Microsoft

Intro to GitOps

Ben Mitchell | Sr. CSA | Microsoft

Agenda

- Introductions
- Getting things Setup
- What is GitOps
- Getting Started with Git
- Getting Started with Docker
- Config Management
- Automation with GitHub Actions
- Close Out

Introductions

Benjamin Mitchell

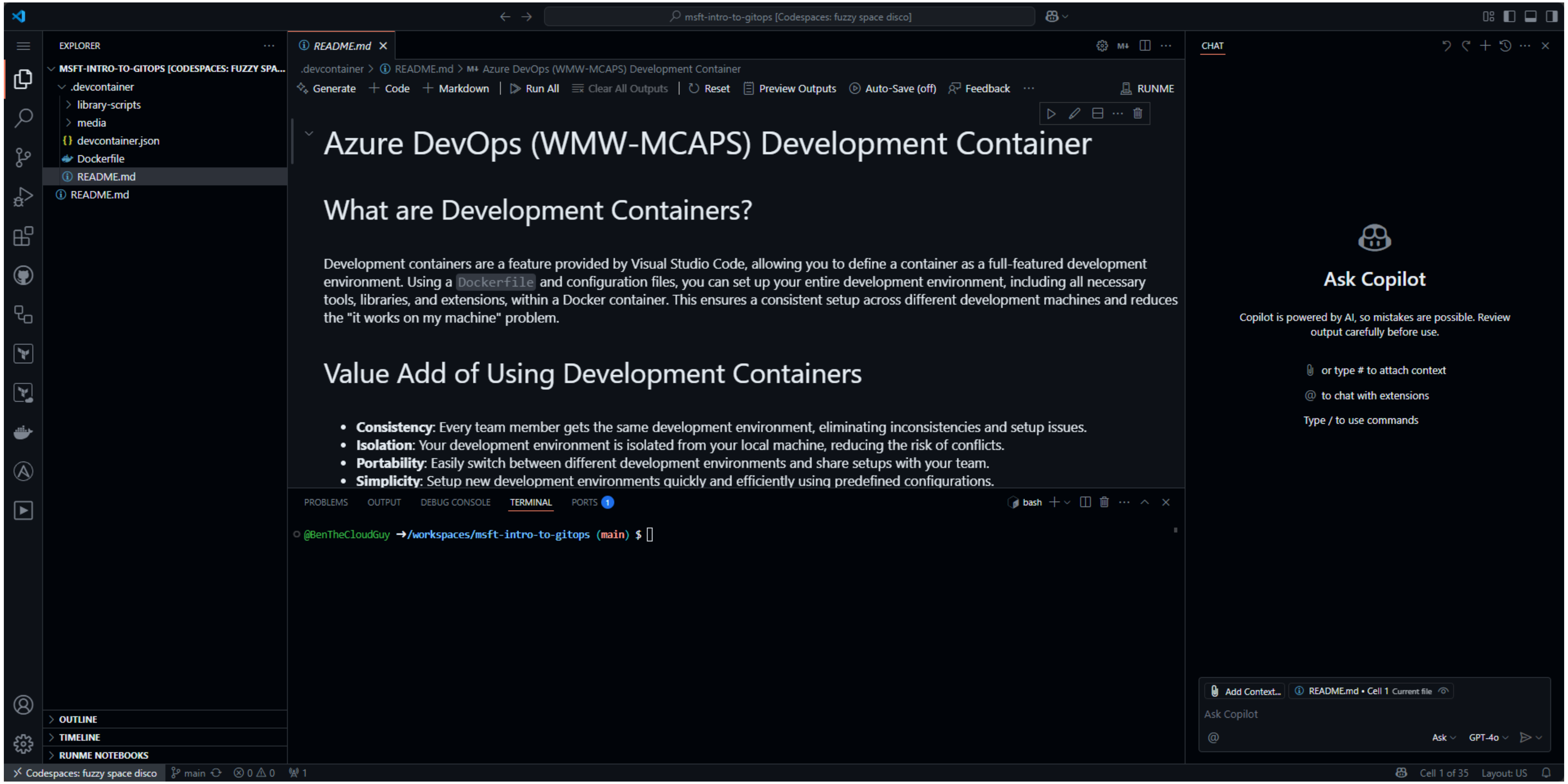
Sr. Cloud Solution Architect

Microsoft

Getting Setup

- Sign into GitHub to access the workshop materials.
- Clone the Workshop Directory to your local machine.
- Launch the GitHub Codespace for seamless development.
- Ensure all participants have the necessary tools installed.
- Quick look at VSCode and CodeSpaces

CodeSpaces and VSCode



msft-intro-to-gitops [Codespaces: fuzzy space disco]
EXPLORER
MSFT-INTRO-TO-GITOPS [CODESPACES: FUZZY SPA...
.devcontainer
library-scripts
media
devcontainer.json
Dockerfile
README.md
README.md
OUTLINE
TIMELINE
RUNME NOTEBOOKS
README.md
.devcontainer > README.md > Azure DevOps (WMW-MCAPS) Development Container
Generate
Code
Markdown
Run All
Clear All Outputs
Reset
Preview Outputs
Auto-Save (off)
Feedback
RUNME
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS 1
bash
CHAT
Ask Copilot
Copilot is powered by AI, so mistakes are possible. Review output carefully before use.
or type # to attach context
@ to chat with extensions
Type / to use commands
Add Context...
README.md • Cell 1 Current file
Ask Copilot
Ask
GPT-4o
Codespaces: fuzzy space disco
main
0 0
1
Cell 1 of 35
Layout: US

Azure DevOps (WMW-MCAPS) Development Container

What are Development Containers?

Development containers are a feature provided by Visual Studio Code, allowing you to define a container as a full-featured development environment. Using a `Dockerfile` and configuration files, you can set up your entire development environment, including all necessary tools, libraries, and extensions, within a Docker container. This ensures a consistent setup across different development machines and reduces the "it works on my machine" problem.

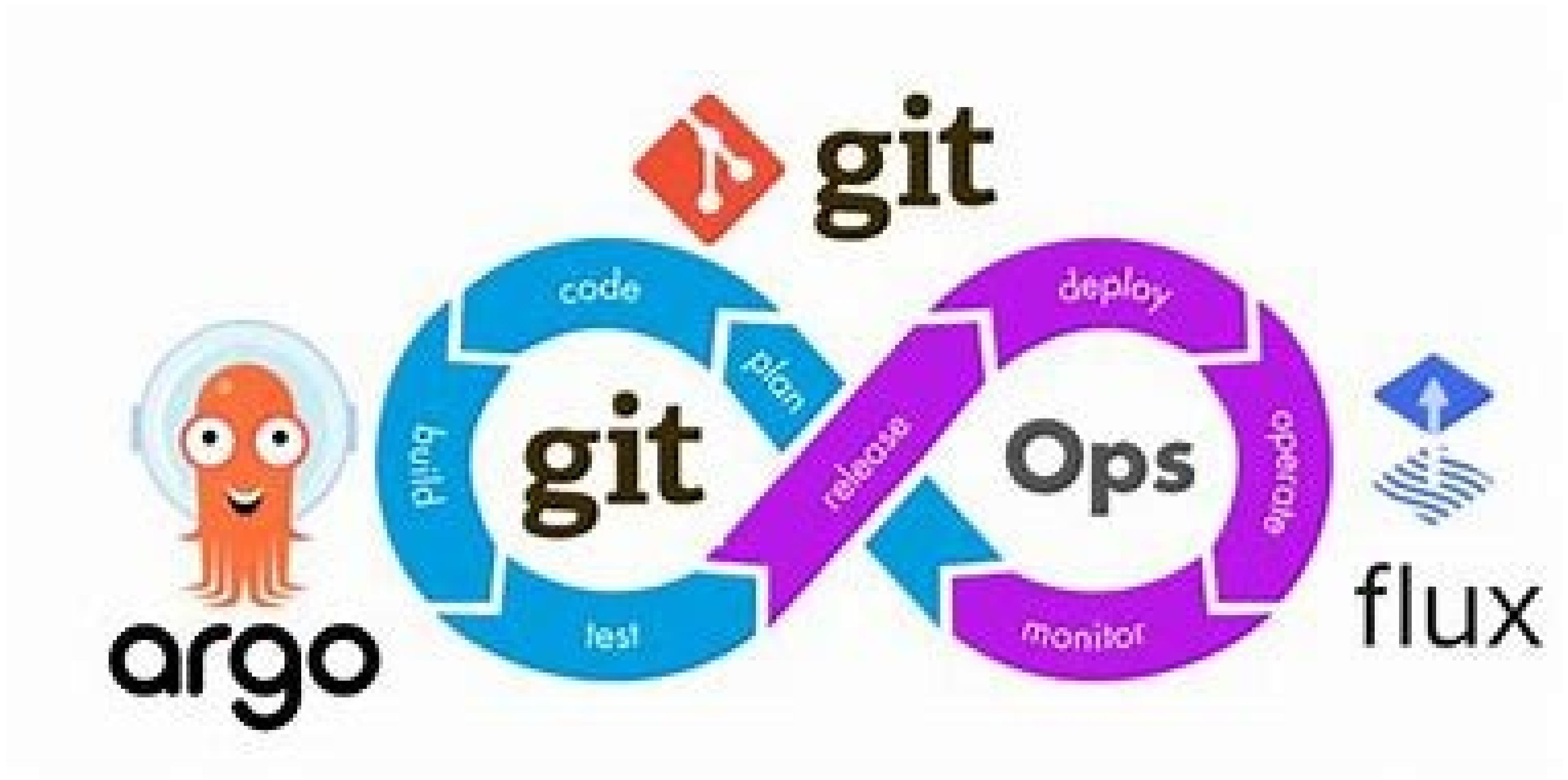
Value Add of Using Development Containers

- **Consistency**: Every team member gets the same development environment, eliminating inconsistencies and setup issues.
- **Isolation**: Your development environment is isolated from your local machine, reducing the risk of conflicts.
- **Portability**: Easily switch between different development environments and share setups with your team.
- **Simplicity**: Setup new development environments quickly and efficiently using predefined configurations.

```
@BenTheCloudGuy ➜/workspaces/msft-intro-to-gitops (main) $ 
```

What is GitOps

What is GitOps



GitOps Principles

Principle #1: GitOps Is Declarative.

- All Infrastructure and application configuration is defined in Code.
- Enable consistency, repeatability, and ease of recovery

Principle #2: GitOps Apps Are Versioned and Immutable

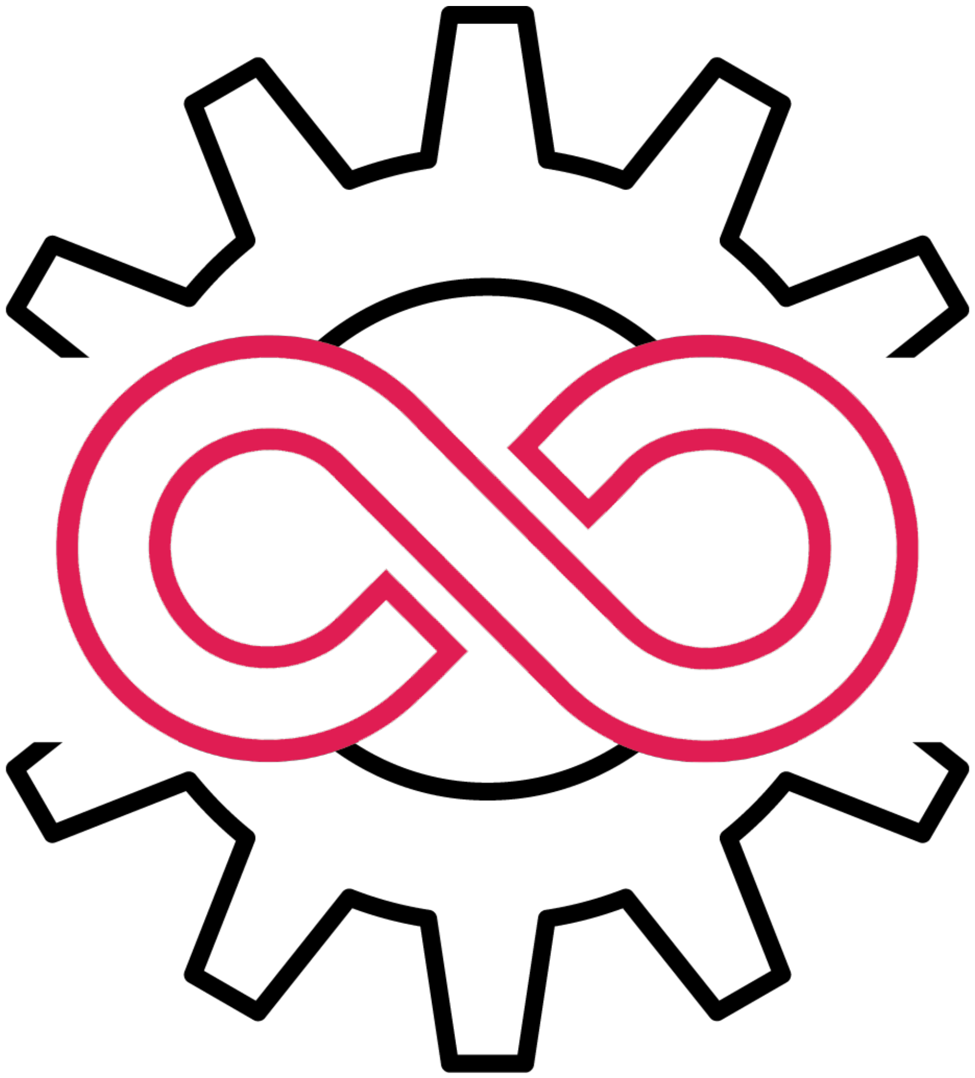
- Every change is recorded, auditable, and revertible
- Git becomes the Single Source of Truth
- Allows safe rollbacks and peer-reviewed changes

Principle #3: GitOps Apps Are Pulled Automatically

- GitOps tools monitor the Git repo and apply updates
- Pull-based model ensures security (no external push)

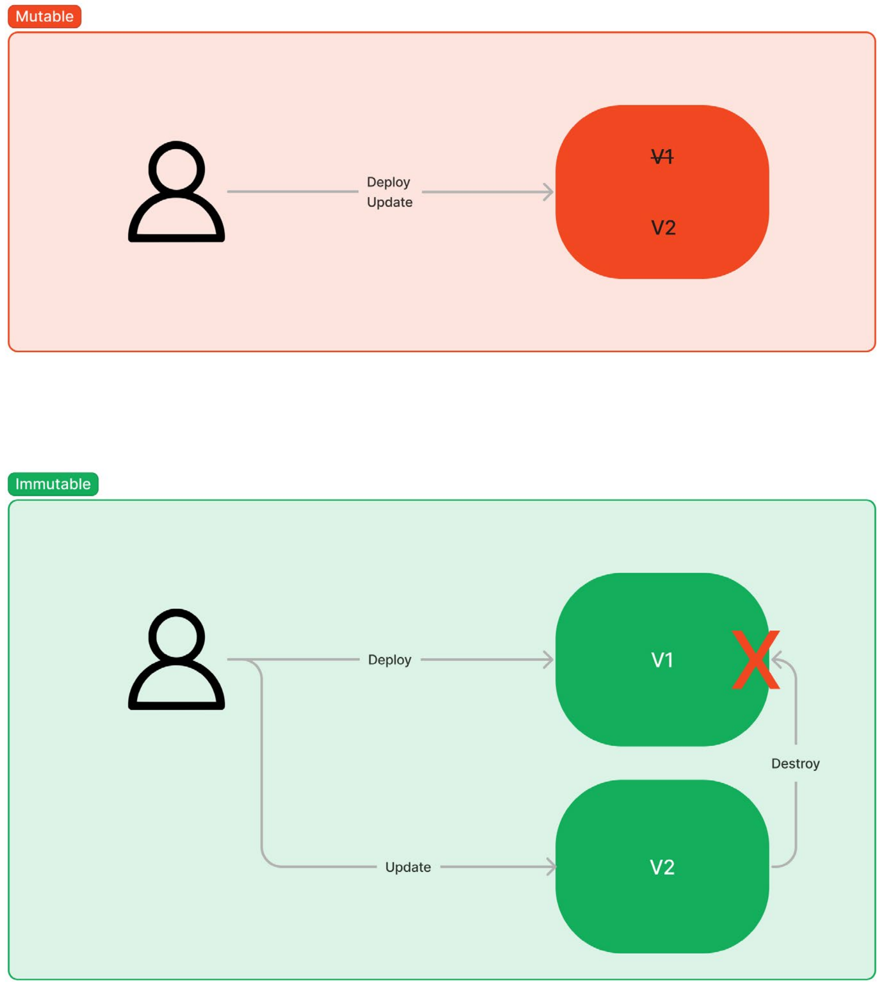
Principal #4: GitOps Apps Are Continuously Reconciled

- Ensures self-healing infrastructure
- Detects and corrects manual drift or unauthorized charges
- Provides observability into sync status



Benefits

- Immutable Infrastructure
- Fully Automated Systems
- Prevent Manual Intervention
- Everything as code
- Auditability & Change Control
- Improved Collaboration
- Faster & Safer Deployments
- Enhanced Security Posture
- Multi-Environment Consistency
- BC/DR built in by design



GitOps Architecture Options

Architecture	Best For	Complexity	Scales Well?	Team Isolation
Single-Repo	Demos, POCs	Low	❌	❌
Two-Tree	SMBs, mid-size ops	Medium	✅	⚠️ Partial
Three-Tree	Enterprise, regulated orgs	High	✅✅	✅
Env-per-Repo	Secure orgs, controlled release flows	Medium	✅	✅
Team-Based	Microservices, agile orgs	Medium	✅✅	✅✅
Layered	Platform teams + app teams	High	✅✅✅	✅
App-of-Apps (Argo CD)	Kubernetes-heavy orgs	Medium	✅✅	⚠️ Coupled
Multi-Tenant	SaaS / B2B / Service Providers	Very High	✅✅✅	✅✅✅

Getting Started with Git

Getting Started with Git

A little bit of history

- Git was created by Linus Torvalds in April 2005 to develop Linux Kernel.
- Replacement for bitkeeper to manage Linux Kernel changes.

What is Git.

- A command line distributed version control solution.
- Uses checksums to ensure Data Integrity.
- Cross Platform(Including Windows).
- Free & open-source control system .

What is Repository

- Repo = Repository
- Usually needs to organize a single project.
- It contains folders, files, images, videos, spreadsheets and datasets --- anything your project needs.

Common Git Commands

- Git config: Configure your git environment
- Git init: Initializes a new Git repository.
- Git clone: Creates a local copy of a remote repository.
- Git add: Stages changes for the next commit.
- Git commit: Saves staged changes with a message.
- Git push: Uploads local commits to the remote repository.
- Git pull: Fetches and merges changes from the remote repository.

Configure Git

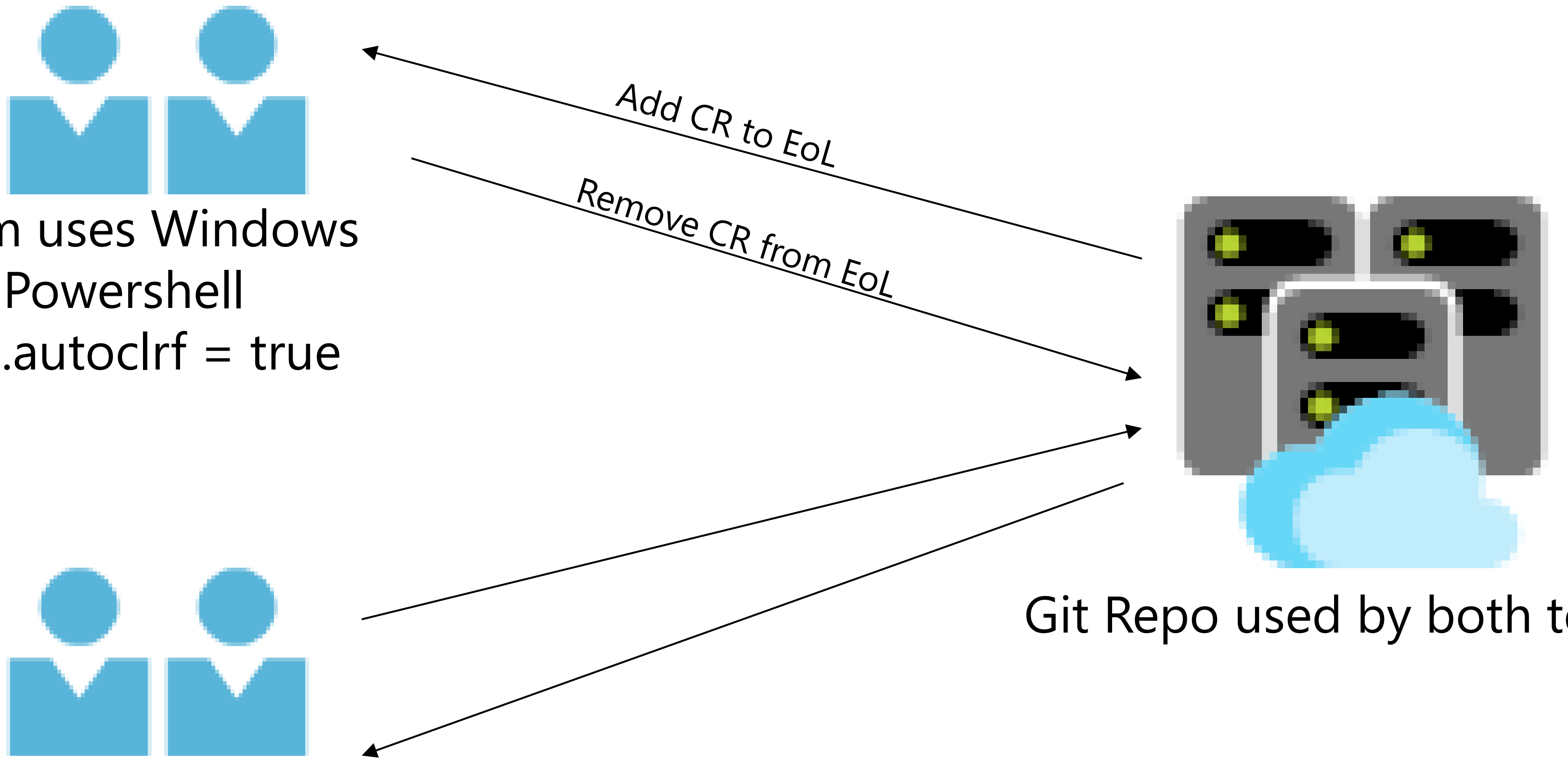
- Name
  - The ***git config --global user.name*** configuration sets the name that will be associated with your commits in Git. This name is displayed in the commit history and helps identify who made each commit. It is particularly useful in collaborative projects to track contributions from different developers.

- Email
  - The ***git config --global user.email*** configuration sets the email address that will be associated with your commits in Git. This email is displayed in the commit history alongside your name. It is used to identify the author of the commit and is often linked to your account on platforms like GitHub, GitLab, or Bitbucket to associate commits with your profile.

- Line Endings
  - Line Endings (autocrlf): The core.autocrlf configuration in Git manages how line endings are handled between different operating systems. It is particularly useful when working in cross-platform environments (e.g., Windows and Linux). Here's how it works:
    - **autocrlf = true**: Converts line endings to LF (Linux/Unix style) when committing and back to CRLF (Windows style) when checking out files. This is recommended for Windows users working in cross-platform projects.
    - **autocrlf = input**: Converts line endings to LF when committing but does not modify them when checking out. This is recommended for Linux/macOS users to ensure consistent LF line endings.
    - **autocrlf = false**: Disables automatic line-ending conversion. Git will leave line endings as-is. This is useful if you want to handle line endings manually.

Full List of Git config options can be found at this link. https://git-scm.com/docs/git-config

Line Endings| Example



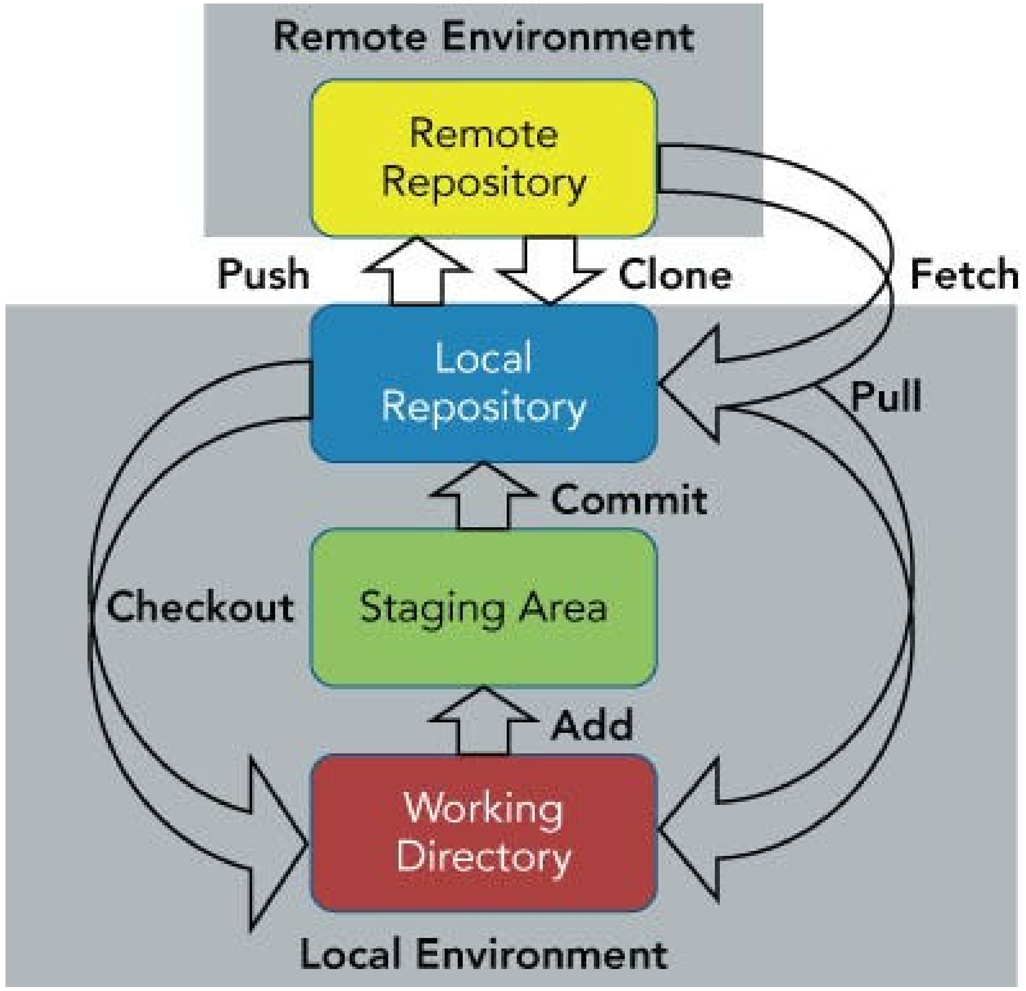
Team uses Windows
and Powershell
core.autoclrf = true

Team uses Linux,
Mac or WSL
primarily
core.autoclrf = input

Git Repo used by both teams.

Git Workflow

- ‘git add <file>’
- ‘git commit’
- ‘git push’
- ‘git clone <repo>’
- ‘git checkout <branch>’



Commit Best Practices

- Commit Often but not too often (5-10 times a day is pretty typical). You should use commit to capture work in progress on features you are working on.
- Commit should contain only 1 thing. If you are adding a new function/feature to your code and discover a bug or issue with something else. You shouldn’t fix the bug as part of your feature commit. Create a second commit for it so it is documented with its own commit and can easily be rolled back.
- Teams should use consisted wording and format for commit messages. This will make code review much easier.
- Commits should contain as much details as needed to explain not just what was changed but why.

Microsoft

Q&A

Lab 01: Introduction to Git

- Work through creating GitHub Repo using CLI
- Cloning Repo down
- Setting up
- Creating First README.md
- Perform “initial commit”
- Create working branch
- Add .gitignore file and commit change
- Create PR and Merge to Main adding .gitignore into main branch.
- Wrap up Q&A

Lab 01: Introduction to Git

Getting started with Docker

Docker Basics

Lab: Docker

Configuration Management

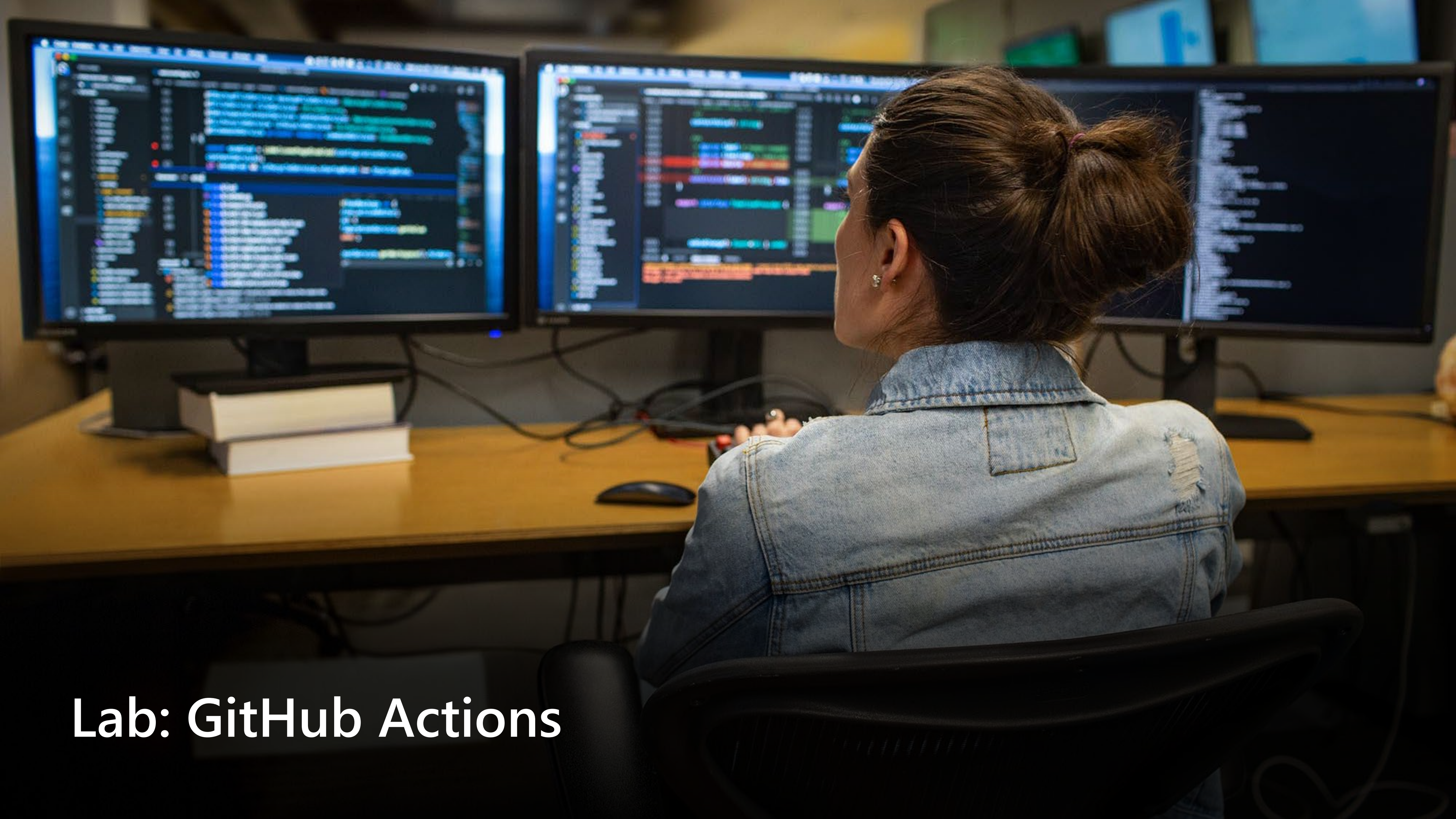
Config Mgmt

Lab: Config Mgmt

Automation with GitHub Actions

CI/CD Pipelines

GitHub Actions



Lab: GitHub Actions

Microsoft

Q&A

Microsoft

Thank you