notifications, Usage, Extensions, Azure Active Directory, Security (Policies, Permissions), Boards, Process, Pipelines (Agent pools, Settings, Deployment pools), Parallel jobs (highlighted), and OAuth configurations. The main content area is titled 'Private projects' and shows a table of parallel job limits. Below this, a section for 'Public projects' also shows a table of limits. The 'Parallel jobs' section in the sidebar is highlighted, indicating the current view.

Project Type	Configuration	Parallel Jobs
Private projects	Microsoft-hosted (Free tier)	1 parallel job up to 1800 mins/mo
	Self-hosted	3 Parallel jobs
	Free parallel jobs	1
	Visual Studio Enterprise subscribers	2
Public projects	Microsoft-hosted	10 Parallel jobs
	Self-hosted	Unlimited Parallel jobs

Descrever Azure Pipelines e projetos de código aberto

- 1** Como me qualificar para a camada gratuita do Azure Pipelines para projetos públicos?
 - O projeto deve ser público e
 - Criar um repositório público
- 2** Há limites a quem pode usar o Azure Pipelines?
 - Usuários ilimitados – sem custo por usuário
 - Usuários com acesso Básico e Stakeholder podem criar pipelines
- 3** Há limites ao número de builds e pipelines de lançamento que posso criar?
 - Sem limites no número de pipelines
 - Agentes de build auto-hospedados ilimitados
 - Dez trabalhos paralelos gratuitos
- 4** Como assinante do Visual Studio Enterprise, recebo trabalhos paralelos adicionais do Azure Pipelines?
 - Assinantes receberão um trabalho paralelo auto-hospedado em cada organização

Explorar o Azure Pipelines e o Visual Designer



Crie e configure os pipelines de build e versão.

Efetue push do código para o repositório de controle de versão.

A build cria um artefato que é usado pelo resto do pipeline.

Agora seu código está atualizado, compilado, testado e empacotado.

Geralmente conhecido como "Pipelines Clássicos"

Descrever o Azure Pipelines e o YAML



Configure o Azure Pipelines para usar seu repositório Git.

Edite o arquivo Azure-pipelines.yml para definir a build.

Efetue push do código para o repositório de controle de versão.

Agora seu código está atualizado, compilado, testado e empacotado.

Embora mais orientada a código, a definição de pipelines usando YAML é preferida

Introdução a Integração Contínua

Introdução à integração contínua

- 1 A CI (integração contínua) é o processo de automatizar a compilação e o teste do código
- 2 A CI incentiva os desenvolvedores a compartilhar códigos e testes de unidade mesclando as alterações no repositório compartilhado de controle de versão
- 3 Quando uma alteração é detectada, ela aciona um sistema de compilação automatizado. O código é compilado usando uma definição de compilação. Os desenvolvedores respondem a quaisquer problemas ou bugs
- 4 A CI mantém a ramificação principal limpa, garantindo que os bugs sejam detectados no início do ciclo de desenvolvimento, tornando a correção menos dispendiosa

Conhecer os quatro pilares da integração contínua



Um **sistema de controle de versão** gerencia as alterações no código-fonte ao longo do tempo.

Um **sistema de gerenciamento de pacotes** é usado para instalar, desinstalar e gerenciar pacotes de software.

Um **sistema de integração contínua** mescla todas as cópias de trabalho do desenvolvedor em uma linha principal compartilhada várias vezes por dia.

Um **processo de compilação automatizado** cria uma compilação de software, incluindo a compilação, a compactação e a execução de testes automatizados.

Explorar os benefícios da integração contínua

- 1 Aprimoramento da qualidade do código com base em comentários rápidos
- 2 Disparo de testes automatizados para cada alteração de código
- 3 Redução dos tempos de compilação para acelerar os comentários e detectar problemas antecipadamente (redução de risco)
- 4 Aprimoramento do gerenciamento de dívidas técnicas e da análise de código
- 5 Redução de mesclagens longas, complexas e propensas a bugs
- 6 Aumento da confiança na integridade da base de código muito antes da implantação de produção
- 7 Comentário rápido para o desenvolvedor

Discussão: integração contínua

1 Você tentou implementar a integração contínua na organização?

2 Se você teve êxito, quais lições você aprendeu?

3 Se você não teve êxito, quais foram os desafios?

Integração com Pipelines

Descrever a anatomia de um pipeline

- 1 Nome** – Embora geralmente seja ignorado (se for ignorado, um nome baseado em data será gerado automaticamente)
- 2 Gatilho** – Mais sobre gatilhos mais tarde, mas na ausência de um explícito, haverá um "gatilho implícito em cada confirmação para qualquer caminho de qualquer ramificação neste repositório"
- 3 Variáveis** – Variáveis "em linha" (mais sobre outros tipos de variáveis posteriormente)
- 4 Trabalho** – Cada pipeline deve ter pelo menos um trabalho
- 5 Pool** – Você configura em qual pool (fila) o trabalho deve ser executado
- 6 Verificação** – O "check-out: self" informa ao trabalho qual repositório (ou repositórios se houver várias check-outs) deve fazer o check-out
- 7 Etapas** – As tarefas reais que precisam ser executadas: nesse caso, uma tarefa de "script" (o script é um alias) que pode executar scripts em linha

Entender a estrutura do pipeline

- 1 Pipeline** – Um ou mais estágios que descrevem um processo de CI/CD
- 2 Estágio** – Coleção de trabalhos relacionados
- 3 Trabalho** – É uma coleção de etapas executadas por um agente em um servidor
- 4 Etapa** – É uma sequência linear de operações que compõem um trabalho
- 5 Tarefas** – São os blocos de construção de um pipeline

Detalhar modelos YAML

O Azure Pipelines aceita a quatro tipos de modelos:

1 Estágio

2 Trabalho

3 Step

4 Variável

Explorar recursos do YAML

Pipelines

Repositórios

Contêineres

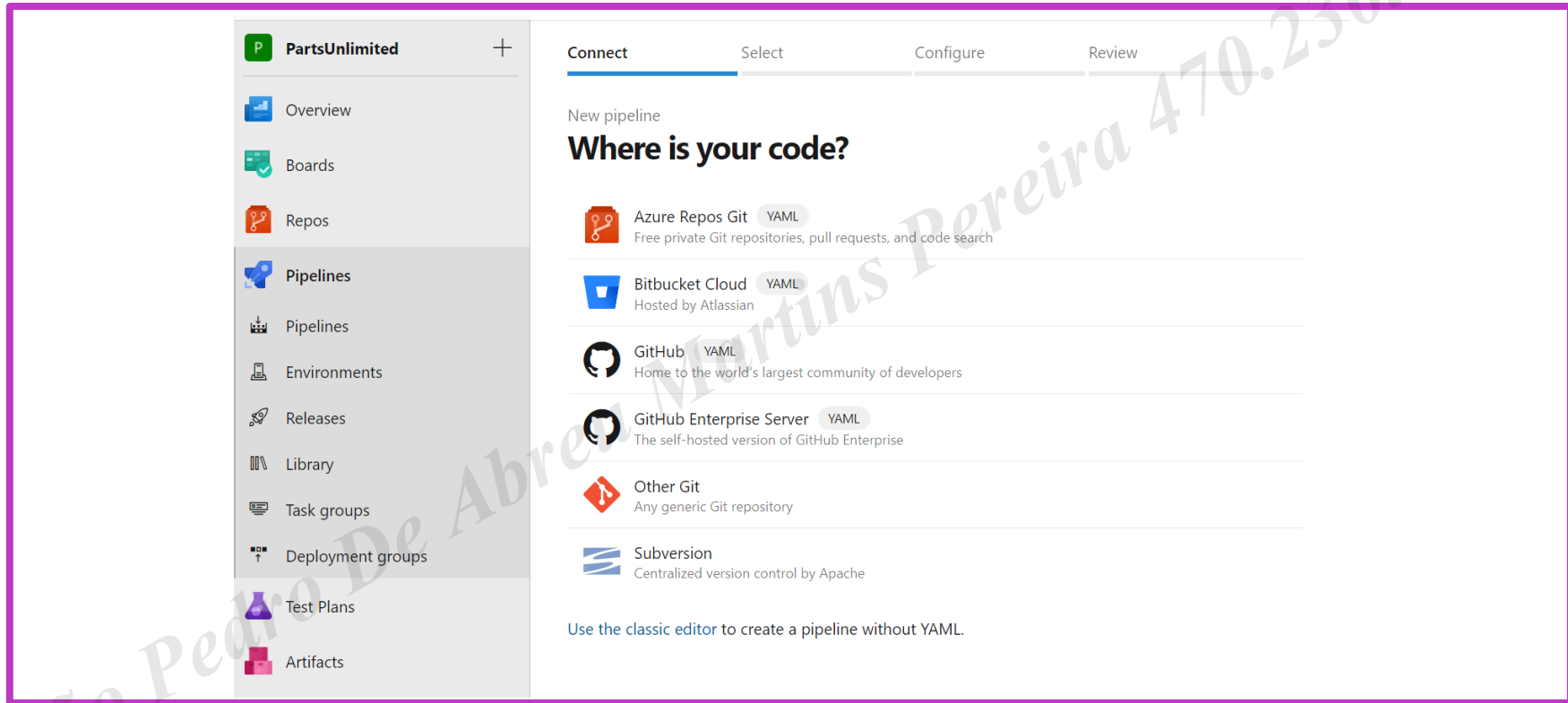
```
resources:  
  pipelines: [ pipeline ]  
  repositories: [ repository ]  
  containers: [ container ]
```

```
resources:  
  pipelines:  
    - pipeline: MyAppA  
      source: MyCIPipelineA  
    - pipeline: MyAppB  
      source: MyCIPipelineB  
      trigger: true  
    - pipeline: MyAppC  
      source: MyCIPipelineC  
      branch: releases/M174  
      version: 20230709.2  
      trigger:  
        branches:  
          include:  
            - main  
            - releases/*  
          exclude:  
            - users/*
```

Usar vários repositórios no seu pipeline

- 1 Comum para utilitários usados em mais de um pipeline
- 2 Adicionar etapas extras de check-out
- 3 Não era aceito em versões anteriores e artefatos foram usados como uma solução alternativa
- 4 Alguns pipelines podem não precisar de repositórios
- 5 Com o Azure Pipelines, é possível criar e validar cada solicitação de pull e fazer commit para seu repositório do **GitHub**.

Repositório GitHub com o Azure Pipelines



The screenshot displays the Azure Pipelines interface for creating a new pipeline. On the left, a sidebar shows the navigation menu with options like Overview, Boards, Repos, Pipelines (selected), Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The main area is titled 'New pipeline' and 'Where is your code?'. It features a progress bar with steps: Connect, Select, Configure, and Review. Below the progress bar, several code providers are listed with their respective icons and descriptions:

- Azure Repos Git** (YAML): Free private Git repositories, pull requests, and code search.
- Bitbucket Cloud** (YAML): Hosted by Atlassian.
- GitHub** (YAML): Home to the world's largest community of developers.
- GitHub Enterprise Server** (YAML): The self-hosted version of GitHub Enterprise.
- Other Git**: Any generic Git repository.
- Subversion**: Centralized version control by Apache.

At the bottom, a link states: [Use the classic editor](#) to create a pipeline without YAML.

Github Actions

O que são Actions?

- 1 Automações no ambiente do GitHub
- 2 Frequentemente usado para compilar implementações de CI/CD
- 3 Com base em arquivos YAML nos repositórios do GitHub
- 4 Executado no GitHub ou em executores auto-hospedados
- 5 Grande número de ações existentes no GitHub Marketplace

Explorar o fluxo de Actions

Eventos que disparam fluxos de trabalho:

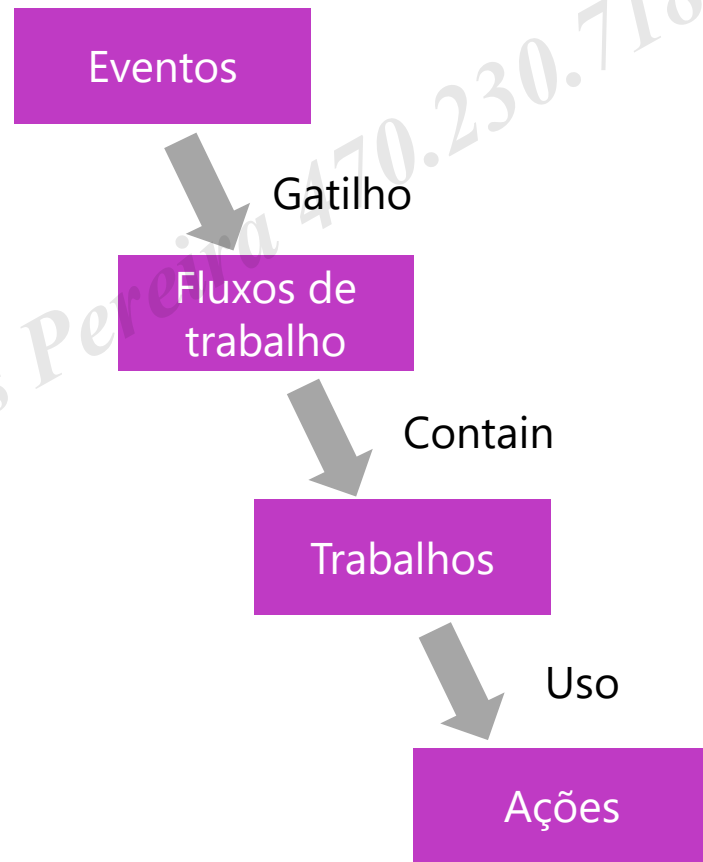
- Cronograma, código, etc.

Os fluxos de trabalho contêm trabalhos:

- Pode conter vários

Trabalhos usam ações:

- Configurado dentro de etapas



Noções básicas de fluxos de trabalho

Defina a automação necessária:

- Eventos e trabalhos
- Escrito em YAML
- Armazenado em `.github/workflows`
- Fluxos de trabalho iniciais disponíveis

```
# .github/workflows/build.yml
name: Node Build

on: push

jobs:
  mainbuild:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        node-version: [12.x]
        os: [windows-latest]

    etapas:
      - uses: actions/checkout@v1
      - name: Run node.js on latest Windows
        uses: actions/setup-node@v1
      por:
        node-version: ${{ matrix.node-version }}
      - name: Install NPM and build
        run: |
          npm ci
          npm run build
```

Descrever elementos de sintaxe de fluxo de trabalho padrão

- 1 Nome:** O nome do fluxo de trabalho (opcional, mas recomendado)
- 2 Ativado:** O evento que aciona o fluxo de trabalho
- 3 Trabalhos:** É a lista de trabalhos a serem executados.
- 4 Executado em:** O executor do destino
- 5 Etapas:** O conjunto de etapas a serem executadas
- 6 Usos:** Uma ação predefinida para recuperar
- 7 Executar:** Informa ao trabalho para executar um comando no executor

Explorar Eventos

Definido pela cláusula **ativado**

- Eventos agendados
- Eventos de código
- Eventos manuais
- Eventos de webhook
- Eventos externos

```
em:  
  schedule:  
    - cron: '0 8-17 * * 1-5'
```

```
em:  
  pull_request
```

```
em:  
  [push, pull_request]
```

```
em:  
  pull_request:  
    branches:  
      - develop
```

```
em:  
  gollum
```


Explorar trabalhos

```
jobs:
  inicialização:
    runs-on: ubuntu-latest
    etapas:
      - run: ./setup_server_configuration.sh
  build:
    etapas:
      - run: ./build_new_server.sh
```

Os fluxos de trabalho contêm um ou mais trabalhos. Os trabalhos contêm um conjunto de etapas a serem executadas em ordem.

```
jobs:
  inicialização:
    runs-on: ubuntu-latest
    etapas:
      - run: ./setup_server_configuration.sh
  build:
    needs: startup ←
    etapas:
      - run: ./build_new_server.sh
```

Os trabalhos são executados em paralelo por padrão, mas podem ser configurados com dependências.

Explorar os executores

As etapas do trabalho são executadas em executores:

- Script de shell
- Ações

Para código JavaScript e executores hospedados em Node.js:

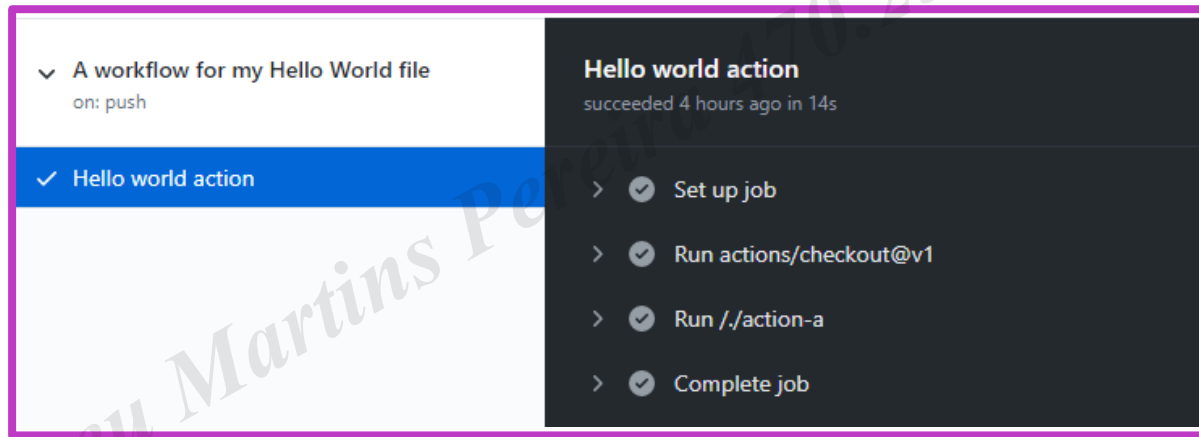
- Windows
- MacOS
- Linux

Para outras linguagens ou opções:

- Contêineres de Docker no Linux
- Auto-hospedado, mas não use em repositórios públicos

Saída do console de ações

- A saída do console das ações está disponível diretamente na interface do usuário do GitHub



Examinar a versão e testar uma ação

É possível solicitar liberações específicas de ações:

- Usar tags
- Usar branches
- Uso de hashes de SHA

```
etapas:  
- uses: actions/install-timer@v2.0.1
```

```
etapas:  
- uses: actions/install-timer@develop
```

```
etapas:  
- uses: actions/install-timer@327239021f7cc39fe7327647b213799853a9e
```

Integração Contínua com Github Actions

Descrever a integração contínua com Actions

Exemplo de fluxo de trabalho mostrado:

- Executa quando o código é enviado por push
- Funciona na versão mais recente do ubuntu
- Usa a versão 10 do Node.js

Etapas:

- Conferir o código
- Configurar dotnet
- Compilar o projeto

```
name: dotnet Build
```

```
on: [push]
```

```
jobs:
```

```
  build:
```

```
    runs-on: ubuntu-latest
```

```
    strategy:
```

```
      matrix:
```

```
        node-version: [10.x]
```

```
etapas:
```

```
- uses: actions/checkout@main
```

```
- uses: actions/setup-node@v1
```

```
por:
```

```
dotnet-version: '3.1.x'
```

```
- run: dotnet build eShopOnWeb
```

Examinar variáveis de ambiente

Os fluxos de trabalho de CI geralmente precisam de variáveis de ambiente

- Variáveis padrão do GitHub (GITHUB_ prefix)
- Variáveis personalizadas

```
jobs:
  verify-connection:
    etapas:
      - name: Verify Connection to SQL Server
        run: node testconnection.js
    env:
      PROJECT_SERVER: PH202323V
      PROJECT_DATABASE: HAMain
```

Compartilhar artefatos entre trabalhos

Os fluxos de trabalho de CI precisam passar artefatos entre trabalhos. Ações padrão:

- Upload-artifact
- Download-artifact

Pode configurar retenção

```
uses: actions/upload-artifact
por:
  name: harness-build-log
  path: bin/output/logs/harness.log
```

```
uses: actions/upload-artifact
por:
  name: harness-build-logs
  path: bin/output/logs/
```

```
uses: actions/upload-artifact
por:
  name: harness-build-logs
  path: bin/output/logs/harness[ab]?/*
```

```
uses: actions/upload-artifact
por:
  name: harness-build-log
  path: |
    bin/output/logs/harness.log
    bin/output/logs/harnessbuild.txt
```

Examinar notificações do fluxo de trabalho

Mostra o status de um fluxo de trabalho em um repositório

Normalmente adicionado ao README.md

Pode ser específico da ramificação

Adicionado por meio de URLs

<https://github.com/accountname/repositoryname/workflows/workflowname/badge.svg>

Os espaços nos nomes devem ser codificados como %20

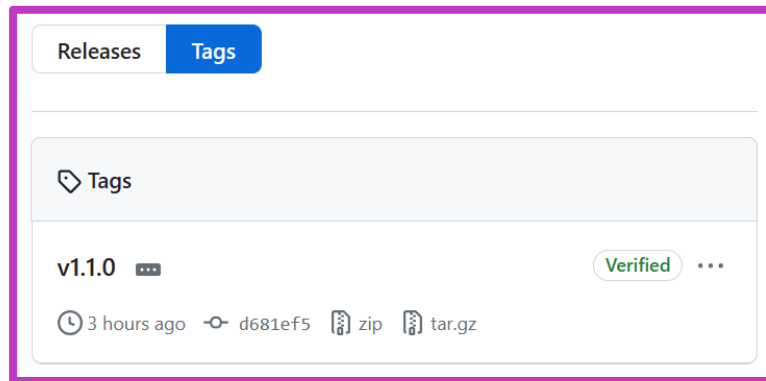
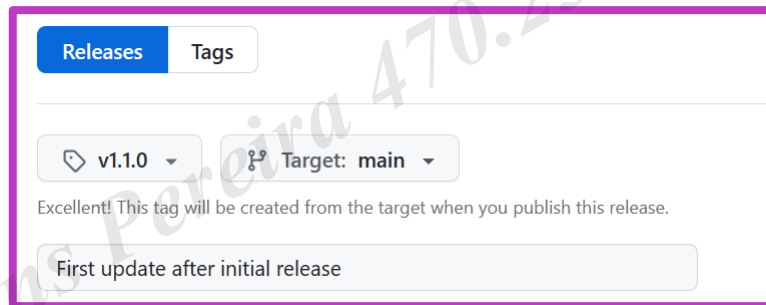


Descrever as práticas recomendadas para a criação de ações

- 1 Crie ações encadeáveis – evite ações monolíticas
- 2 Controlar a versão das ações como outro código
- 3 Fornecer um rótulo **mais recente**
- 4 Adicionar a documentação apropriada: como README.md
- 5 Adicionar metadados de **action.yml** detalhados
- 6 Considerar contribuir para o marketplace

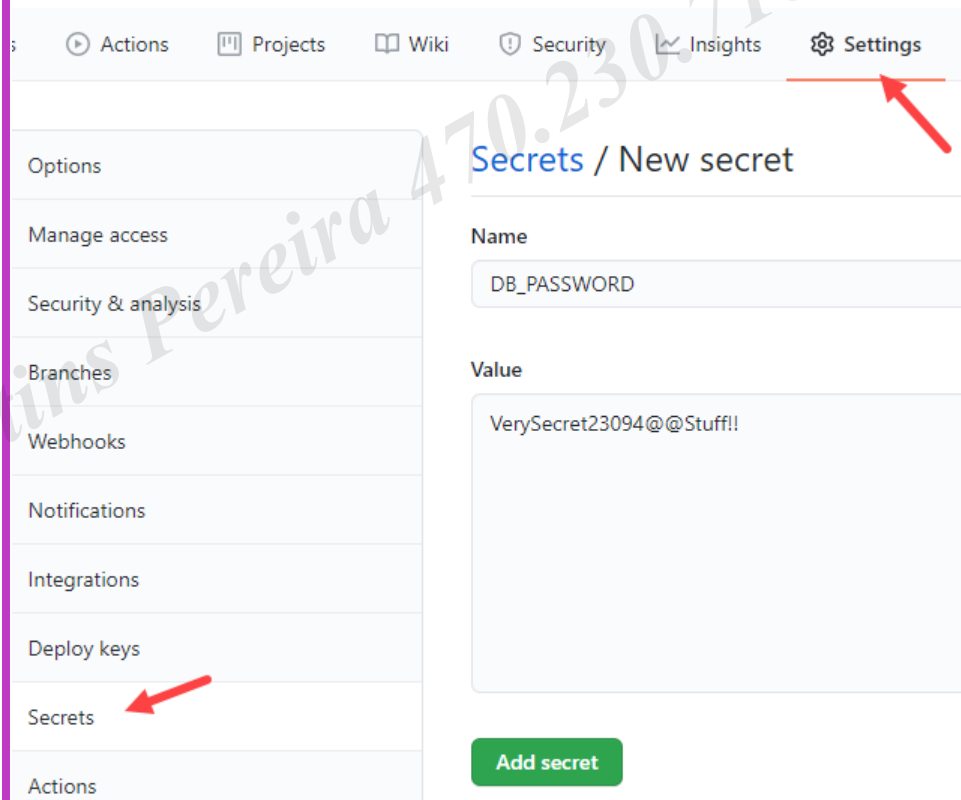
Marcar versões com marcas Git

- As versões são baseadas em tags Git
- Marcar um ponto específico no histórico do repositório
- As tags podem ser exibidas no histórico de um repositório



Criar segredos criptografados

- Como variável de ambiente, mas criptografada
- Criado no nível do repositório ou da organização
- Criado/atribuído na interface de usuário do GitHub



Usar segredos em um fluxo de trabalho

- Segredos não passados automaticamente aos executores
- Podem ser passados como entradas ou variáveis de ambiente
- Evite passar segredos em argumentos da linha de comando

```
steps: actions/upload-artifact
  - name: Test Database Connectivity
    por:
      db_username: ${ secrets.DBUserName }
      db_password: ${ secrets.DBPassword }
```

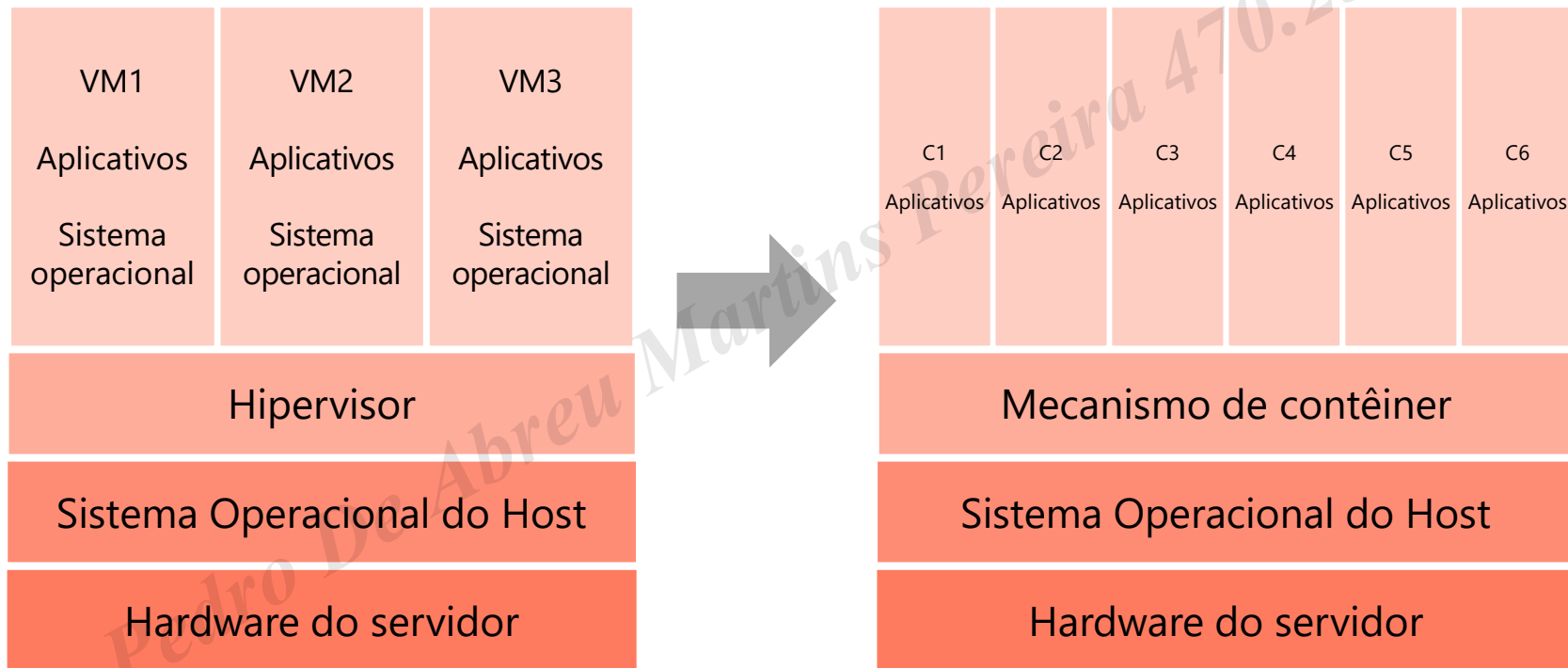
```
etapas:
  - shell: pwsh
  env:
    DB_PASSWORD: ${ secrets.DBPassword }
  run: |
    db_test "$env.DB_PASSWORD"
```

Estratégia de CI para Containers

Por que contêineres?

- 1 Portáteis** – Executam contêineres sempre que o Docker for compatível
- 2 Consistentes** – Garante que os desenvolvedores estão trabalhando com o mesmo código
- 3 Leves** – Usam muito menos disco, CPU e memória do que VMs
- 4 Eficientes** – Implantação, inicialização, aplicação de patches e atualizações rápidas

Examinar a estrutura de contêineres



Discussão: contêineres versus máquinas virtuais

Em seu ambiente de desenvolvimento:

- 1 Você usa atualmente algum tipo virtualização?
- 2 Você prefere implantar contêineres ou máquinas virtuais?

Trabalhar com contêineres do Docker

- 1 Docker build** – Cria a imagem

- 2 Docker pull** – Recupera a imagem de um registro de contêiner

- 3 Execução do Docker** – Executa a imagem para criar uma instância de contêiner (será extraída automaticamente se ainda não tiver sido)

Entender os principais conceitos do Dockerfile

```
FROM ubuntu
LABEL maintainer="johndoe@contoso.com"
ADD appsetup /
RUN /bin/bash -c 'source $HOME/.bashrc; \
echo $HOME'
CMD ["echo", "Hello World from within the container"]
```



Os Dockerfiles são arquivos de texto que contêm os comandos necessários para o **Docker build** montar uma imagem

Dockerfiles de vários estágios

Builds de vários estágios proporcionam os benefícios do padrão do construtor sem o incômodo de manter arquivos separados

O código progride de um estágio para o próximo

```
FROM mcr.microsoft.com/dotnet/core/aspnetcore:3.1
AS base
WORKDIR /app
EXPOSE 80
EXPOSE 443
```

```
FROM mcr.microsoft.com/dotnet/core/sdk:3.1 AS
build
WORKDIR /src
COPY ["WebApplication1.csproj", ""]
RUN dotnet restore "./WebApplication1.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "WebApplication1.csproj" -c
Release -o /app/build
```

```
FROM build AS publish
RUN dotnet publish "WebApplication1.csproj" -c
Release -o /app/publish
```

```
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "WebApplication1.dll"]
```

Considerações para builds de vários estágios

- 1 Adotar a modularidade de contêiner
- 2 Evitar pacotes desnecessários
- 3 Escolher uma base apropriada
- 4 Evitar incluir dados de aplicativos

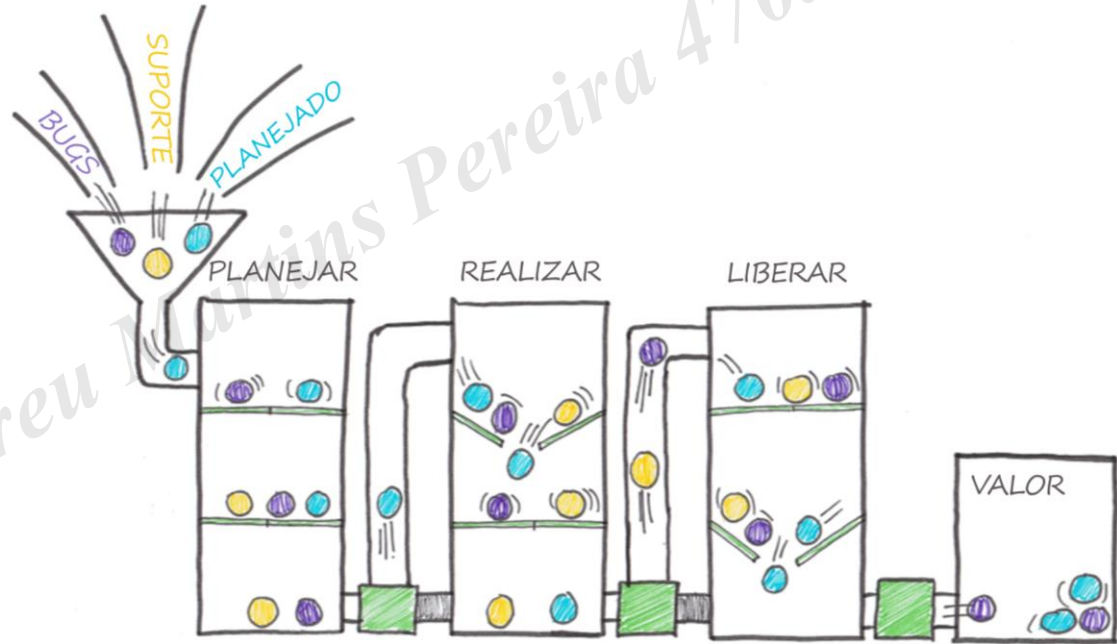
Explorar serviços relacionados ao contêiner do Azure

- 1 As Instâncias de Contêiner do Azure** permitem que você se concentre na criação de seus aplicativos em vez de provisionar e gerenciar a infraestrutura
- 2 O Serviço de Kubernetes do Azure** se tornou rapidamente o padrão para orquestração de contêineres
- 3 O Registro de Contêiner do Azure** permite armazenar e gerenciar suas imagens de contêiner em um registro central
- 4 Os Aplicativos de Contêiner do Azure** permitem criar e implantar microsserviços e aplicativos modernos usando contêineres sem servidor
- 5 O Serviço de Aplicativo do Azure** fornece um serviço gerenciado para aplicativos Web baseados em Windows e Linux

Introdução a Entrega Contínua

Explorar o tradicional ciclo de desenvolvimento de TI

- Um bom número de hotfixes e solicitações de alteração para produção
- Muitas alterações de escopo durante um projeto
- Muitos trabalhos não planejados devido a dívidas técnicas (descompasso de ambiente, qualidade inadequada e entregas)
- Negócios envolvidos, mas não associado à TI

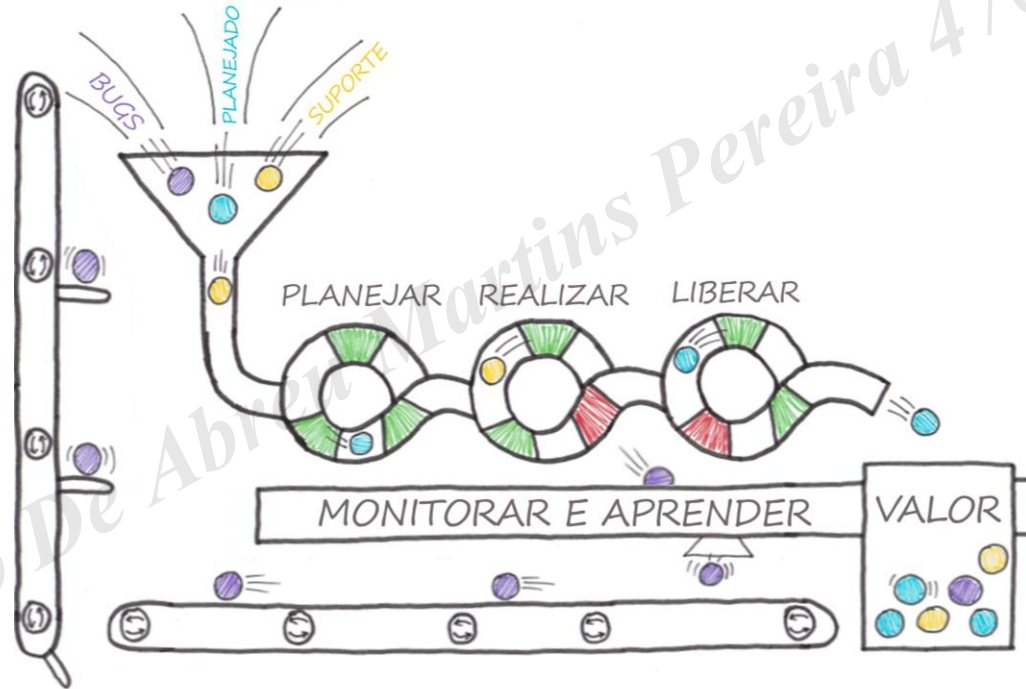


O que é a entrega contínua?

Os oito princípios da entrega contínua:

- O processo de liberação/implantação de software DEVE ser confiável e repetível.
- Manter tudo sob o controle de código-fonte
- Todos têm responsabilidade pelo processo de versão
- Automatização completa.
- Pronto significa "lançado"
- Melhorar continuamente
- Se algo for difícil ou custoso, faça com mais frequência
- Alta qualidade de build.

Migrar para a entrega contínua



Noções básicas sobre versões e implantações



A versão e a implantação geralmente estão associadas



Versão não é o mesmo que implantação



Separe a versão funcional da versão técnica:

A versão funcional expõe recursos aos clientes

A versão técnica implanta a funcionalidade

Entender o processo de versão em relação à versão



O processo de versão envolve todas as etapas pelas quais você passa ao mover seu artefato que vem de uma das fontes de artefato.



Uma versão é um pacote ou contêiner que acomoda um conjunto de artefatos especificado em um pipeline de lançamento no processo de CI/CD. Dessa forma, contém todas as informações necessárias para executar todas as tarefas e ações no pipeline de versão, como os estágios (ou ambientes), as tarefas de cada um, os valores dos parâmetros e variáveis da tarefa, as políticas de versão, como gatilhos, aprovadores e opções de enfileiramento de versão.



Pode haver várias versões em um pipeline de liberação (ou processo de lançamento).

Discussão – A necessidade de entrega contínua na organização

- A organização precisa da entrega contínua?
- Você usa o agile/scrum?
 - Todos estão envolvidos ou apenas os departamentos de desenvolvimento?
- É possível implantar seu aplicativo várias vezes por dia? Por quê?
- Qual é o principal gargalo da entrega contínua na organização?
 - A organização
 - Arquitetura de aplicativo
 - Habilidades
 - Ferramentas
 - Testes
 - Algo mais?

Release Pipelines

Descrever as funcionalidades do pipeline de lançamento do Azure DevOps

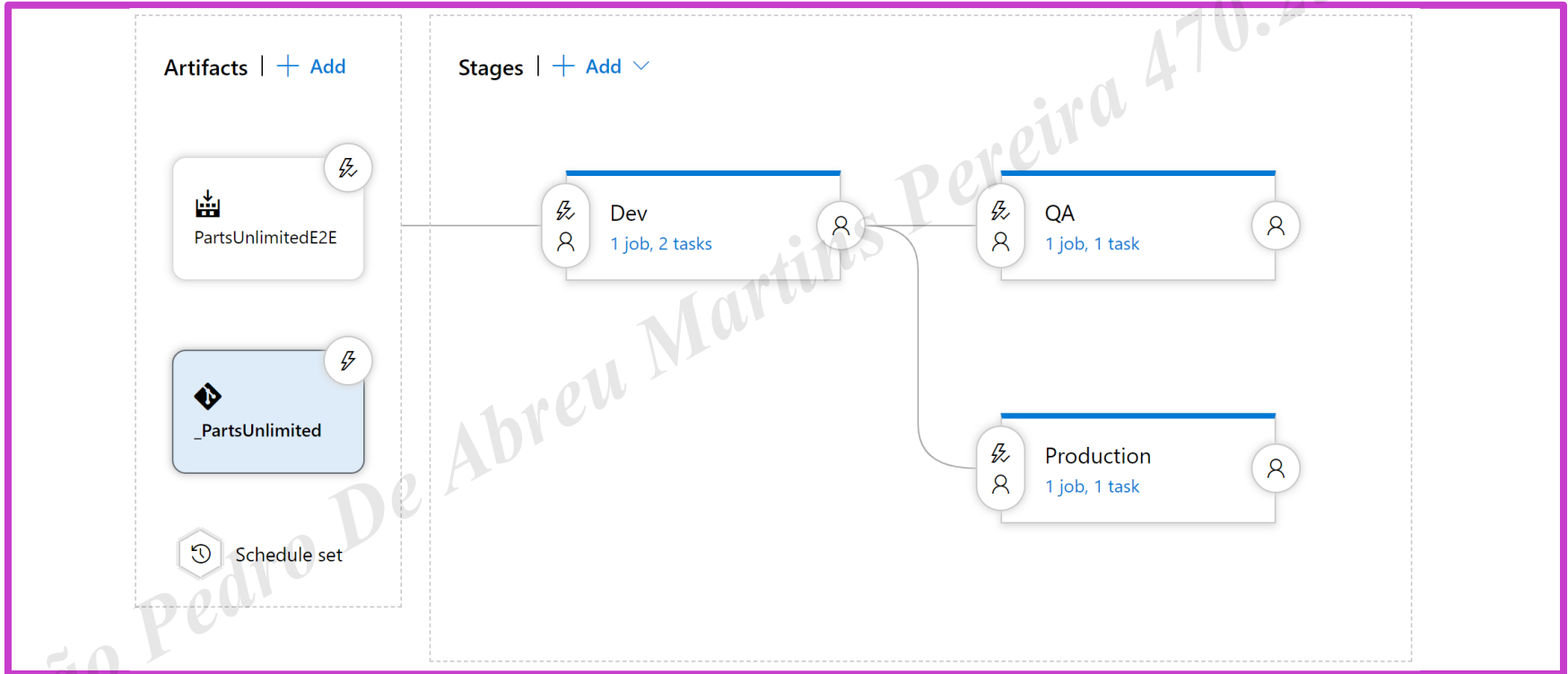


Suporte a pipelines como código (por meio de YAML)



A maioria dos recursos dos pipelines de versão clássica está disponível

Explorar pipelines de lançamento



Explorar fontes de artefatos



Artefatos de compilação



Repositórios de pacotes



Repositórios de contêineres



Controle do código-fonte

Escolher a fonte de artefato apropriada



Rastreabilidade e capacidade de auditoria



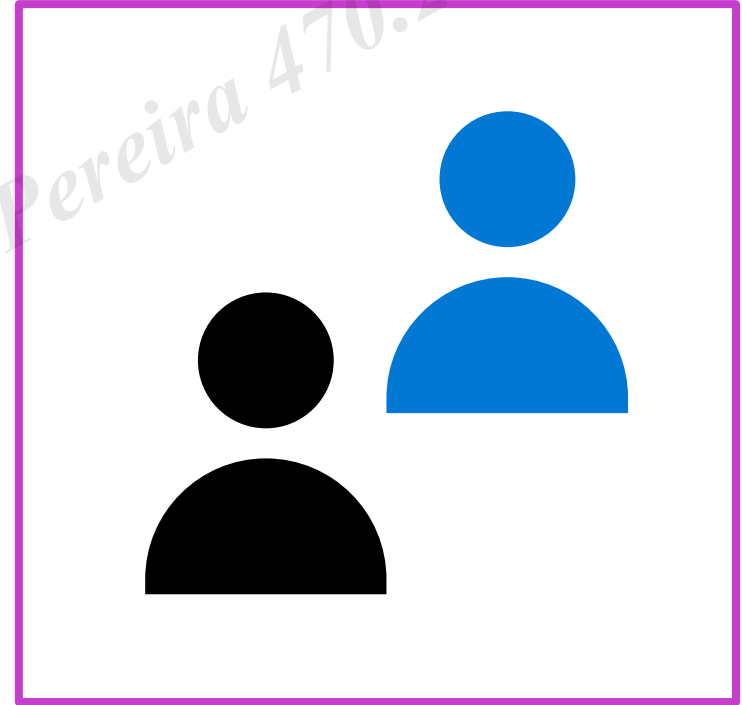
Imutabilidade



Controle de versão

Examinar as considerações para uma implantação em fases

- Queremos/precisamos implantar todos os dias?
- Qual é o ambiente de destino?
- É usado por uma equipe ou é usado por várias equipes?
- Quem são os usuários? Eles querem uma nova versão várias vezes por dia?
- Quanto tempo leva para implantá-la?
- Há algum tempo de inatividade? O que acontece com o desempenho? Usuários foram impactados?



Estágios de implantação

1 O que é um estágio?

Diferentes termos usados em várias ferramentas de versão (estágios/ambiente)
Um limite lógico em um pipeline que contém um ou mais trabalhos

2 Qual é o ambiente de destino?

3 Ambientes de vida longa ou curta


4 Finalidade de um ambiente

5 Lançamento de versão e correção de bugs

Explorar tarefas de compilação e versão


- Unidades de código executável usadas para executar ações designadas em uma ordem especificada
- Etapas para compilar, testar, executar utilitários, empacotar e implantar
- Modelo extensível
- Tarefas da comunidade disponíveis no marketplace

Add tasks

 Refresh

 Search


All Build Utility Test Package Deploy Tool Marketplace


 **Download Secure File**
Download a secure file to a temporary location on the build or release agent


 **Extract Files**
Extract a variety of archive and compression files such as .7z, .rar, .tar.gz, and .zip.

 **FTP Upload**
FTP Upload

 **GitHub Release**
Create, edit, or delete a GitHub release.

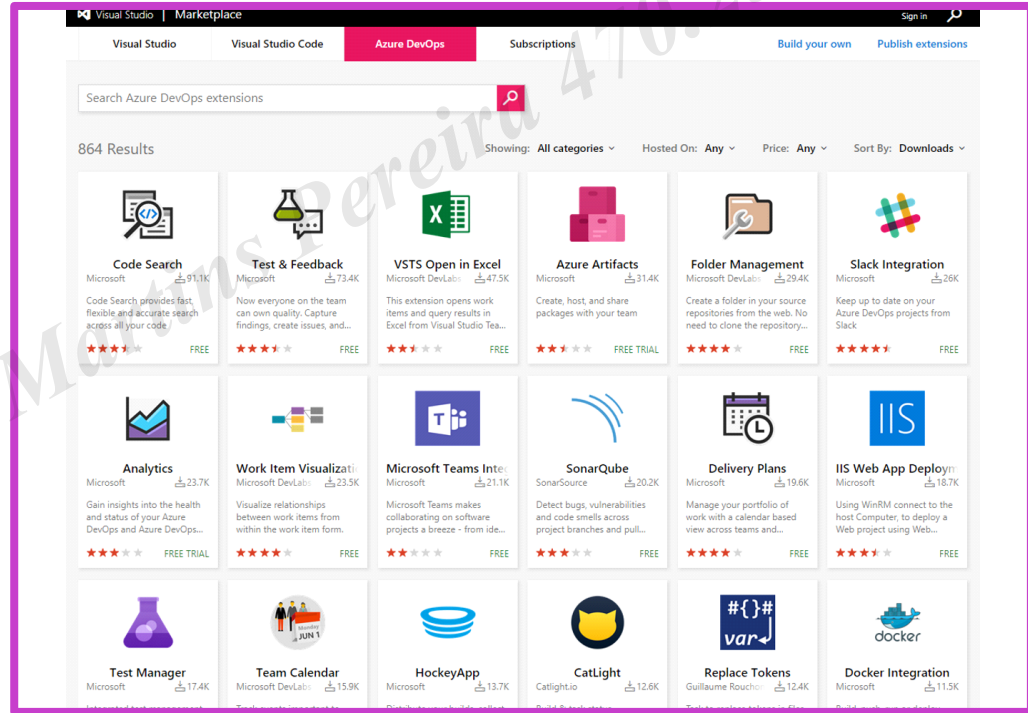
 **Install Apple Certificate**
Install an Apple certificate required to build on a macOS agent

 **Install Apple Provisioning Profile**
Install an Apple provisioning profile required to build on a macOS agent

 **Install SSH Key**
Install an SSH key prior to a build or release

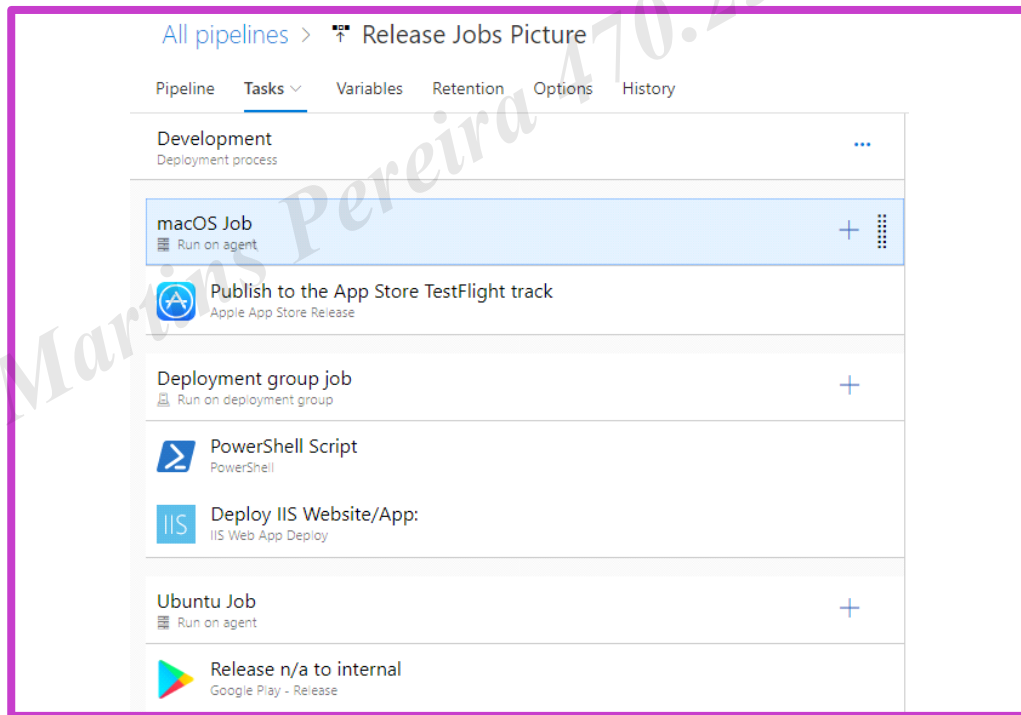
Explorar tarefas personalizadas de build e lançamento

- Privado ou público acessível
- Acesso a variáveis que, de outra forma, não são acessíveis.
- Usar e reutilizar um ponto de extremidade seguro para um servidor de destino
- Distribuição com segurança e eficiência em toda a organização
- Os usuários não veem os detalhes da implementação



Explorar os trabalhos de lançamento

- Um trabalho é uma série de tarefas que são executadas sequencialmente no mesmo destino
- **Pode ser combinado em um pipeline para permitir a implantação multiplataforma:**
 - Por exemplo, implantar back-end .NET via Windows, aplicativo iOS via MacOS e frontend Angular via Linux
- Os trabalhos são executados no computador host em que o agente está instalado



Discussão: como usar trabalhos de versão

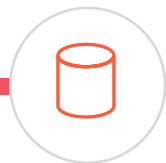
Você vê uma finalidade para os trabalhos de versão em seu pipeline? Como você os configuraria?

Os tópicos que são interessantes para você considerar são:

- Você tem artefatos de várias fontes?
- Deseja executar implantações em servidores diferentes simultaneamente?
- Você precisa de várias plataformas?
- Quanto tempo seu lançamento leva?
- É possível executar a implantação em paralelo ou ela precisa ser executada em sequência?

Estratégias de Versão

Noções básicas sobre a cadência de entrega e os três tipos de gatilhos



Gatilho de implantação
contínua





Gatilho
agendado




Gatilho
manual

Explorar aprovações de versão

 As aprovações de versão não controlam **como**, mas controlam **se** você deseja fazer entregas várias vezes por dia

 As aprovações manuais ajudam a criar confiança sobre o processo de versão automatizada

 Os portões de versão oferecem controle adicional sobre o início e a conclusão do pipeline de implantação. Eles geralmente podem ser configurados como uma condição de pré-implantação e pós-implantação e podem executar a validação com outros sistemas automatizados até que requisitos específicos sejam verificados.

Explorar os portões de versão

Os portões de versão oferecem controle adicional sobre o início e a conclusão do pipeline de implantação. Geralmente, eles são configurados como condições de pré-implantação e pós-implantação.

- 1 Gerenciamento de incidentes e problemas
- 2 Notificação de usuários por integração com sistemas de colaboração
- 3 Validação de qualidade
- 4 Verificação de segurança em artefatos
- 5 Experiência do usuário relativa à linha de base
- 6 Gerenciamento de mudanças
- 7 Integridade da infraestrutura

Usar portões de versão para proteger a qualidade

- Sem novos problemas de bloqueador
- Cobertura de código no novo código maior que 80%
- Sem violações de licença
- Nenhuma vulnerabilidade em dependências
- Nenhuma nova dívida técnica introduzida
- Verificações de conformidade
- Há itens de trabalho vinculados à versão?
- A versão foi iniciada por uma pessoa diferente daquela que fez commit do código?
- O desempenho não é afetado após uma nova versão?

YAML Multi Stage pipelines

Descrever estratégias de trabalhos de implantação



Habilitar a inicialização



Implantar a atualização



Encaminhar o tráfego para a versão atualizada



Testar a versão atualizada após o roteamento do tráfego



Se houver uma falha, execute as etapas para restaurar a última versão boa conhecida

Descrever ganchos do ciclo de vida



preDeploy – usado para executar etapas que inicializam recursos antes do início da implantação do aplicativo.



deploy – usado para executar etapas que implantam seu aplicativo.



routeTraffic – usado para executar etapas que servem o tráfego para a versão atualizada.



postRouteTraffic – usado para executar as etapas depois que o tráfego é roteado.



on: failure ou on: success – usado para executar etapas para reverter ações ou limpar.

Descrever estratégias de trabalhos de implantação – RunOnce

runOnce é a estratégia de implantação mais simples em que todos os ganchos de ciclo de vida são executados:

- preDeploy
- deploy
- routeTraffic
- postRouteTraffic

Então, **on:success** ou **on: failure** é executado.

```
strategy:
  runOnce:
    preDeploy:
      pool: [ server | pool ] # See pool schema.
      etapas:
        script: [ script | bash | pwsh | powershell | checkout
                  | task | templateReference ]
    deploy:
      pool: [ server | pool ]
      etapas:
        . . .
    routeTraffic:
      pool: [ server | pool ]
      etapas:
        . . .
    postRouteTraffic:
      pool: [ server | pool ]
      etapas:
        . . .
    em:
      failure:
        pool: [ server | pool ]
      etapas:
        . . .
    success:
      pool: [ server | pool ]
      etapas:
        . . .
```

Descrever estratégias de trabalhos de implantação – rolling

- Uma implantação em andamento substitui instâncias da versão prévia de um aplicativo com instâncias da nova versão.
- Ela pode ser configurada especificando a palavra-chave em andamento na estratégia: nó.

```
strategy:
  rolling:
    maxParallel: [ number or percentage as x% ]
    preDeploy:
      steps: [ script | bash | pwsh | powershell |
checkout | task | templateReference ]
    deploy:
      steps:
        . . .
    routeTraffic:
      steps:
        . . .
    postRouteTraffic:
      steps:
        . . .
    on:
      failure:
        steps:
          . . .
      success:
        steps:
          . . .
```

Descrever estratégias de trabalhos de implantação – Canary

- Estratégia de implantação avançada que ajuda a atenuar o risco envolvido na implantação de novas versões de aplicativos.
- Distribui as alterações para um pequeno subconjunto de servidores primeiro.
- Lança em mais servidores em sua infraestrutura e roteia mais tráfego para ele.

```
strategy:
  canary:
    increments: [ number ]
    preDeploy:
      pool: [ server | pool ] # See pool schema.
      steps:
        - script: [ script | bash | pwsh | powershell |
checkout | task | templateReference ]
    deploy:
      pool: [ server | pool ]
      steps:
        . . .
    routeTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    postRouteTraffic:
      pool: [ server | pool ]
      steps:
        . . .
    on:
      failure:
        pool: [ server | pool ]
      steps:
        . . .
      success:
        pool: [ server | pool ]
      steps:
        . . .
```

Descrever estratégias de trabalhos de implantação – exemplo de Canary para AKS

No próximo exemplo, a estratégia canário do AKS implantará primeiro as alterações com pods de 10%, seguidos por 20%, enquanto monitora a integridade durante o postRouteTraffic. Se tudo correr bem, promoverá 100%.

```
jobs:
- deployment:
environment: smarthotel-dev.bookings
pool:
  name: smarthotel-devPool
strategy:
  canary:
    increments: [ 10,20 ]
  preDeploy
  etapas:
    - script: initialize, cleanup...
    deploy:
  etapas:
    - script: echo deploy updates...
    - task: KubernetesManifest@0
      entradas:
        actions: $(strategy.action)
        namespace: 'default'
        strategy: $(strategy.name)
        percentage: $(strategy.increment)
        manifest: 'manifest.yml'
  postRouteTraffic:
  pool: server
  etapas:
    script: echo monitor application health...
  em:
  failure:
    etapas:
      - script: echo clean-up, rollback...
  success:
    etapas:
      - script: echo checks passed, notify...
```

Referências de livros

- The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win
- Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations
- DevOps Handbook: How to Create World-Class Agility, Reliability, & Security in Technology Organizations
- Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation
- Lean Software Development: An Agile Toolkit

Referências de sites e recursos on line

- <https://www.devopsinstitute.com/blog/>
- <https://git-scm.com/>
- <https://learn.microsoft.com/en-us/azure/devops/?view=azure-devops>
- <https://docs.github.com/pt>
- <https://azuredevopslabs.com/>

OBRIGADO!

[linkedin.com/in/rogeriorodrigues](https://www.linkedin.com/in/rogeriorodrigues)