Microsoft

AZ-400.1 Module 1: Getting Started with Source Control

Foundational Practices of DevOps

Foundational practices and the 5 stages of DevOps evolution

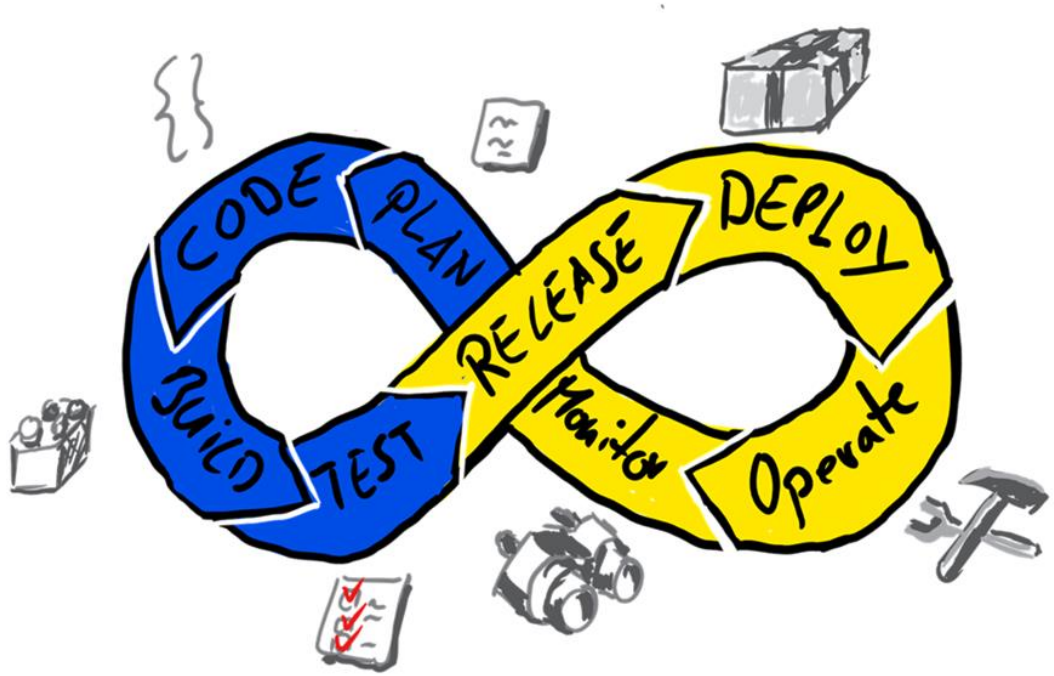
	Defining practices* and associated practices	Practices that contribute to success
Stage 0	• Monitoring and alerting are configurable by the team operating the service. • Deployment patterns for building applications or services are reused. • Testing patterns for building applications or services are reused. • Teams contribute improvements to tooling provided by other teams. • Configurations are managed by a configuration management tool.	
Stage 1	• **Application development teams use version control.** • **Teams deploy on a standard set of operating systems.**	• Build on a standard set of technology. • Put application configurations in version control. • Test infrastructure changes before deploying to production. • Source code is available to other teams.
Stage 2	• **Build on a standard set of technology.** • **Teams deploy on a single standard operating system.**	• Deployment patterns for building applications and services are reused. • Rearchitect applications based on business needs. • Put system configurations in version control.
Stage 3	• **Individuals can do work without manual approval from outside the team.** • **Deployment patterns for building applications and services are reused.** • Infrastructure changes are tested before deploying to production.	• Individuals can make changes without significant wait times. • Service changes can be made during business hours. • Post-incident reviews occur and results are shared. • Teams build on a standard set of technologies. • Teams use continuous integration. • Infrastructure teams use version control.
Stage 4	• **System configurations are automated.** • **Provisioning is automated.** • Application configurations are in version control. • Infrastructure teams use version control.	• Security policy configurations are automated. • Resources made available via self-service.
Stage 5	• **Incident responses are automated.** • **Resources available via self-service.** • Rearchitect applications based on business needs. • Security teams are involved in technology design and deployment.	• Security policy configurations are automated. • Application developers deploy testing environments on their own. • Success metrics for projects are visible. • Provisioning is automated.

*** The practices that define each stage are highlighted in bold font.**

Lesson 01: What is Source Control?

Introduction to Source Control

- DevOps is a revolutionary way to release software quickly and efficiently while maintaining a high level of security
- Source control (version control) is a critical part of DevOps



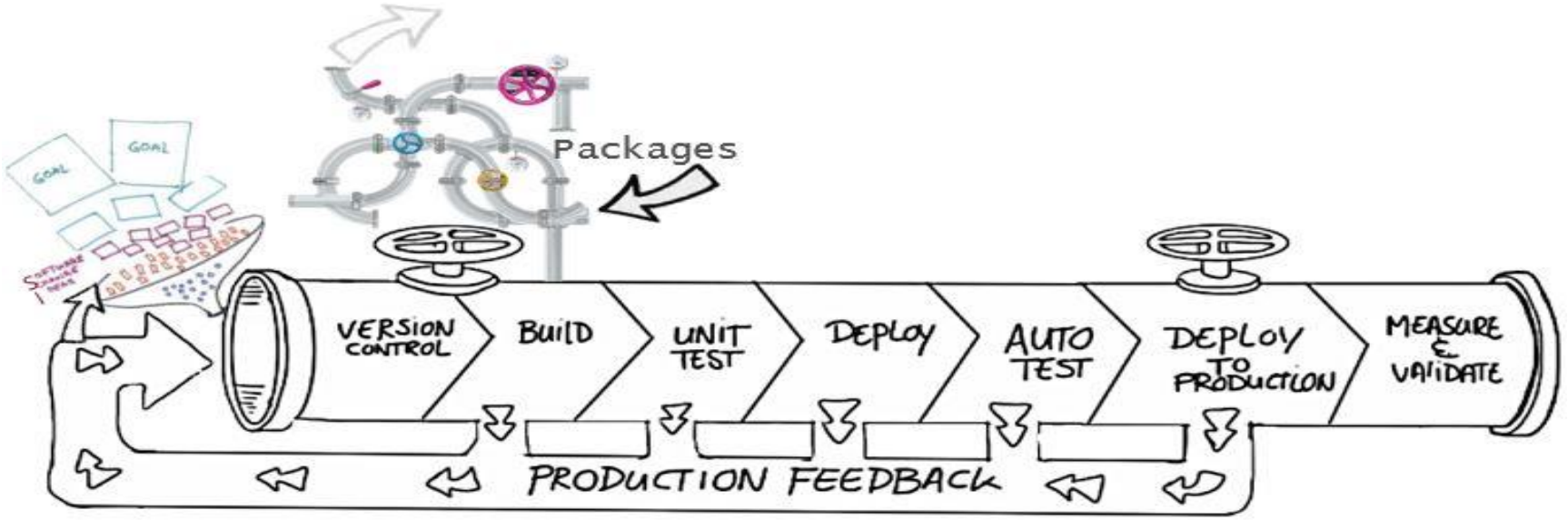
What is Source Control?

- The practice of tracking and managing changes to code
- Provides a running history of code development
- Helps to resolve conflicts when merging contributions from multiple sources
- Protects source code from both catastrophe and human error
- Benefits include: reusability, traceability, manageability, efficiency, collaboration, and learning.

Lesson 02: Benefits of Source Control

Benefits of Source Control

- Create workflows
- Work with versions
- Collaboration
- Maintains history of changes
- Automate tasks



Best Practices for Source Control

- Make small changes
- Don’t commit personal files
- Update often and right before pushing to avoid merge conflicts
- Verify your code change before pushing it to a repository; ensure it compiles and tests are passing.
- Pay close attention to commit messages as these will tell you why a change was made
- Link code changes to work items
- No matter your background or preferences, be a team player and follow agreed conventions and workflows

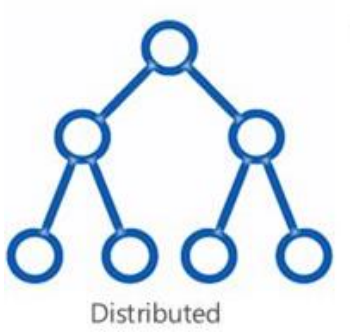
Lesson 03: Types of Source Control Systems

Centralized Source Control

Strengths	Best Used for
• Easily scales for very large codebases • Granular permission control • Permits monitoring of usage • Allows exclusive file locking	• Large integrated codebases • Audit and access control down to the file level • Hard to merge file types

- There is a single central copy of your project and programmers commit their changes to this central copy
- Common centralized version control systems are TFVC, CVS, Subversion (or SVN) and Perforce

Distributed Source Control

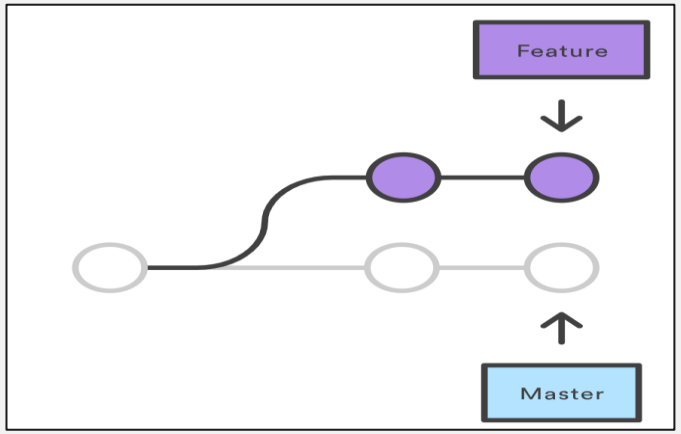


Strengths	Best Used for
• Cross platform support • An open source friendly code review model via pull requests • Complete offline support • Portable history • An enthusiastic growing user based	• Small and modular codebases • Evolving through open source • Highly distributed teams • Teams working across platforms • Greenfield codebases

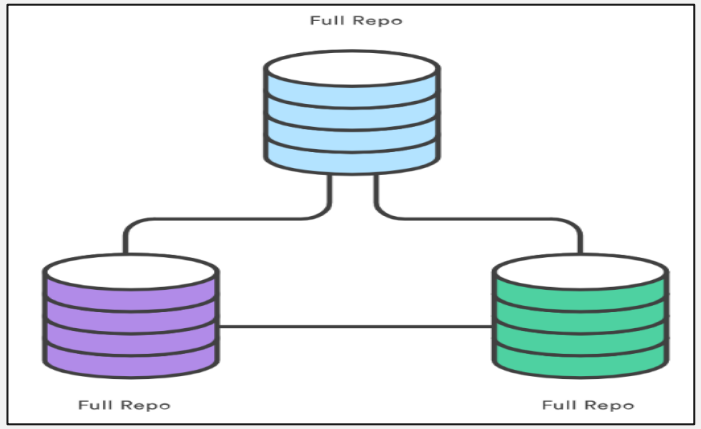
- Every developer clones a copy of a repository and has the full history of the project
- Common distributed source control systems are Mercurial, Git and Bazaar.

Why Git?

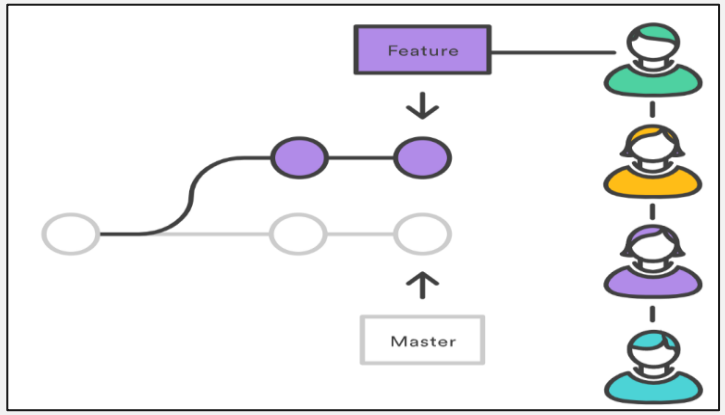
Feature branches



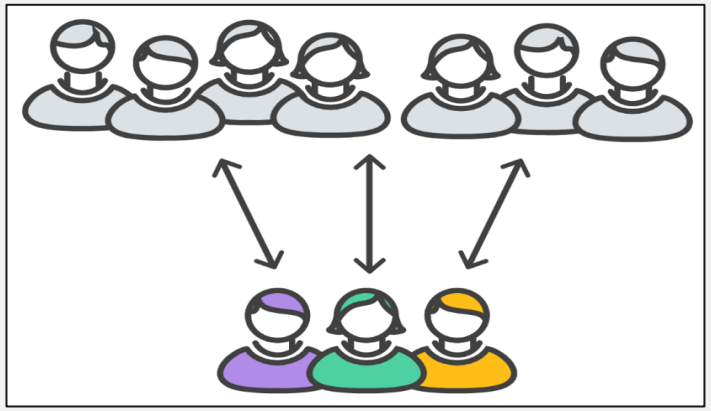
Distributed development



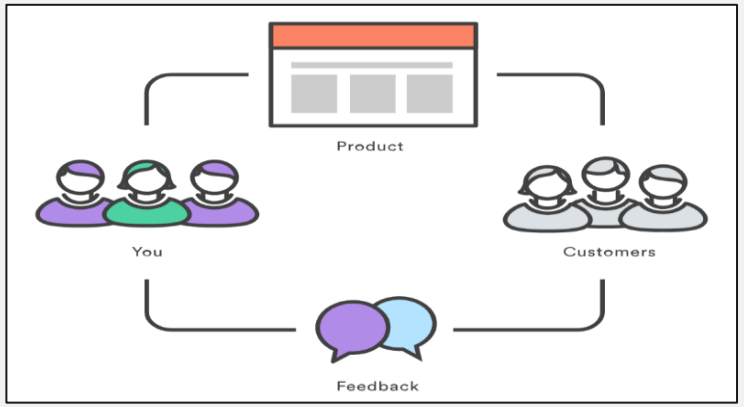
Pull requests



Community

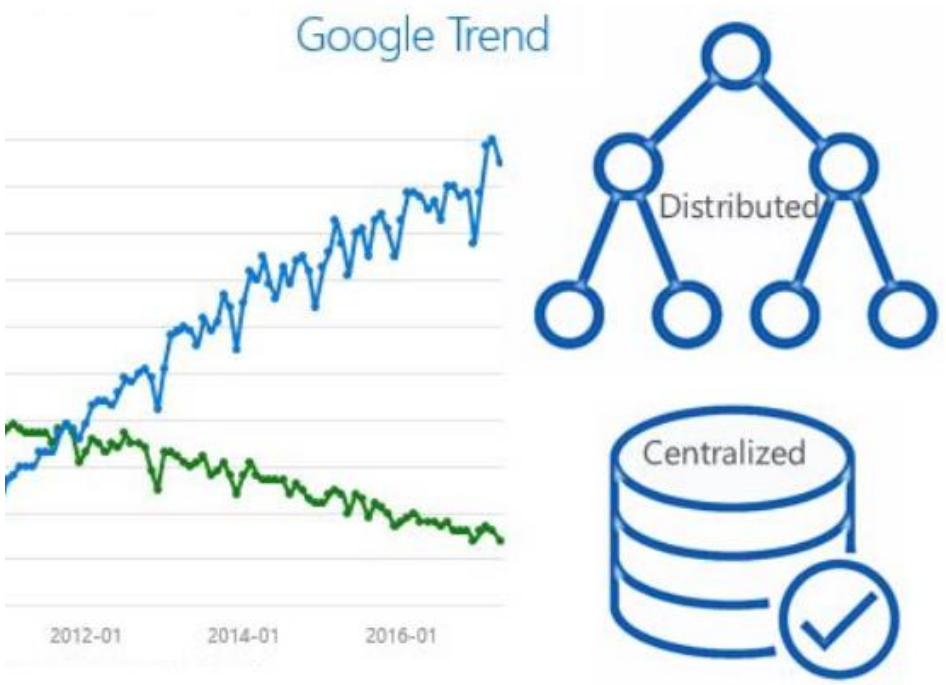


Release cycles



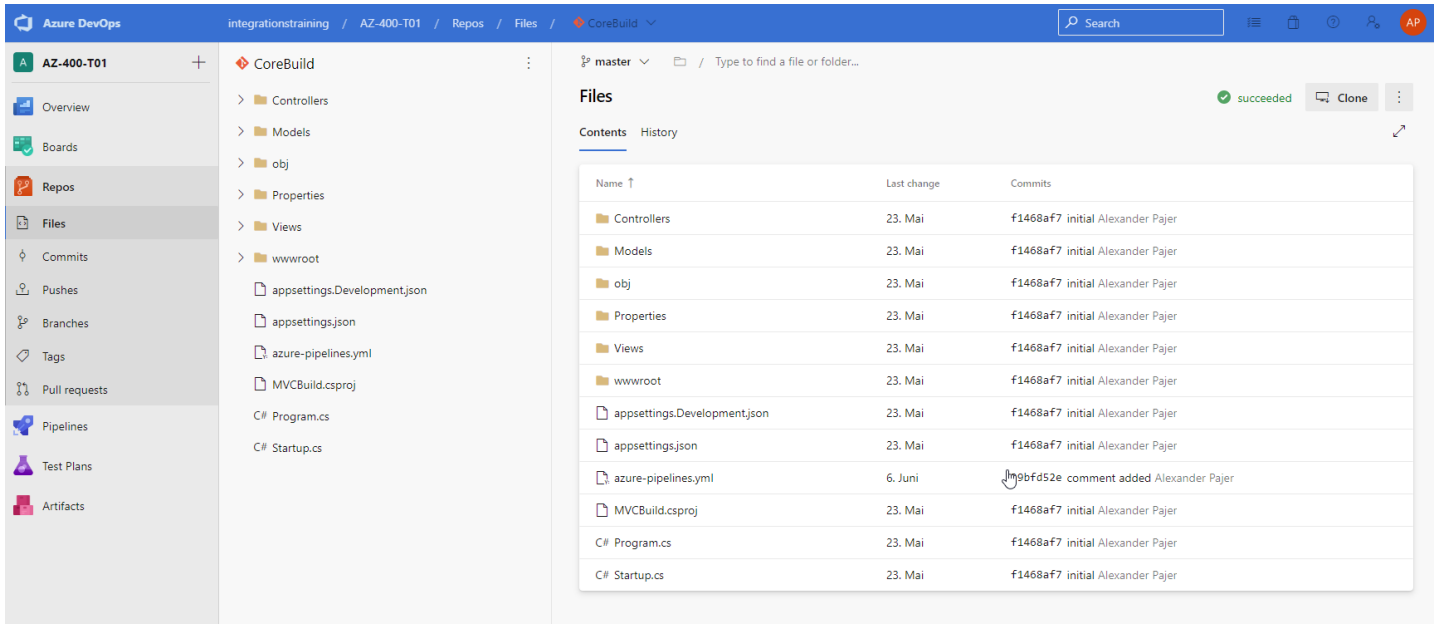
Git and TFVC

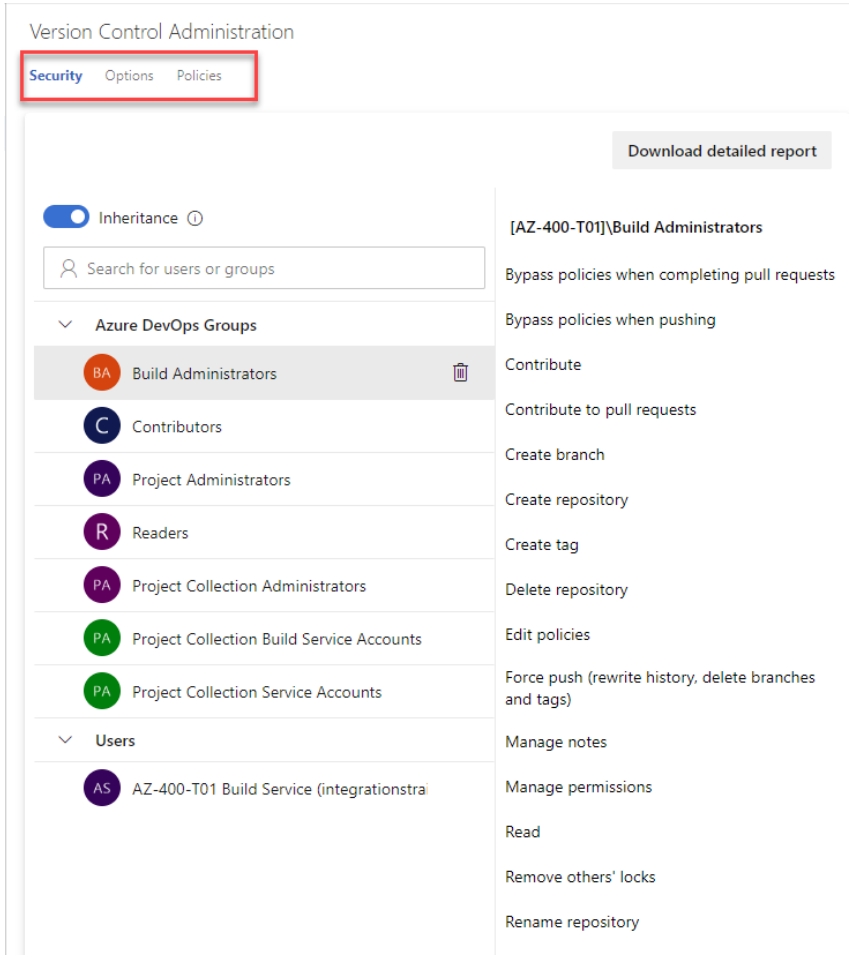
- Git
  - Distributed source control system
  - Each developer has a copy of the source repository on their dev machine
- TFVC
  - Centralized source control system
  - Team members have only one version of each file on their dev machines
  - In the *Server workspaces* model, before making changes, team members publicly check out files
  - In the *Local workspaces* model, each team member takes a copy of the latest version of the codebase with them and works offline as needed



Azure Repos

- Git Repos hosted in Azure DevOps
- Each Project can host multible Repos





Working with Git Locally

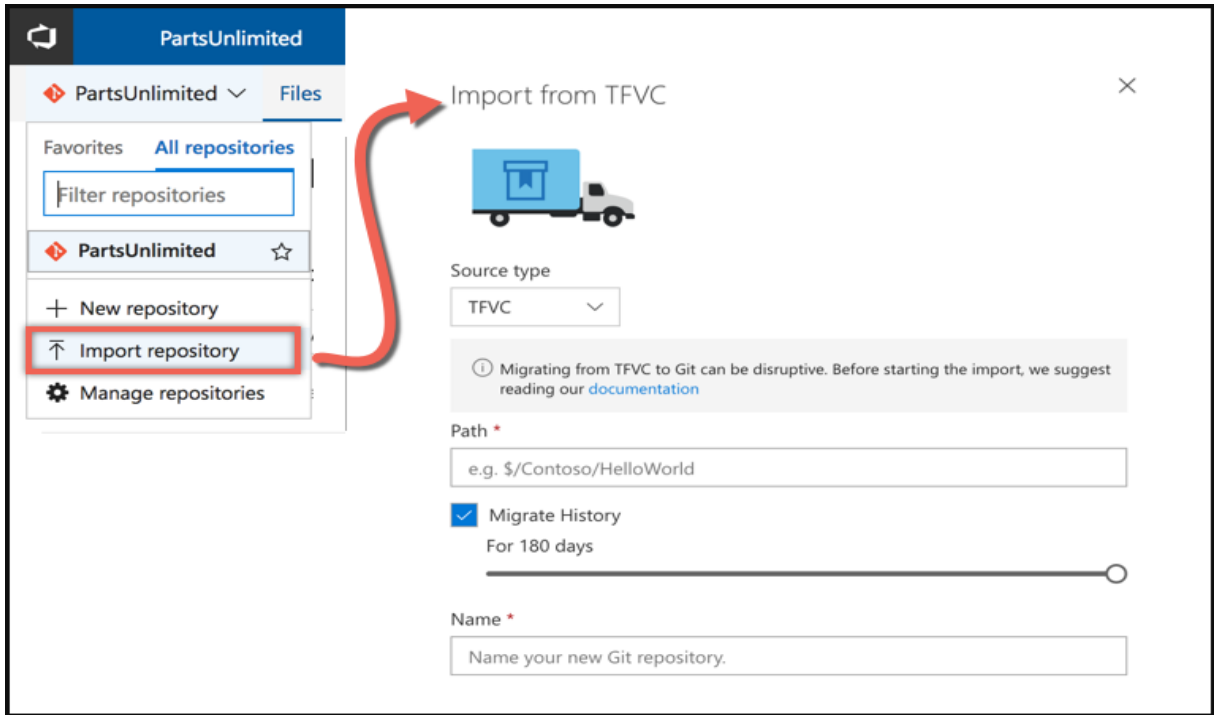
Lesson 05: Migrating from TFVC to Git

Migrating from TFVC to Git

- Migrating the tip
  - Only the latest version of the code
  - History remains on the old server
- Migrating with history
  - Tries to mimic the history in git
- Recommend to migrate the tip, because:
  - History is stored differently – TFVC stores change sets, Git stores snapshots of the repository
  - Branches are stored differently – TFVC branches folders, Git branches the entire repository

Migrating from TFVC to Git

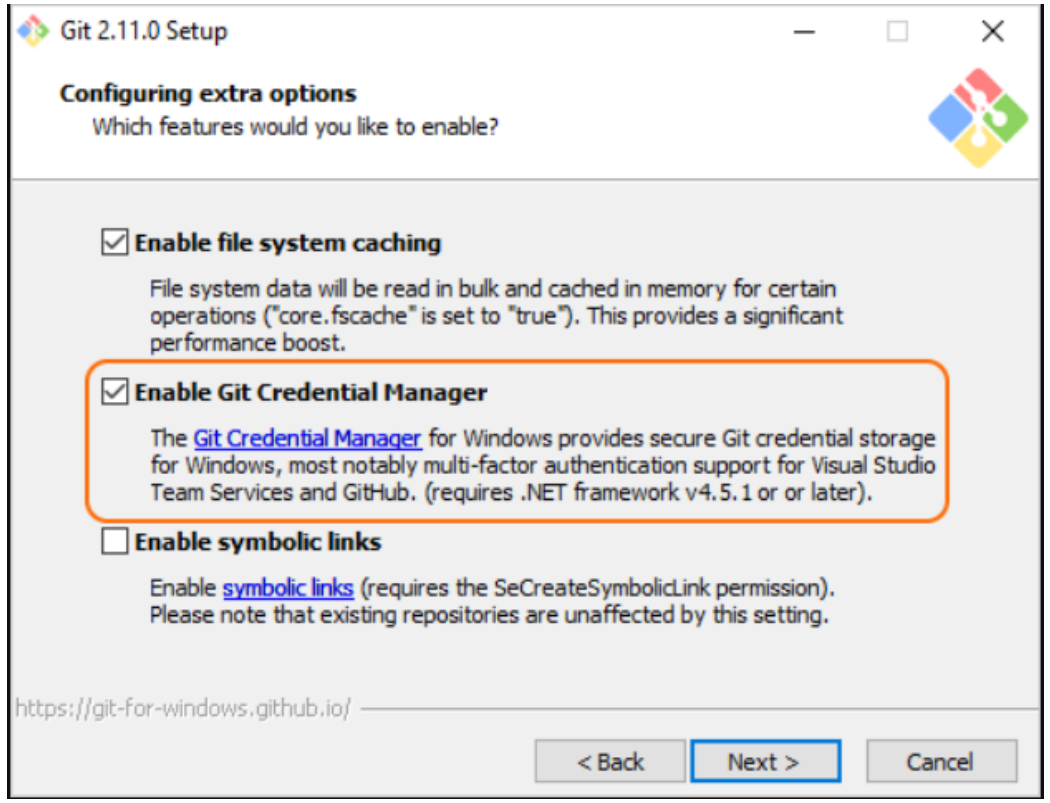
- Single branch import
- Full synchronization
  - Done using Git-tfs
  - A two way bridge between TFS und Git
  - https://github.com/git-tfs/git-tfs



Lesson 06: Authenticating to Your Git Repos

Authenticating to Your Git Repos

- Git Credential Manager simplifies authentication with your Azure DevOps Services/TFS Git repos
- Supports MFA and two-factor authentication with GitHub repositories
- You can also configure your IDE with a Personal Access Token or SSH to connect with your repos



Git Credentials Management

- List Git Config
  - git config --list
- Set User and E-Mail
  - git config --global user.name "Your Name"
  - git config --global user.email "your.email@yourdomain.com
- Unset Credentials
  - git config --global --unset credential.helper

Microsoft

AZ-400.1 Module 2: Scaling Git for Enterprise DevOps

Lesson 01: How to Structure Your Git Repo

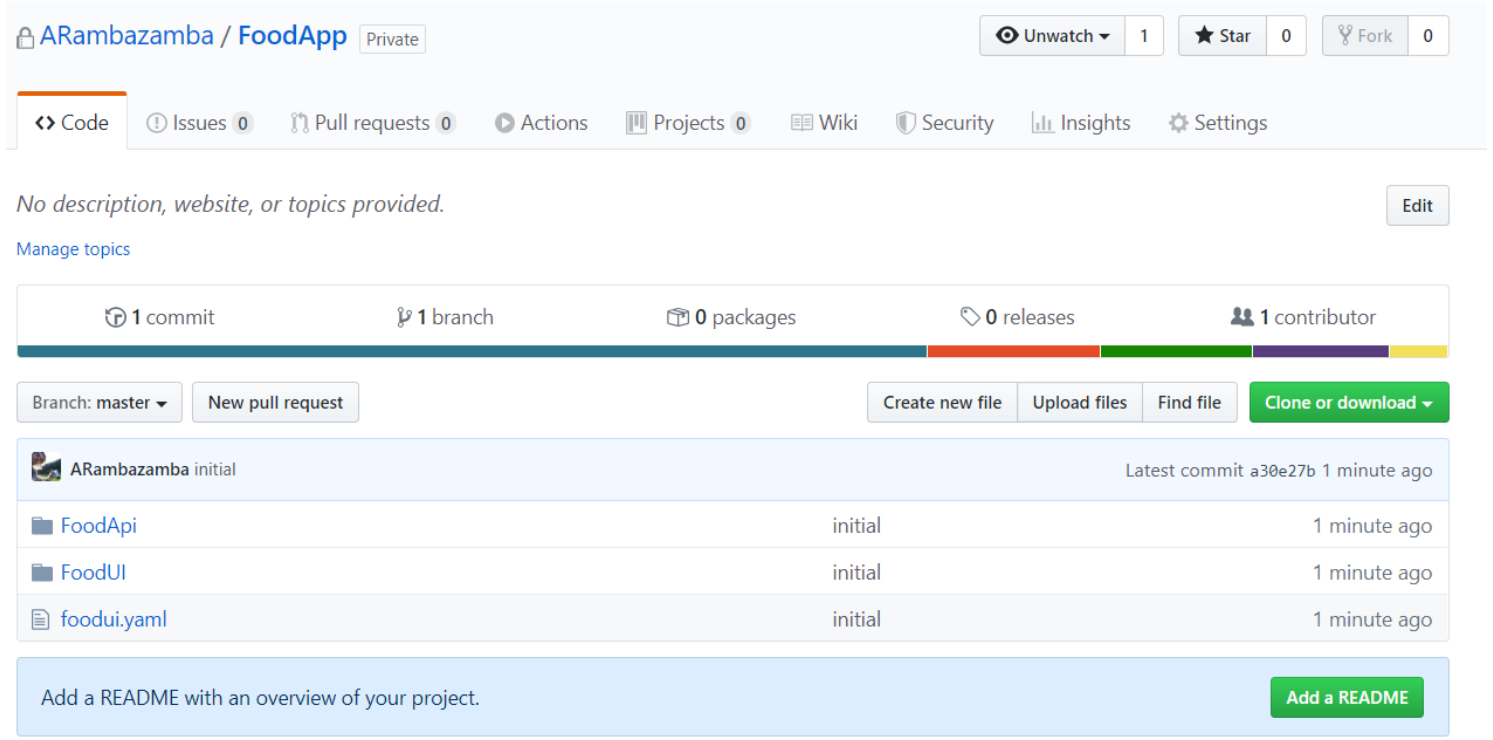
Mono vs Multi Repos

A repository is a place where the history of your work is stored

	Advantages
Mono-repo - source code is kept in a single repository	• Clear ownership • Better scale • Narrow clones
Multiple-repo – each project has its own repository	• Better developer testing • Reduced code complexity • Effective code reviews • Sharing of common components • Easy refactoring

FoodApp - Monorepo Sample

- .NET Core WebApi
- Angular Frontend



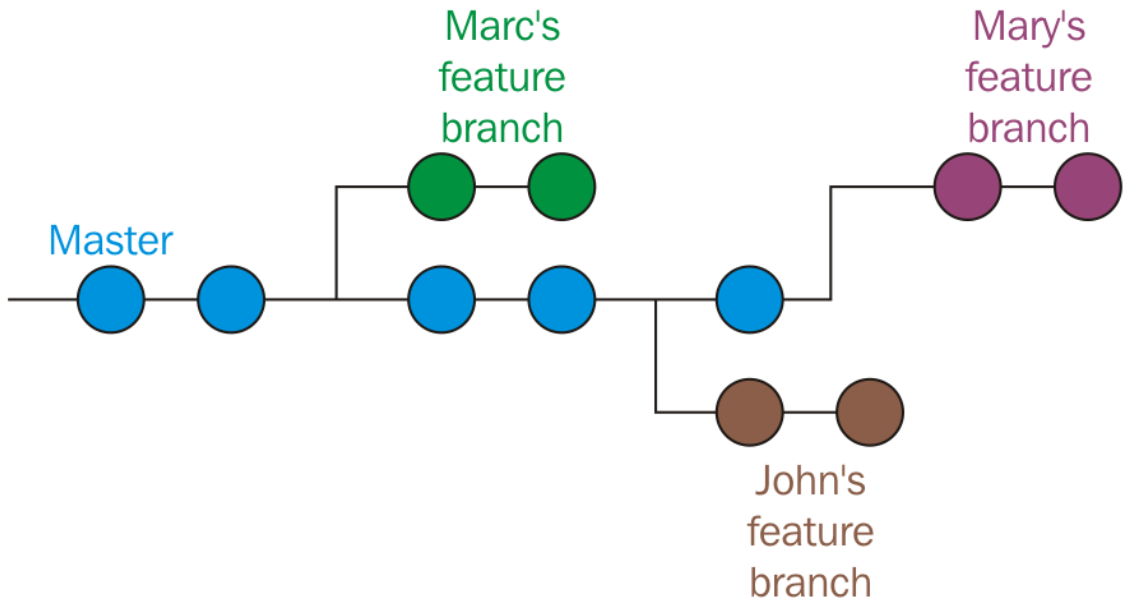
Lesson 02: Git Branching Workflows

Branching Workflow Types

- Feature branching
  - All feature development should take place in a dedicated branch instead of the master branch
- Gitflow branching
  - A strict branching model designed around the project release
- Forking Workflow
  - Every developer uses a server-side repository

Feature Branch Workflow

- All feature development should take place in a dedicated branch instead of the master branch
- Encapsulating feature development leverages pull requests, which are a way to initiate discussions around a branch
- Share a feature with others without touching any official code



GitFlow Branch Workflow

- GitFlow is great for a release-based software workflow
  - Wrapper around GIT
- GitFlow offers a dedicated channel for hotfixes to production
- Requires Git Bash >= 2.23

Basic Git Flow Commands

- Init Repo
  - git flow init
- Start / Finish a Feature
  - git flow feature start | finish
- Push a Branch to server
  - git push --set-upstream origin develop
- Releases
  - git flow release start | finish

Forking Branch Workflow

- Started with a Fork (Copy) of a server side Repo (typically addressed as Upstream)
  - Forking Branch workflow gives every developer their own server-side repository
- Most often seen in public open source projects
- Contributions are integrated
  - To the Fork of the Developer
  - To the Upstream using a Pull Request
- Typically follows a branching model based on the GitFlow Workflow

Git Branching model for continuous delivery

Lesson 03: Why Care about GitHooks

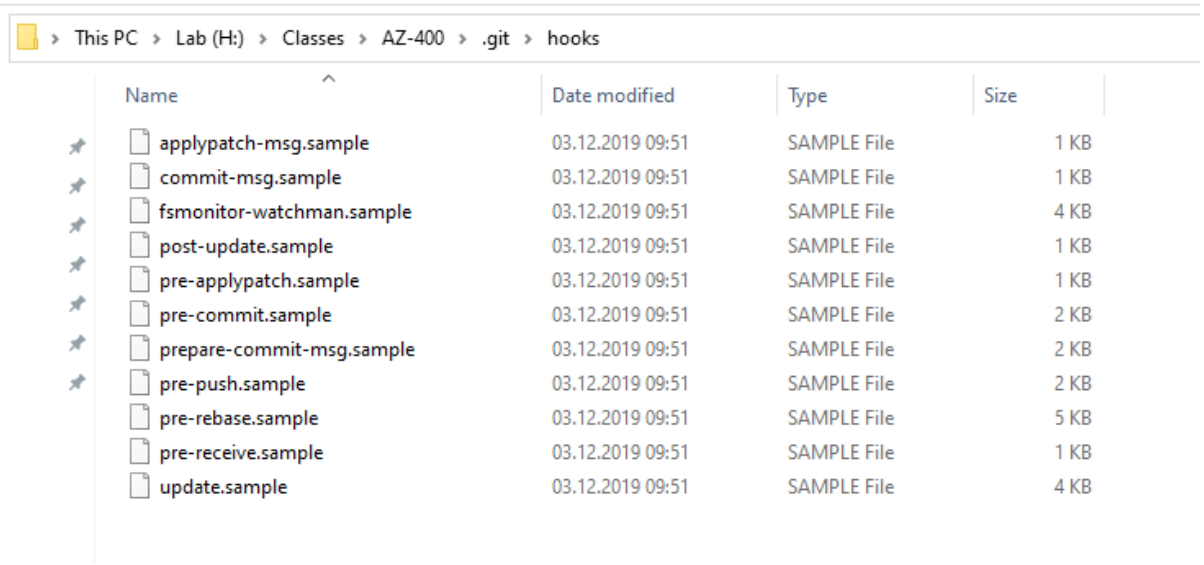
Why Care about GitHooks?

- A mechanism that allows arbitrary code to be run before, or after, certain Git lifecycle events occur
- Use GitHooks to enforce policies, ensure consistency, and control your environment
- Can be either client-side or server-side

✔ Using GitHooks is like having little robot minions to carry out your every wish

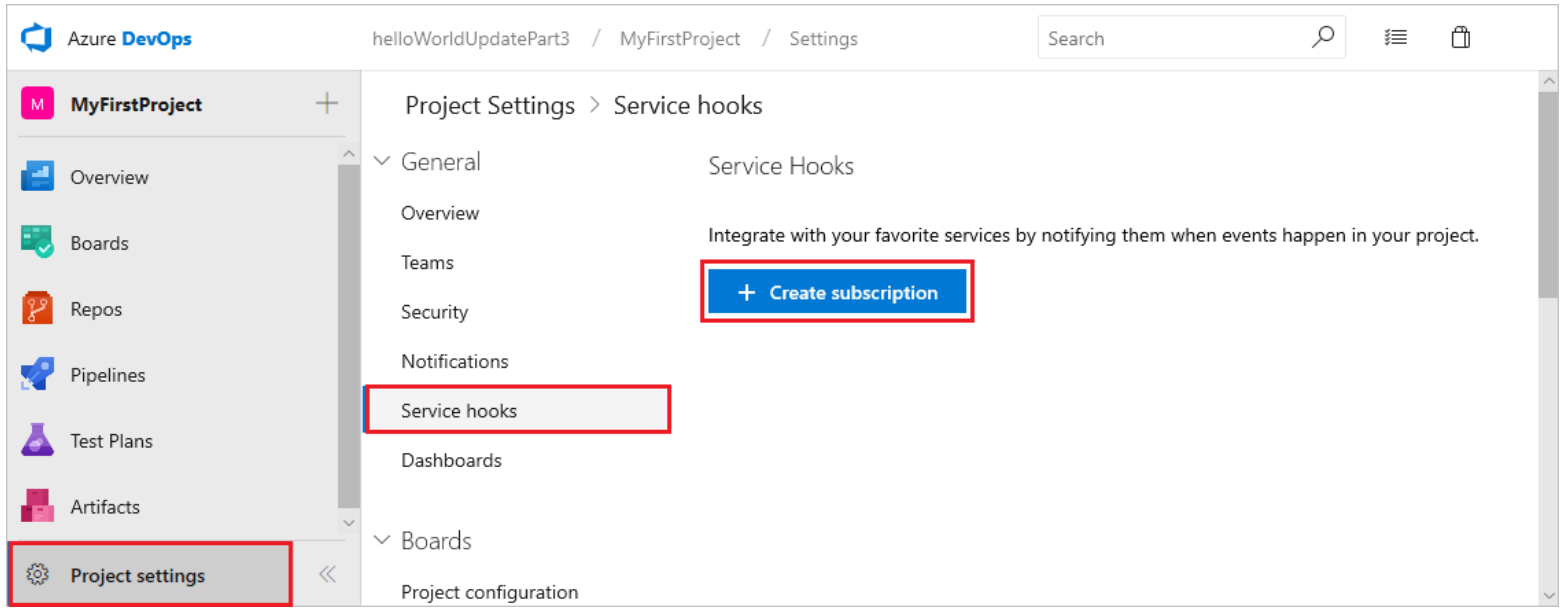
GitHooks in Action

- Will my code:
  - Break other code?
  - Introduce code quality issues?
  - Drop the code coverage?
  - Take on a new dependency?
- Will the incoming code:
  - Break my code?
  - Introduce code quality issues?
  - Drop the code coverage?
  - Take on a new dependency?



Web Hooks

- Azure DevOps also supports Web Hooks that send event info as JSON



Lesson 04: Fostering Internal Open Source

Fostering Internal Open Source

- Fork-based pull request workflows allows anybody to contribute
- **Inner Source** brings all the benefits of open source software development inside your firewall
- We recommend the forking workflow for large numbers of casual or occasional committers

GitVersion Versioning

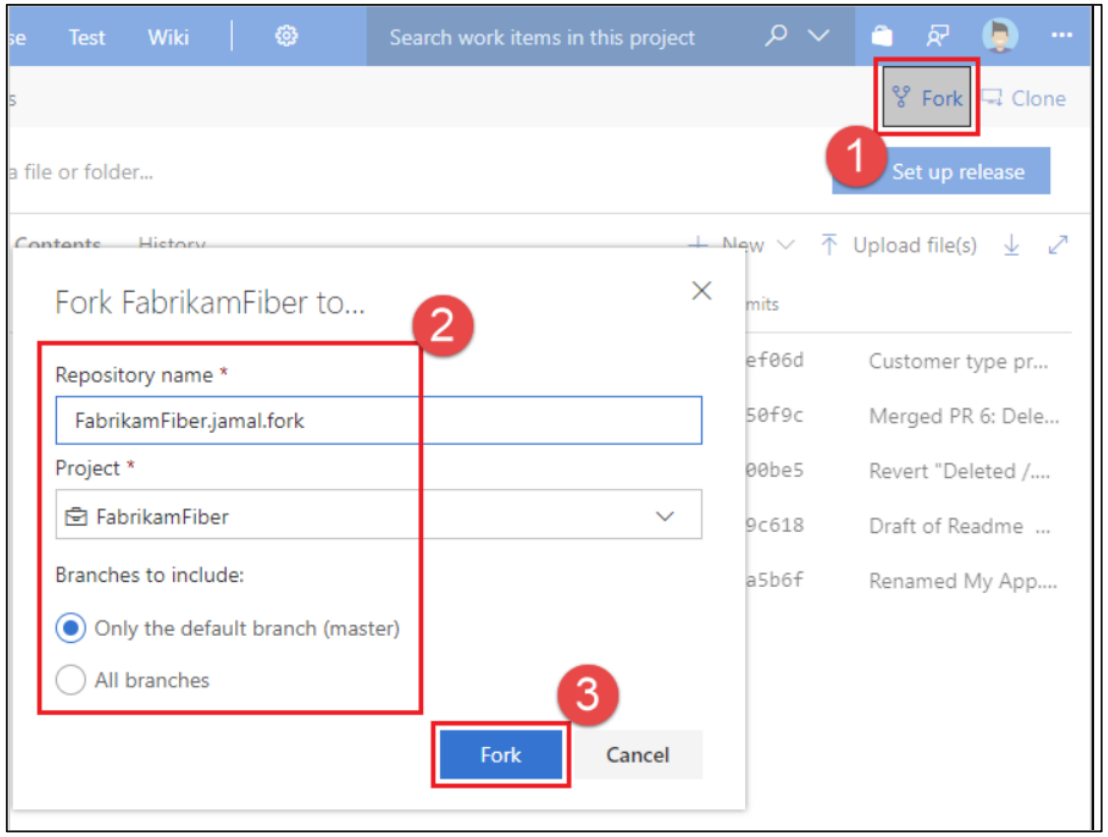
- Semantic Versioning is all about releases, not builds
- GitVersion is a tool to help you achieve Semantic Versioning
- GitVersion calculates the version based on:
  - the nearest tag
  - the commit messages between this commit and the nearest tag
  - the name of the branch

Create version numbers using the approach **MAJOR.MINOR.PATCH**

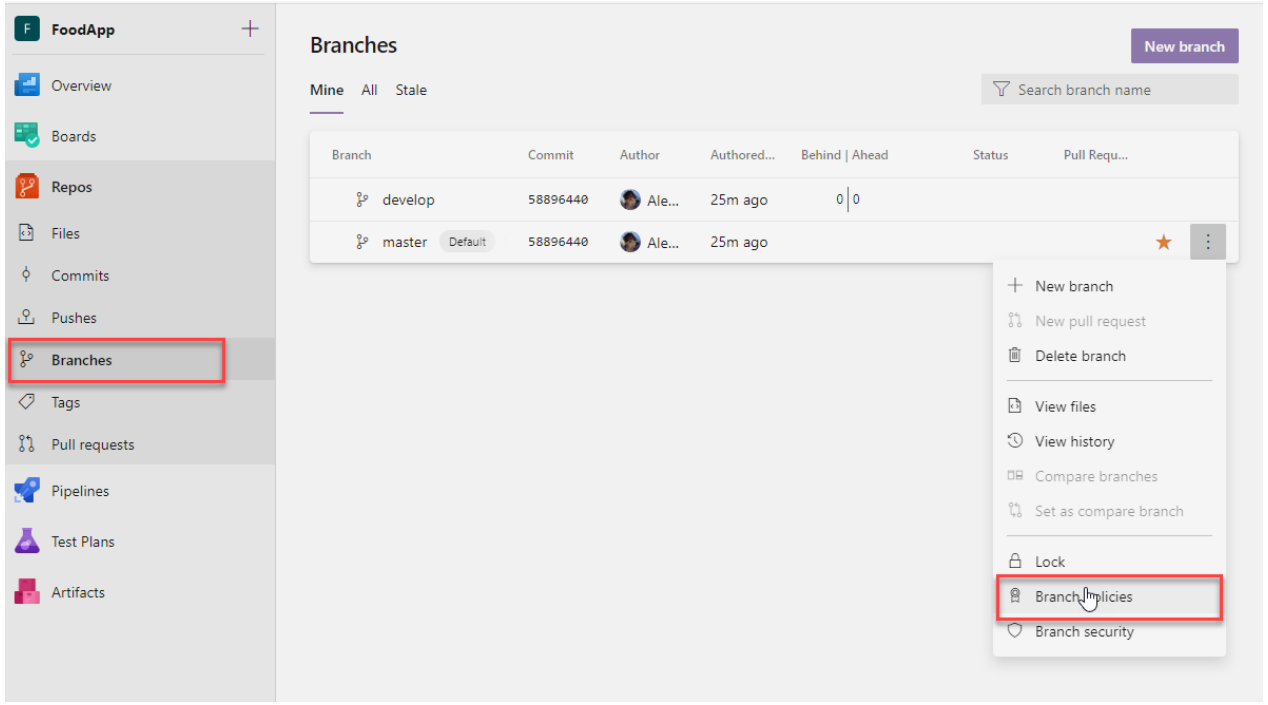
- **MAJOR** – Increment when you make incompatible API changes
- **MINOR** – Increment when you add new functionality that is backward compatible
- **PATCH** – Increment when you add bug fixes that are backward compatible

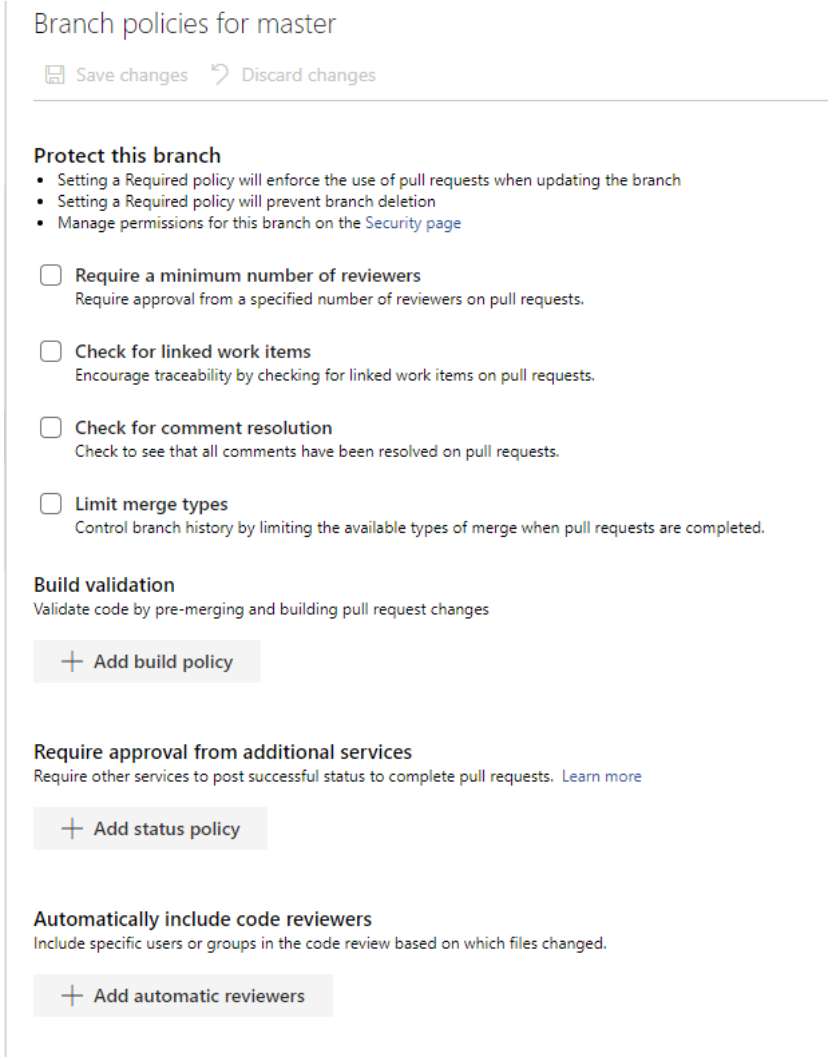
Implementing the Fork Workflow

- What's in a fork?
- Sharing code between forks
- Choosing between branches and forks
- The forking workflow
  - Create a fork
  - Clone it locally
  - Make your changes locally and push them to a branch
  - Create and complete a PR to upstream
  - Sync your fork to the latest from upstream

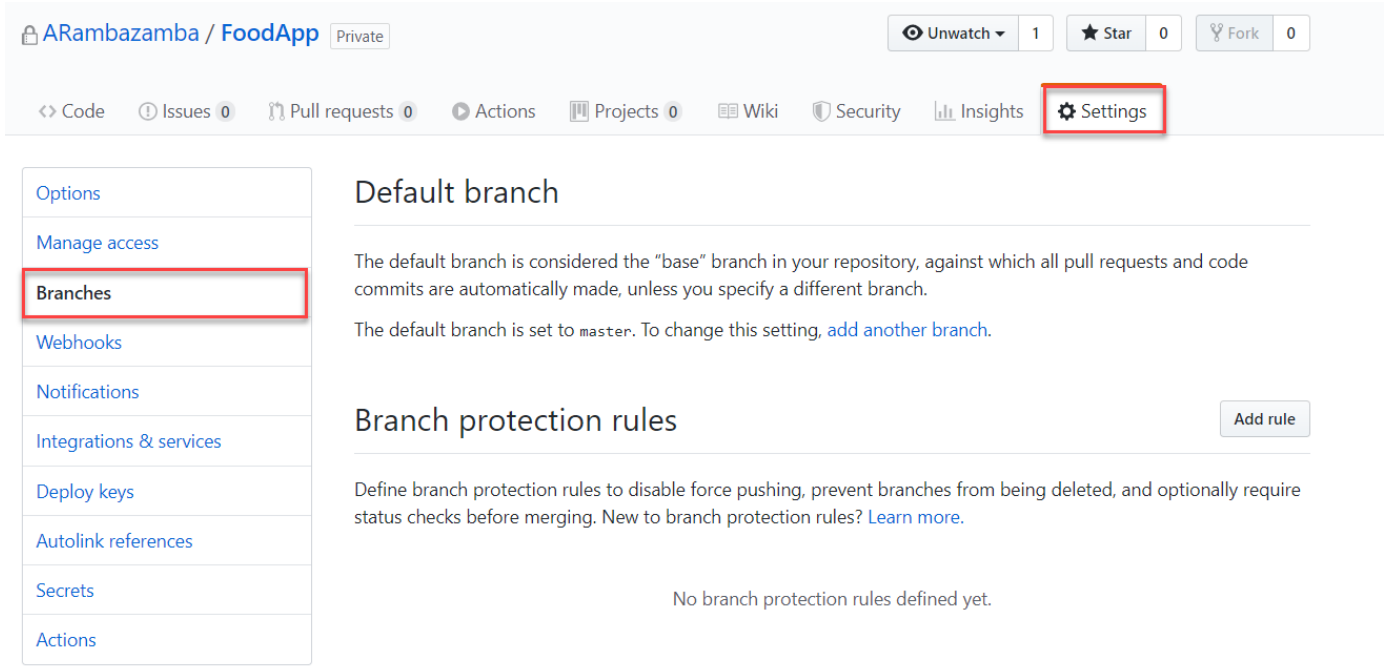


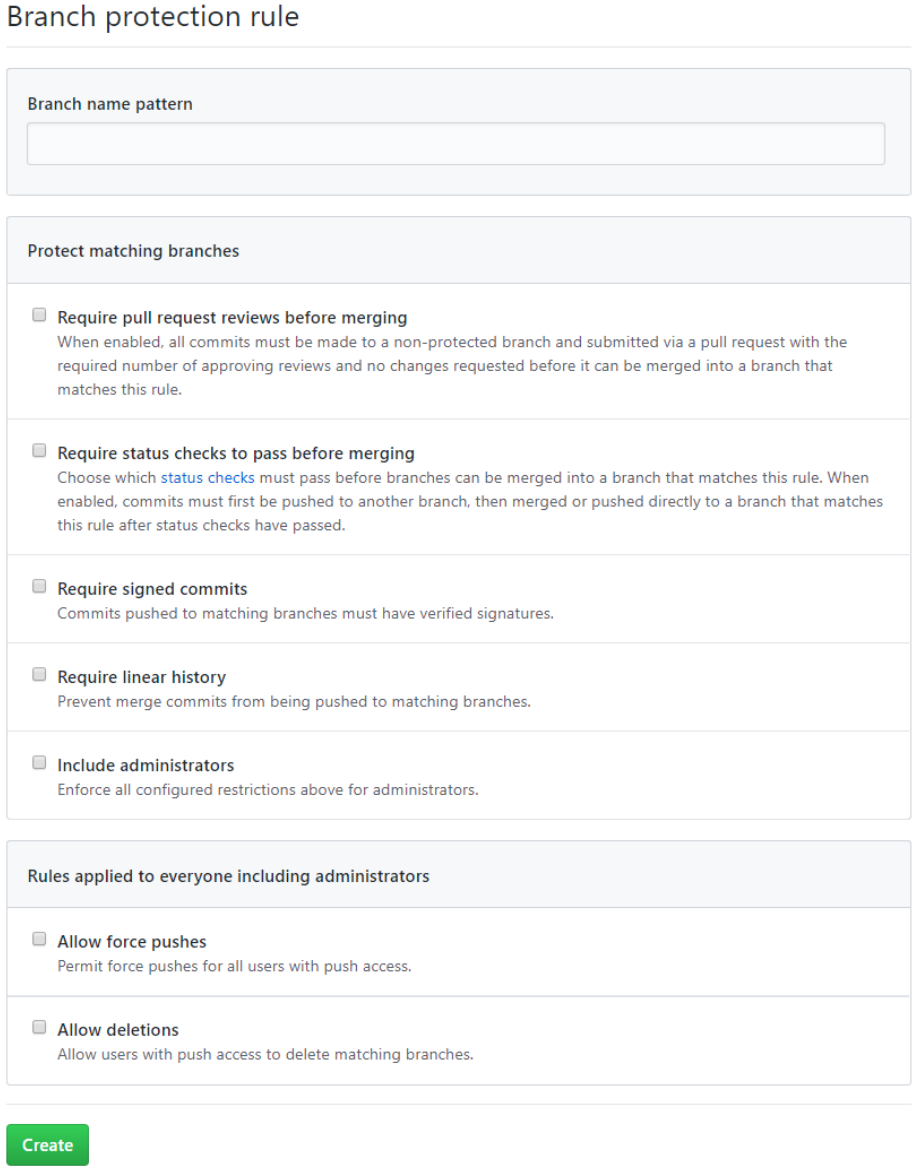
Protecting Branches Azure Repos





Branch Policies GIT





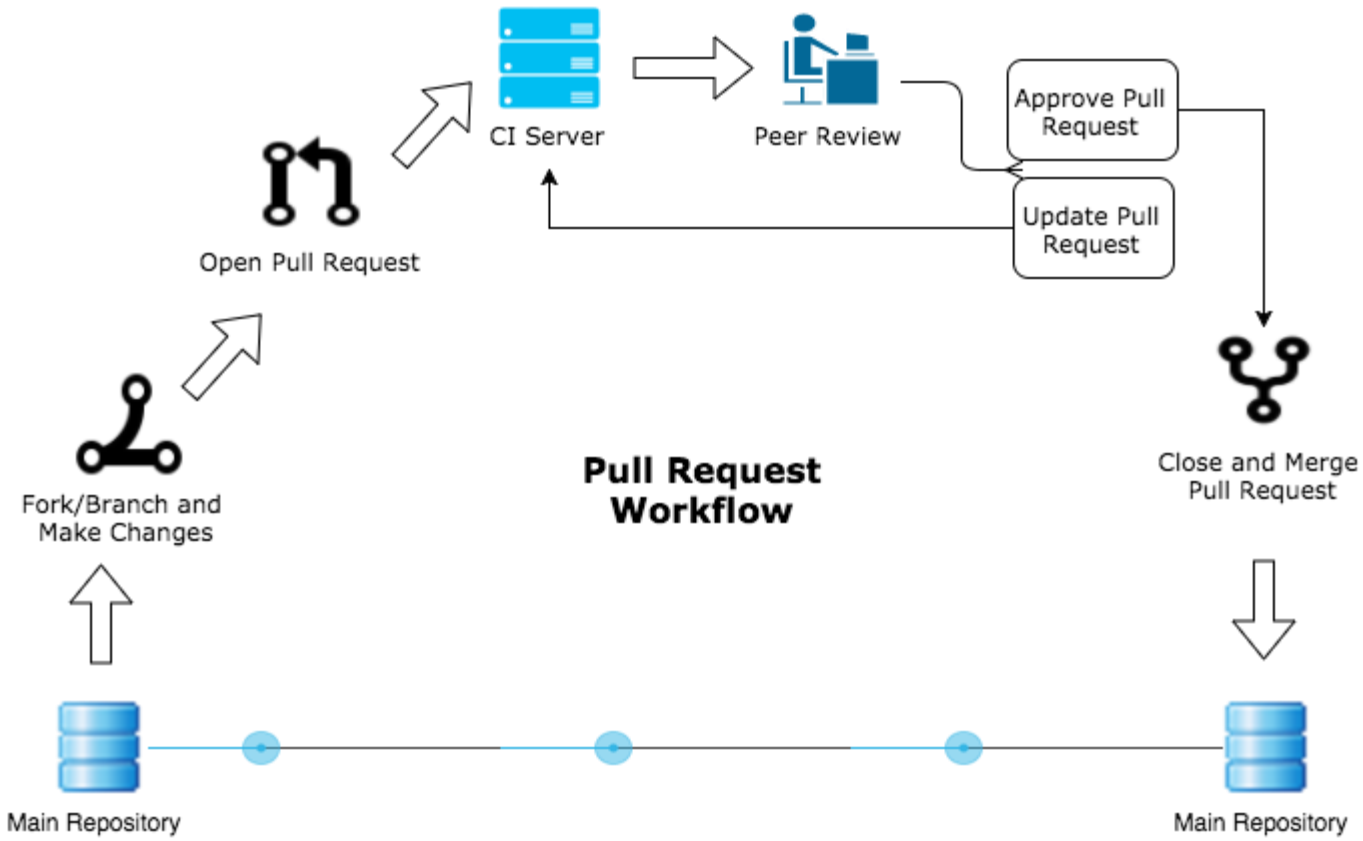
Public Projects

- Azure DevOps Team Projects can be public – no Microsoft account required
- Public projects enables anonymous users to:
  - Browse the code base, download code, view commits, branches, and pull requests
  - View and filter work items
  - View a project page or dashboard
  - View the project Wiki
  - Perform semantic search of the code or work items
- Private projects require users to be granted access to the project and signed in to access the services

Lesson 05: Collaborating with Pull Requests

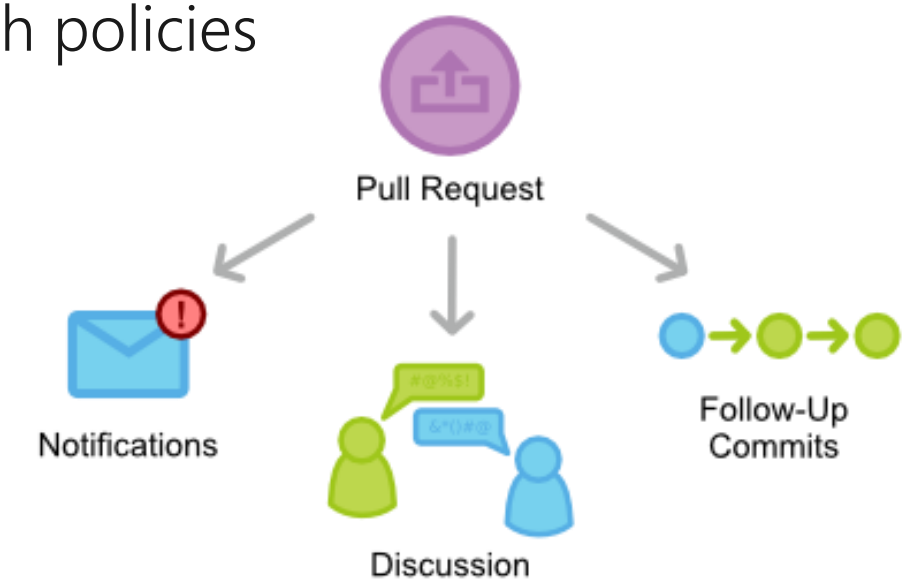
Pull Request Overview

- Base Pattern for Open Source
- Can be used internally



Collaborating with Pull Requests

- Pull requests let you tell others about changes
- Collaboration using the Shared Repository Model
- Review and merge your code in a single collaborative process
- Provide good feedback and protect branches with policies



Lab: Code Review with Pull Requests

- In this lab, Version Controlling with Git in Azure Repos, you will work branching and merging. You will learn how to:
  - Exercise 6: Working with pull requests
  - Exercise 7: Managing repositories

✔ Note that you must have already completed the prerequisite labs in the Welcome section.

Microsoft

AZ-400.1 Module 3: Manage Build Infrastructure

Lesson 01: The Concept of Pipelines in DevOps

Azure DevOps CLI

- CLI to manage Azure DevOps
- Requires Personal Access Token (PAT)
- Can set default Orga
- Reference available

azure-devops

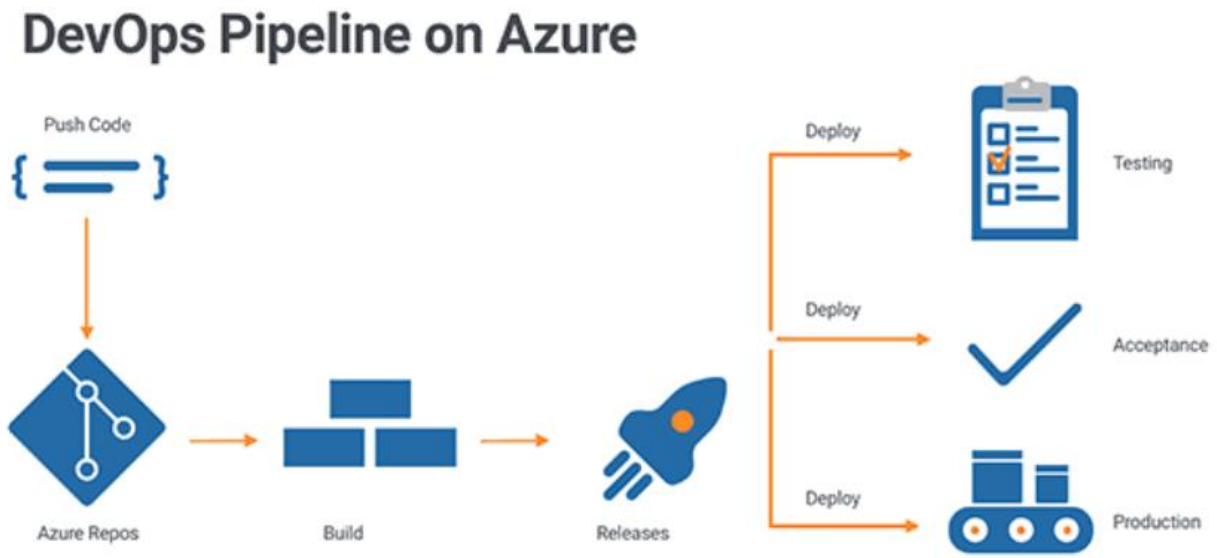
Commands

az artifacts	Manage Azure Artifacts.
az boards	Manage Azure Boards.
az devops	Manage Azure DevOps organization level operations.
az pipelines	Manage Azure Pipelines.
az repos	Manage Azure Repos.

Azure Repos with Azure DevOps CLI

The Concept of Pipelines in DevOps

- Enables a constant flow of changes into production via an automated software production line
- Create a repeatable, reliable and incrementally improving process for taking software from concept to customer
- Require infrastructure, this infrastructure will have a direct impact on the effectiveness of the pipeline



Lesson 02: Azure Pipelines

Azure Pipelines

- Azure Pipelines is a cloud service that you can use to automatically build and test your code project and make it available to other users
- Works great with Continuous Integration and Continuous Delivery
  - Work with any language or platform - Python, Java, PHP, Ruby, C#, and Go
  - Ready to use or Custom Build Agents
  - Deploy to different types of targets at the same time
  - Integrate with Azure deployments - container registries, virtual machines, Azure services, or any on-premises or cloud target (Microsoft Azure, Google Cloud, or Amazon cloud services)
  - Build on Windows, Linux, or Mac machines
  - Integrate with GitHub, Work with open-source projects

Azure Key Terms

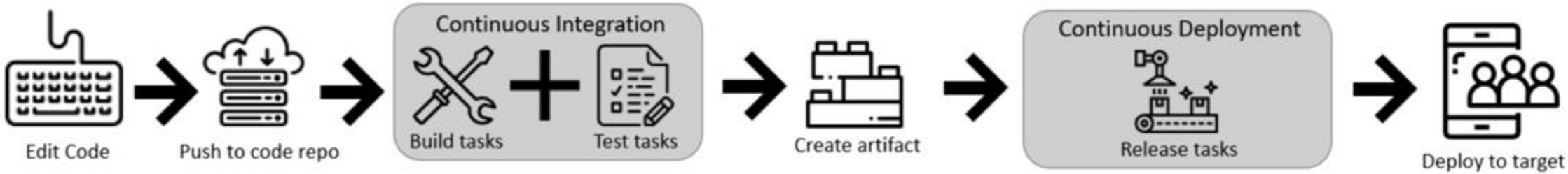
- Agent
  - Cloud Hosted / Self Hosted
- Build
  - An execution of a pipeline
- Artifact
  - Result of a build
- Continuous Integration
  - Integration of Code, Testing, Security Scanning
- Continuous Delivery
  - Code is built, tested, and deployed to one or more test / production stages
- Deployment Group / -Target
  - VM, App Service, Container, ...
- Pipeline
  - Orchestrates Build / Deployment
- Job
  - A step of a build
- Release
- Task
  - Building block of a pipeline
- Trigger

Pipeline Types

- Visual Designer
  - Easy to get started
  - Seperation of Build / Release
- YAML based
  - Must be enabled manually
  - Support Multistage
  - Implemented using azure-pipelines.yml -> can be checked in to source control
  - Support splitting -> Linked Pipelines

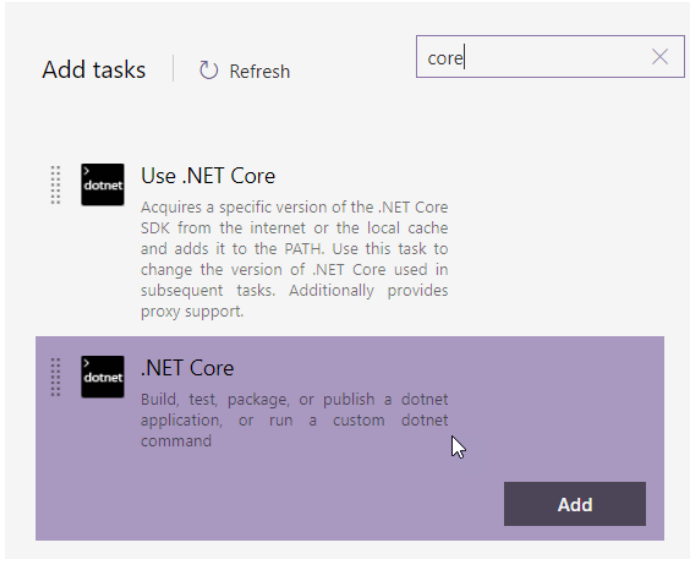
Azure Pipelines and Visual Designer

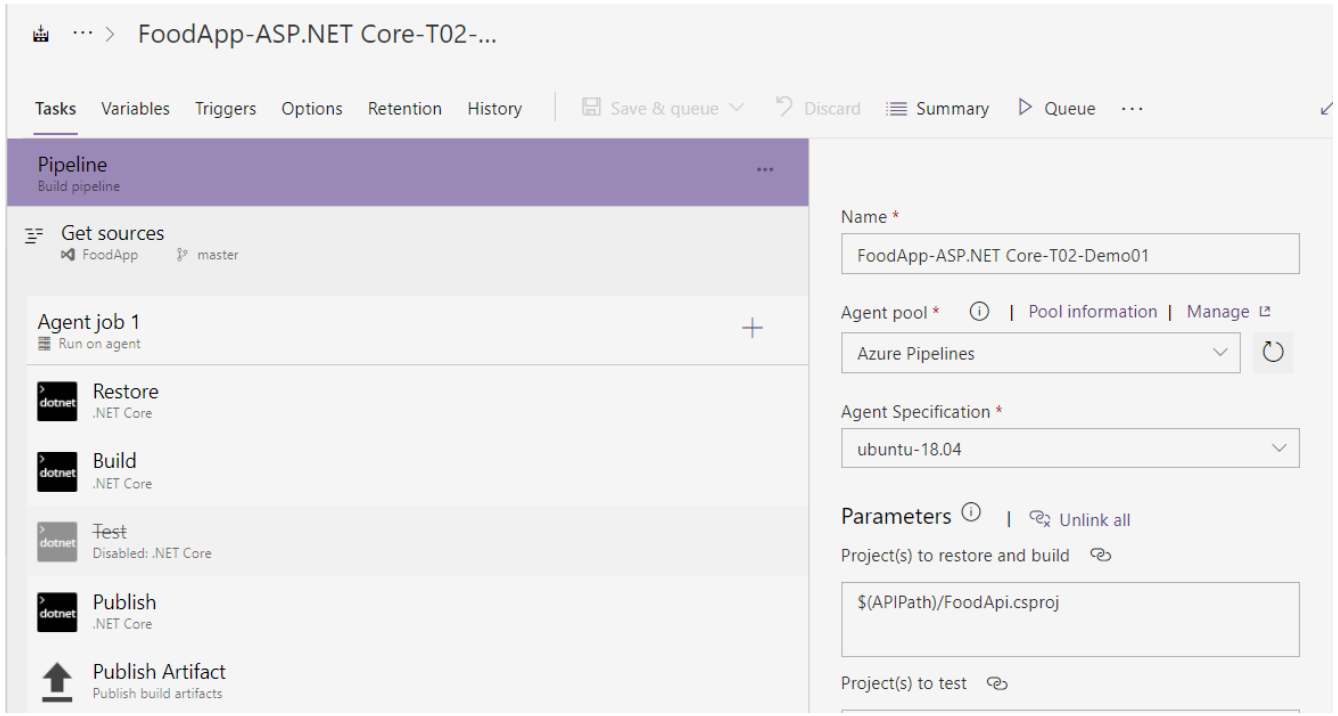
- Configure your pipelines with the Visual Designer
- Create and configure your build and release pipelines
- Push your code to your version control repository
- The build creates an artifact that's used by the rest of your pipeline
- Your code is now updated, built, tested, and packaged



Azure Pipelines and Visual Designer

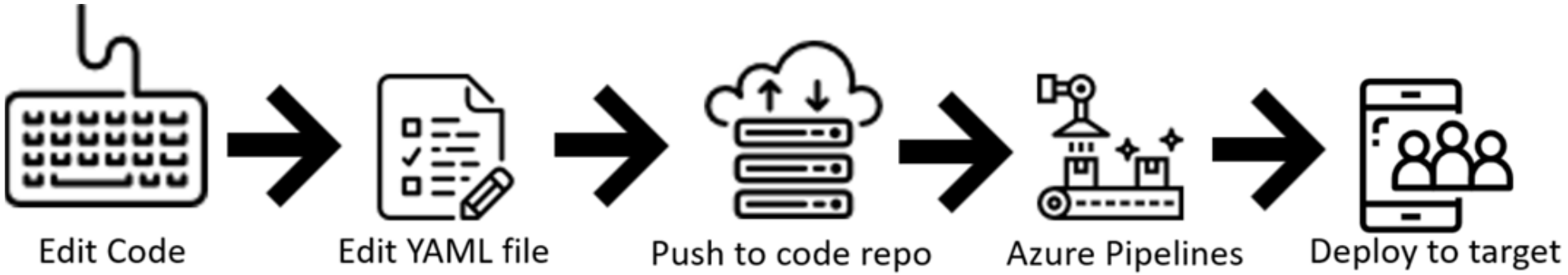
- A **task** is simply a packaged **script** or procedure that has been abstracted with a set of inputs





Azure Pipelines and YAML

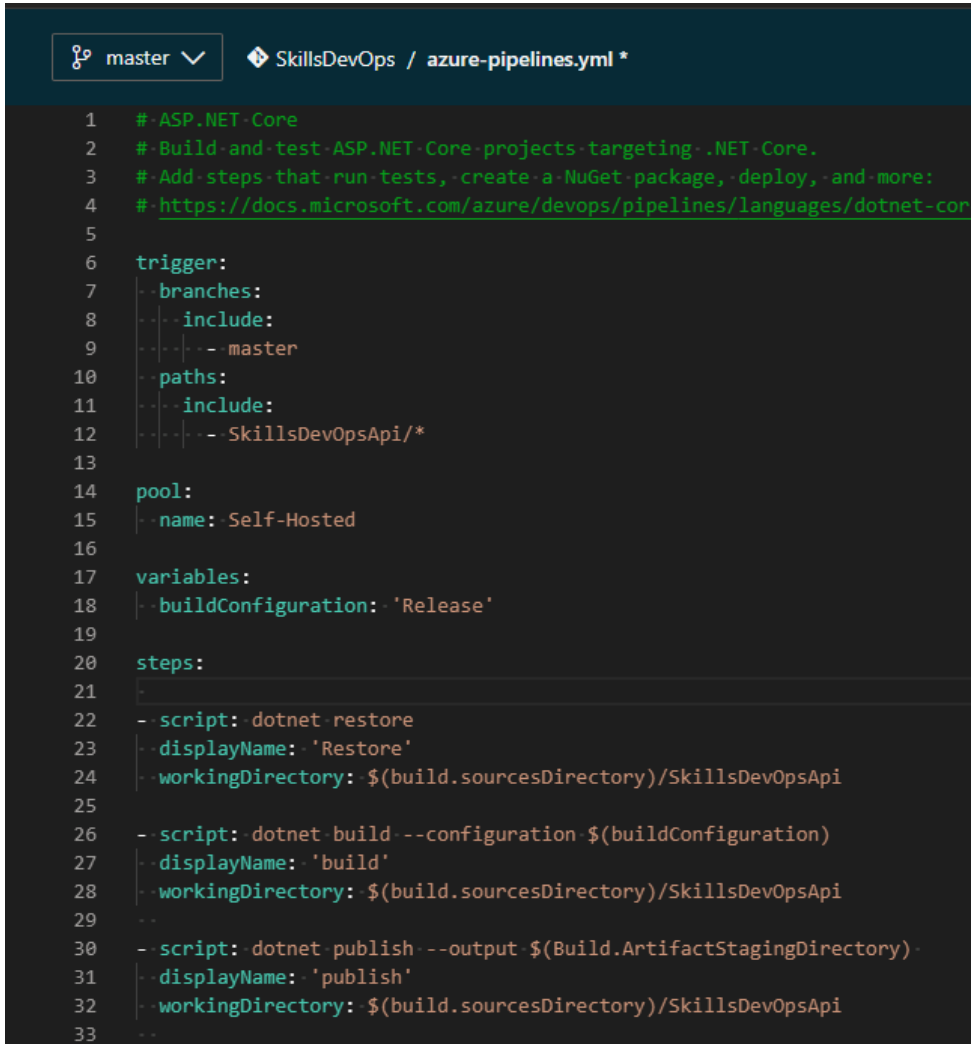
- Configure your pipelines in a YAML file that exists alongside your code
- Configure Azure Pipelines to use your Git repo
- Edit your azure-pipelines.yml file to define your build
- Push your code to your version control repository
- Your code is now updated, built, tested, and packaged



Azure Pipelines and YAML

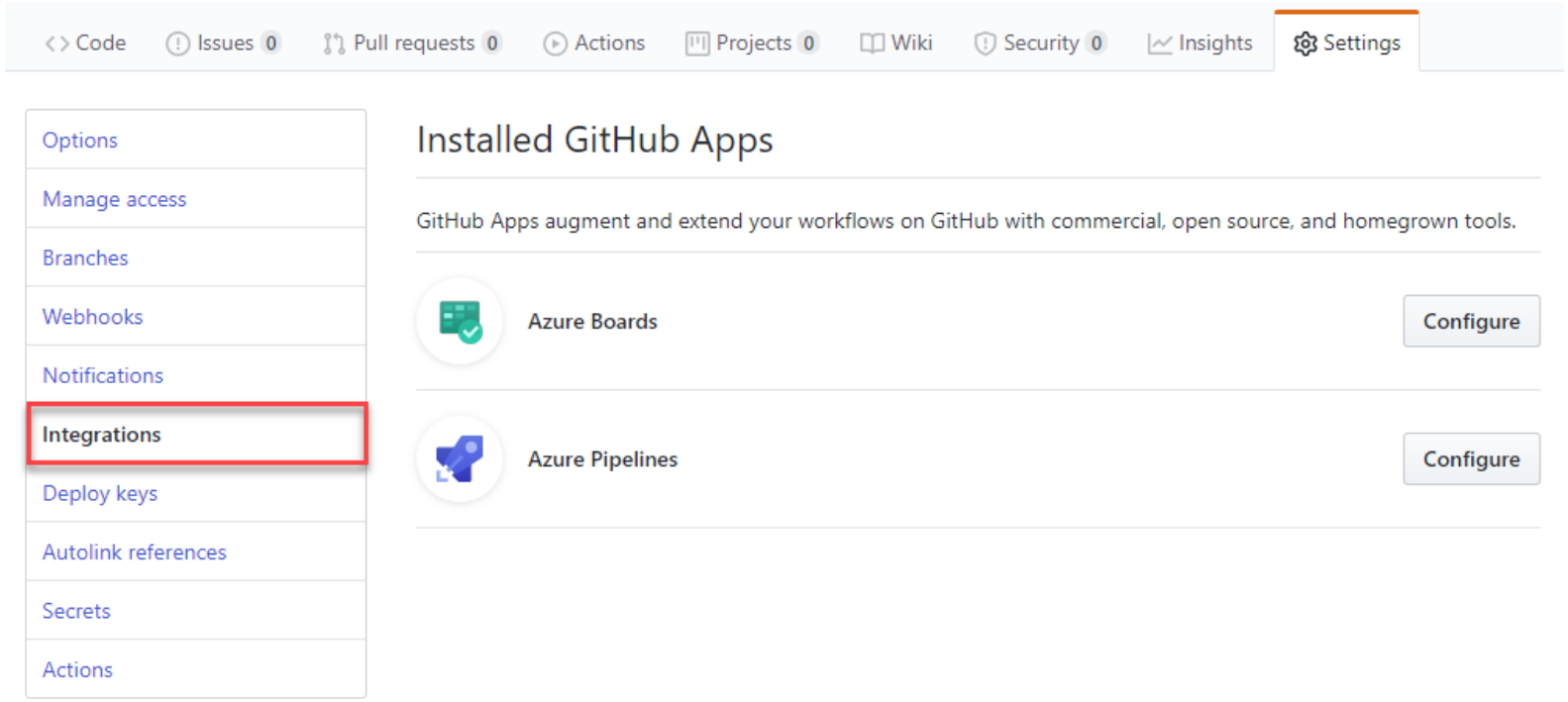
- This hierarchy is reflected in the structure of a YAML file like:

  - Pipeline
    - Stage A
      - Job 1
        - Step 1.1
        - Step 1.2
        - ...
      - Job 2
        - Step 2.1
        - Step 2.2
        - ...
    - Stage B
      - ...

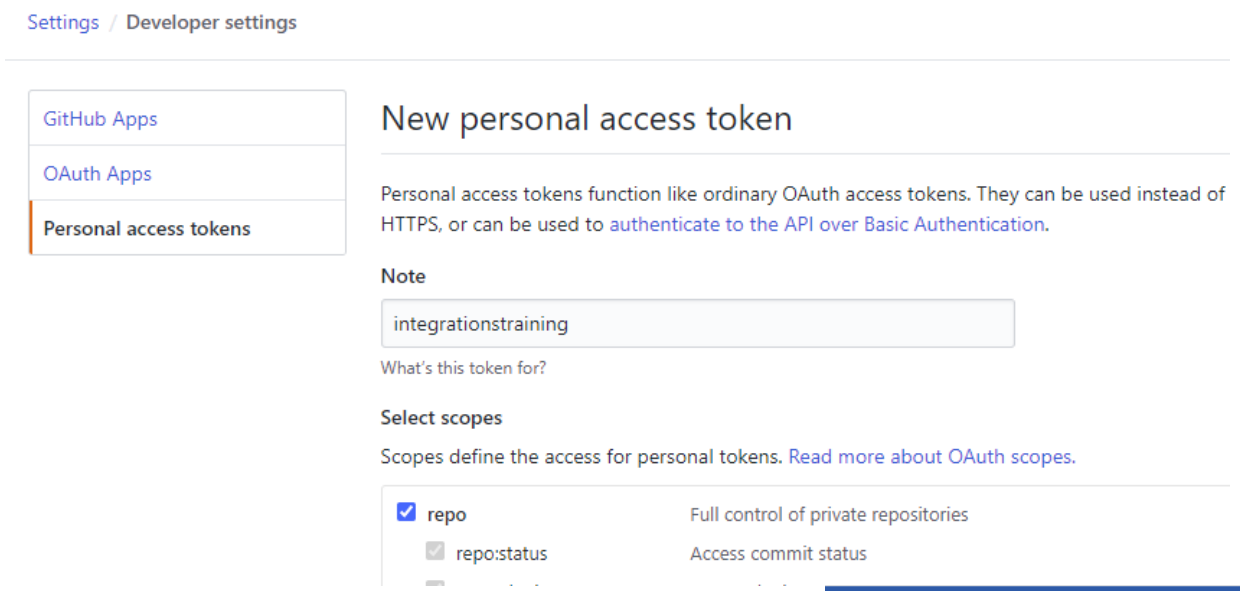


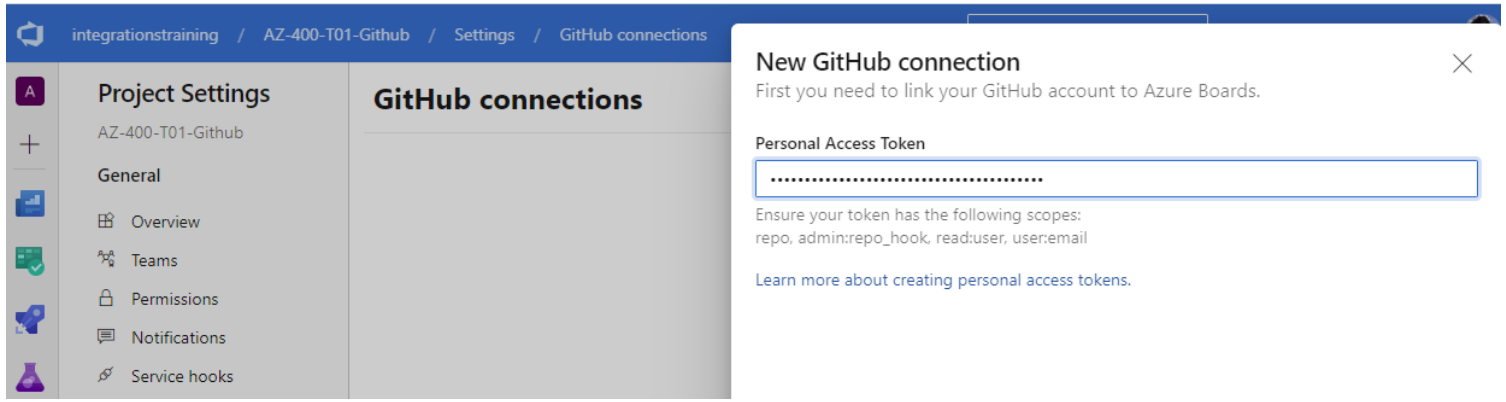
Integration External Source Control with Azure Pipelines

- By connecting with GitHub repositories, you enable linking between GitHub commits, pull requests, and issues to work items.



Github Service Connection





Lab: Integrate Your GitHub Projects With Pipelines

In this lab, Integrate Your GitHub Projects With Azure Pipelines, you will how easy it is to set up Azure Pipelines with your GitHub projects and how you can start seeing benefits immediately. You will learn how to:

- Install Azure Pipelines from the GitHub Marketplace.
- Integrate a GitHub project with an Azure DevOps pipeline.
- Track pull requests through the pipeline.

✔ Note that you must have already completed the prerequisite labs in the Welcome section.

Microsoft

AZ-400.1 Module 4: Managing Application Config & Secrets

Lesson 03: Rethinking Application Config Data

Rethinking Application Config Data

- Configuration information is stored in files
- Changes can require downtime and administrative overhead
- Challenging to manage changes to local configurations across multiple running instances of the application

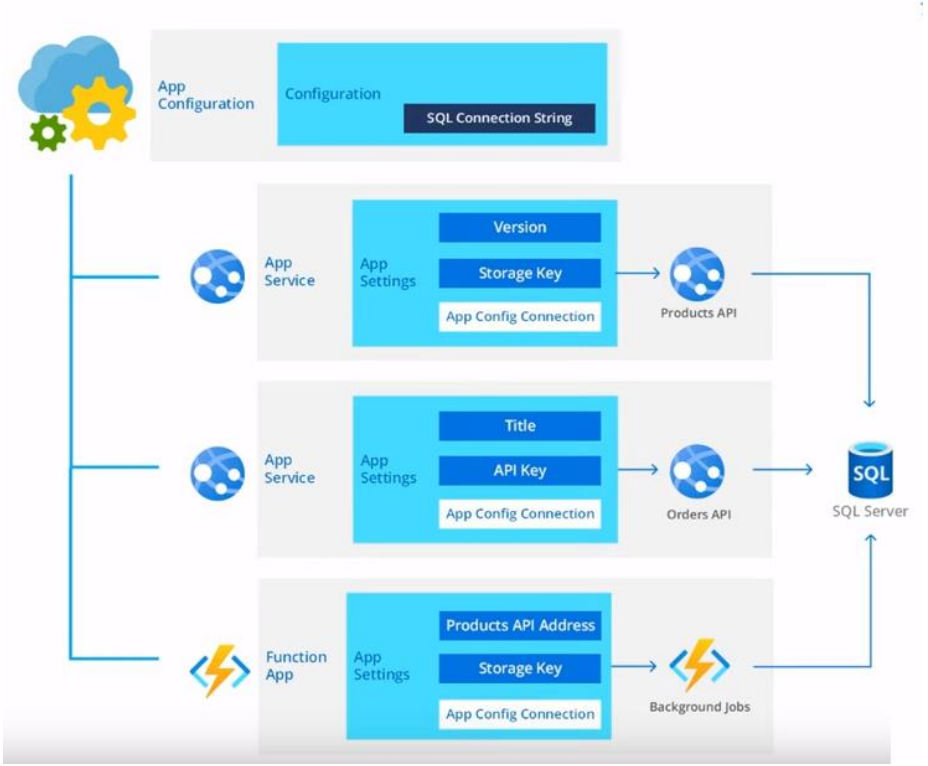
- **Discussion**: How will secure this information?

Separation of Concerns

- **Configuration Custodian**: Responsible for generating and maintaining the life cycle of configuration values
- **Configuration Consumer**: Responsible for defining the schema (loose term) for the configuration and then consuming the configuration values in the application or library code
- **Configuration Store**: The underlying store that is leveraged to store the configuration
- **Secret Store**: A separate store for persisting secrets

Azure App Configuration

- Provides a service to centrally manage application settings and feature flags
- Complements Azure Key Vault, which is used to store application secrets.



Lab: Integrating Azure Key Vault with Azure Pipeline

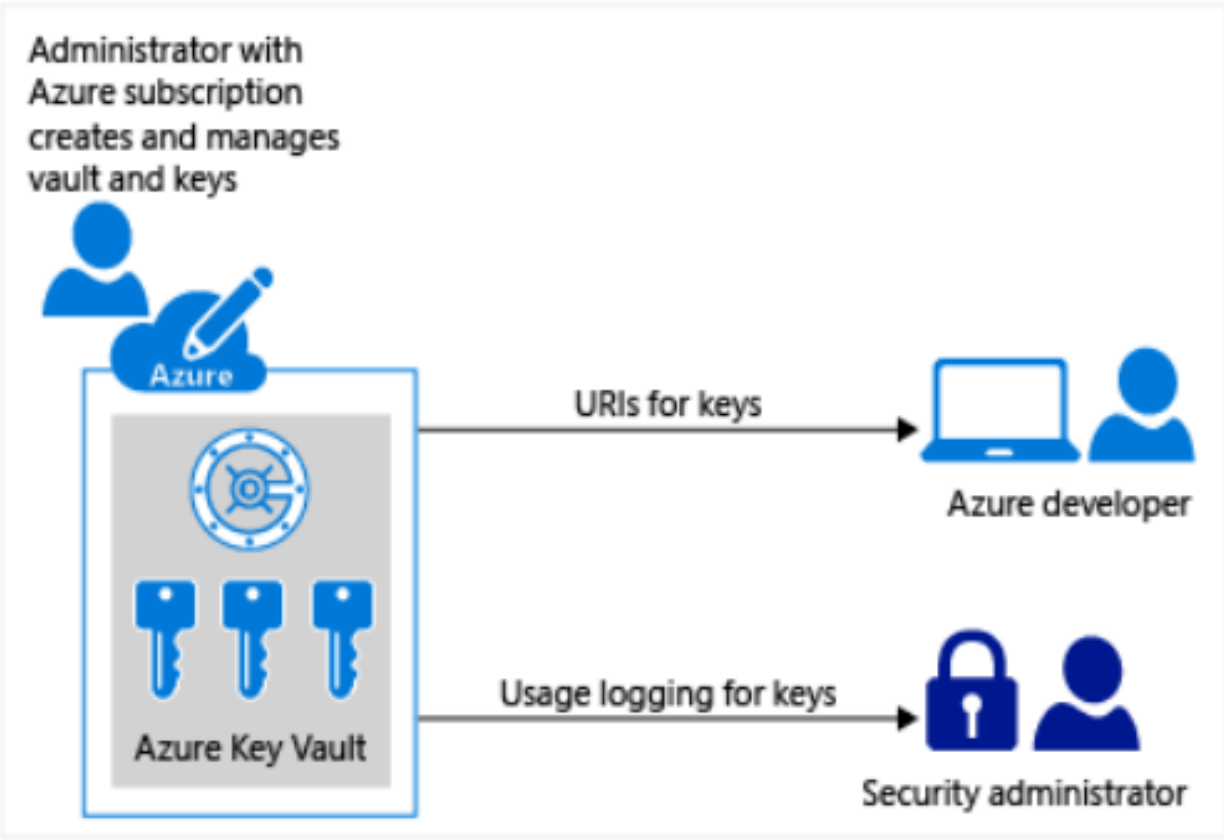
Complete the Integrating Azure KeyVault with Azure DevOps. You will learn how to:

- Create a key vault, from the Azure portal, to store a MySQL server password
- Configure permissions to let a service principal to read the value
- Retrieve the password in an Azure pipeline and passed on to subsequent tasks

Lesson 04: Manage Secrets, Tokens, and Certificates

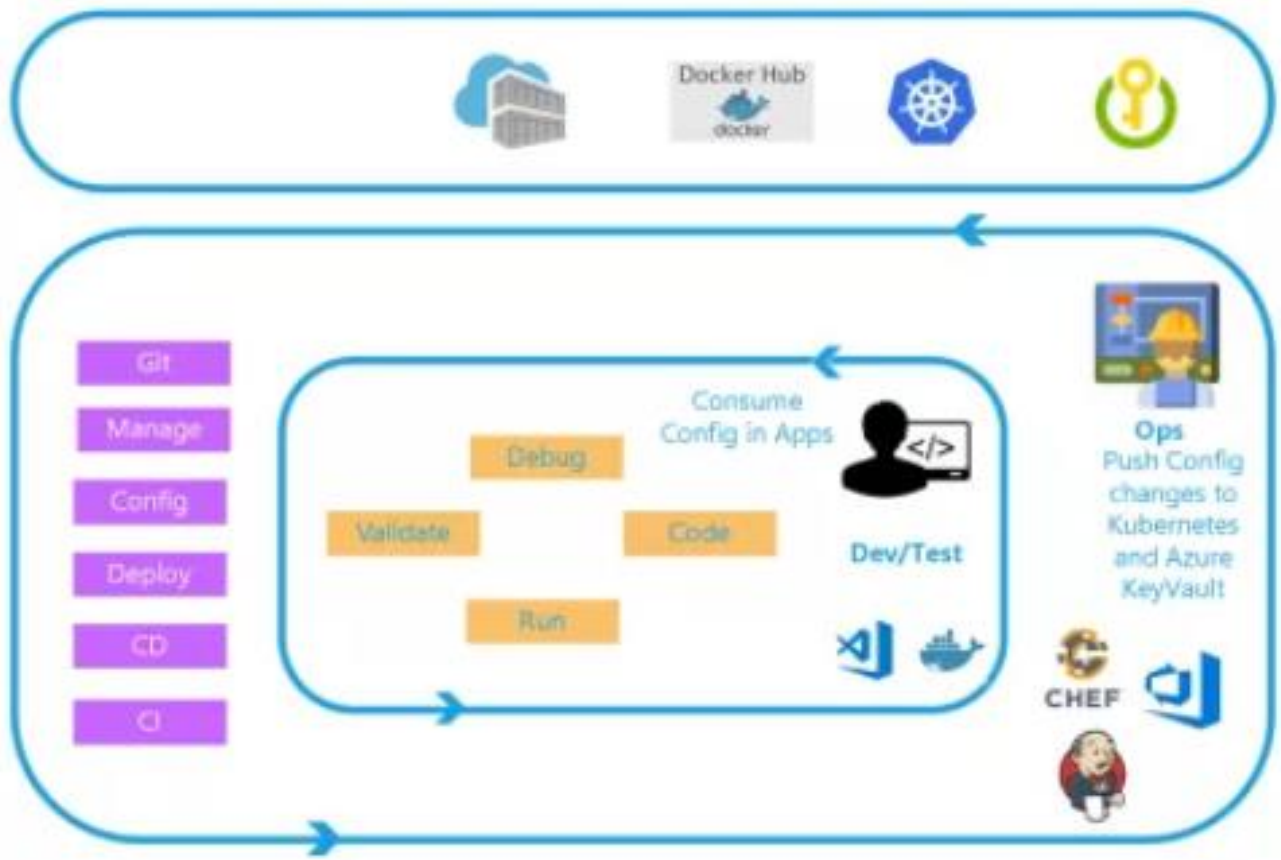
Manage Secrets, Tokens & Certificates

- Centralize application secrets
- Securely store secrets and keys
- Monitor access and use
- Simplified administration of application secret
- Integrate with other Azure services



DevOps Inner and Outer Loop

- The Inner loop is focused on the developer teams iterating over their solution development
- Developer teams consume the configuration published by the outer loop
- DevOps Engineers govern the configuration management and pushes the changes



Kubernetes and Azure Key Vault

- Use Kubernetes and Azure Key Vault together to get the best security benefits:
  - Azure Key Vault Secret store
  - Kubernetes ConfigMaps
  - Kubernetes Secrets