# Full CI/CD

DevOps Advanced

# Agenda

DevOps Advanced

- Source Control
- Continuous Integration
- Continuous Deployment
- Full CI/CD Sample

# Source Control

Full CI/CD

# Source Control

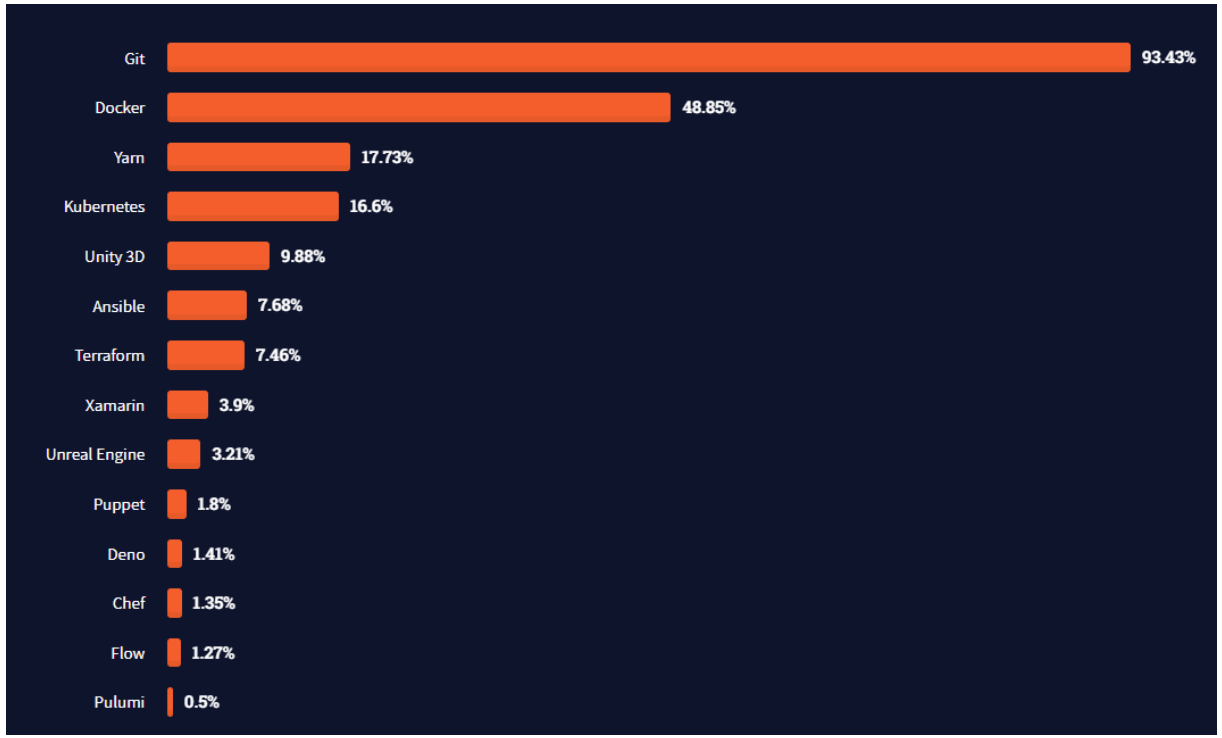
- Source Control as a huge impact on DevOps
- Everything as Code is a reality: Source code, pipelines, infra, docs, …
- Single source of truth
- To have a clear, updated and full picture of our solution we may rely on our source control

# Source Control Benefits

- Full history of all the files (of all your system)
- Branching
- Simultaneous Development
- Auditing
- Allow the implementation of different automation strategies
- Promote collaboration
- Resiliency (remote servers)

# Git as King

- Centralized: Subversion, TFV
- Distributed: Mercurial, Git
- Git is by far the most used source control
- Source: Stack Overflow Survey 2021, 76,253 responses
- After 2021, Git was excluded from this question :)



# Branching Strategies

- Branching strategy means the way you handle and manage branches on remote repo
- This definition is crucial to have a clean and stable remote repo
- Needs to be explained and accepted by all team members in order to make it happen
- Have a crucial role on CI/CD automation processes
- On local repos, each one may have different and their own strategy but need to follow remote strategy every time wants to propagate (push) its changes

# Trunk Based Development

- Use a trunk (main) branch where you have always a buildable and deployable version
- Every new development starts from this branch with a creation of a new branch
- At some point in time, several branches may exist with different developments
- When reaching a stable development, a PR must be open to start merging process
- You must never change directly your trunk branch
- New branches may follow a topic (short-lived branches) or feature (long-lived branches) strategy

# Branching Best Practices

- Define a naming strategy
  - Group similar branches (bugfix, topic, feature)
  - Define different policies for each branch type
  - May use personal branches identifying them with user identifier
- Always define a flow that match your team skills, size and maturity
  - Start with trunk-based development
  - Keep your flow simple and add complexity only if needed
  - Always develop using branches

# Pull Requests

- Pull requests is not a Git feature but instead a feature very popular on all Git Service Providers (GitHub, GitLab, Azure DevOps, …)
- Feature created by GitHub to make easier code review within (distributed) teams
- Pull requests exists to enable peer reviews bringing more quality to all produced code
- Each team element can bring its own experience, knowledge and skills to all code developed within the team

# Pull Requests

- At the end of a pull request what you get is a merge between two branches
- Should not be used for one element (team lead) validation but can be used to manage which feature/improvement will be part of main version
- With Pull Requests you may protect your main branch and only allow updates after code being reviewed
- Using PRs is a crucial practice to enable collaboration between developers inside a team
- Organizations that want to go “one step further” into innersourcing, PRs are mandatory to handle collaboration between teams inside organization

# Pull Requests: Additional Features

- Better definition of product timeline handling which PRs you want to close
- Ask for review/approval from 1+ team elements
- Make mandatory to have a reference to a work item
- Automatic execution of a CI pipeline to test build phase before merge
- Able to have CD pipeline integrated to make functional testing

# Continuous Integration

Full CI/CD

# What is Continuous Integration?

- CI is a mindset, team strategy and a capability. Is not a tool!
- Software development practice where members of a team integrate their work frequently
- Daily integration with verification by an automated build
- This approach leads to significantly reduced integration problems
- Bring much more quality to outcome of the developments

# Goals

- Leverage team collaboration
- Enable parallel development
- Minimize integration debt
- Act as a quality gate
- Automate everything!

# Are you doing it?

- Are all the developers on the team checking into trunk at least once a day?
- Does every change to trunk kick off a build process, including running a set of automated tests to detect regressions?
- When the build and test process fail, does the team fix the build within a few minutes, either by fixing the breakage or by reverting the change that caused the build to break?

# Practices

- Mono-repos vs Multi-repos
- Branching Strategies
- Branching policies
- Pull Requests
- Automated Builds

# Automated Builds

- Coding Standards Checking
- Download Dependent Packages
- Build Code
- Unit Testing
- Code Coverage Analysis
- CredScan
- Static Code Analysis
- Open Source Component Scan
- Create Deployable Package

# Pipeline as Code

- Practice of defining deployment pipelines through source code
- Pipeline as code is part of a larger “as code” movement that includes infrastructure as code, docs as code, etc.
- Teams can configure builds, tests, and deployment in code that is trackable and stored in a centralized source repository.
- Use a declarative language, like YAML (more open), or vendor-specific language (such as Jenkins and Groovy)
- Tries to remove the challenges that UI interfaces bring like limited auditing, hard to versioning, difficult to rollback and prone to break on code changes

# Pipeline as Code: Benefits

- CI pipelines and application code are stored in the same source repository. All the information teams need is located in the same place.
- Developers can make changes without additional permissions and can work in the tools they're already using.
- Teams can collaborate more efficiently. Keeping information accessible means teams can collaborate and then act on their decisions.
- Pipeline changes go through a code review process, avoiding any break in the pipeline integration.

# Continuous Deployment

Full CI/CD

# What is Continuous Deployment?

- Continuous delivery is a software engineering approach in which teams produce software in short cycles, helping to ensure that the software can be released quickly, reliably, at any time, following a repeatable and sustainable process
- It aims at building, testing, and releasing software with greater speed and frequency
- The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production
- A straightforward and repeatable deployment process is important for continuous delivery

# Continuous Deployment vs. Continuous Delivery


Continuous Deployment

MANUAL


Continuous Delivery

AUTO

# DORA metrics

Deployment frequency

Faster lead time for changes

Lower change failure rate

Faster mean time to recover

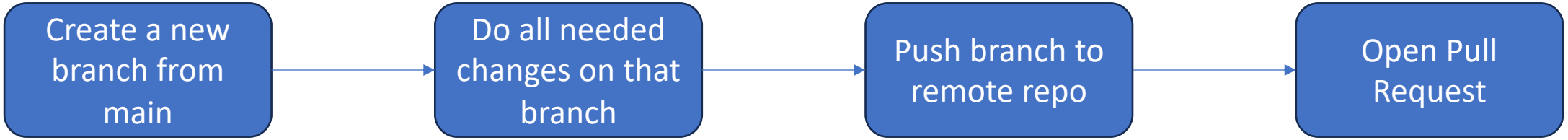
# Practices

- Automated Deployments
- Infra as Code
- Configuration Management as Code
- One Build Only
- Move forward
- Buildable and Deployable Trunk
- Feature Flags
- Shift-Right with Modern Deployments

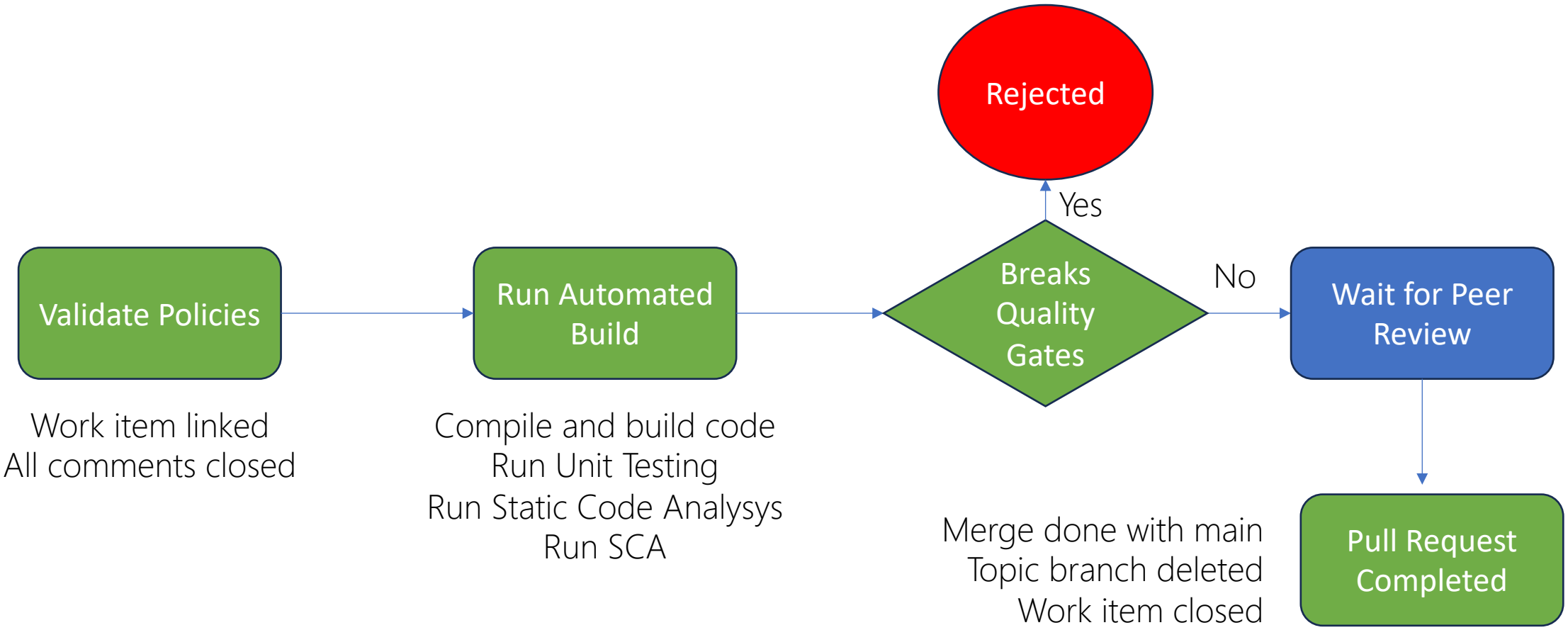
# Full CI/CD Sample

Full CI/CD

# A new request is recevied (work item)



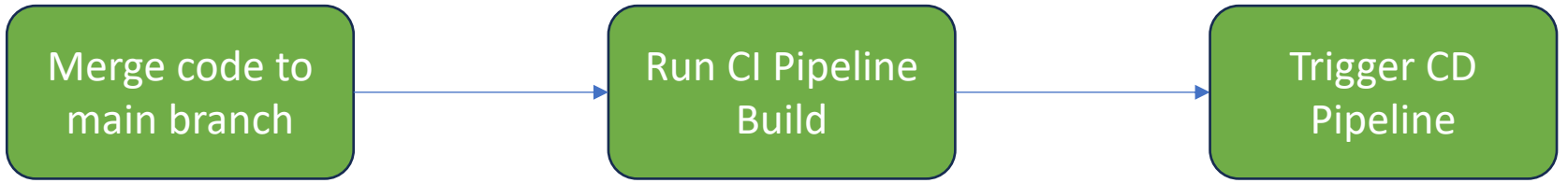
# On your Pull Request…



# On Your Pull Request…

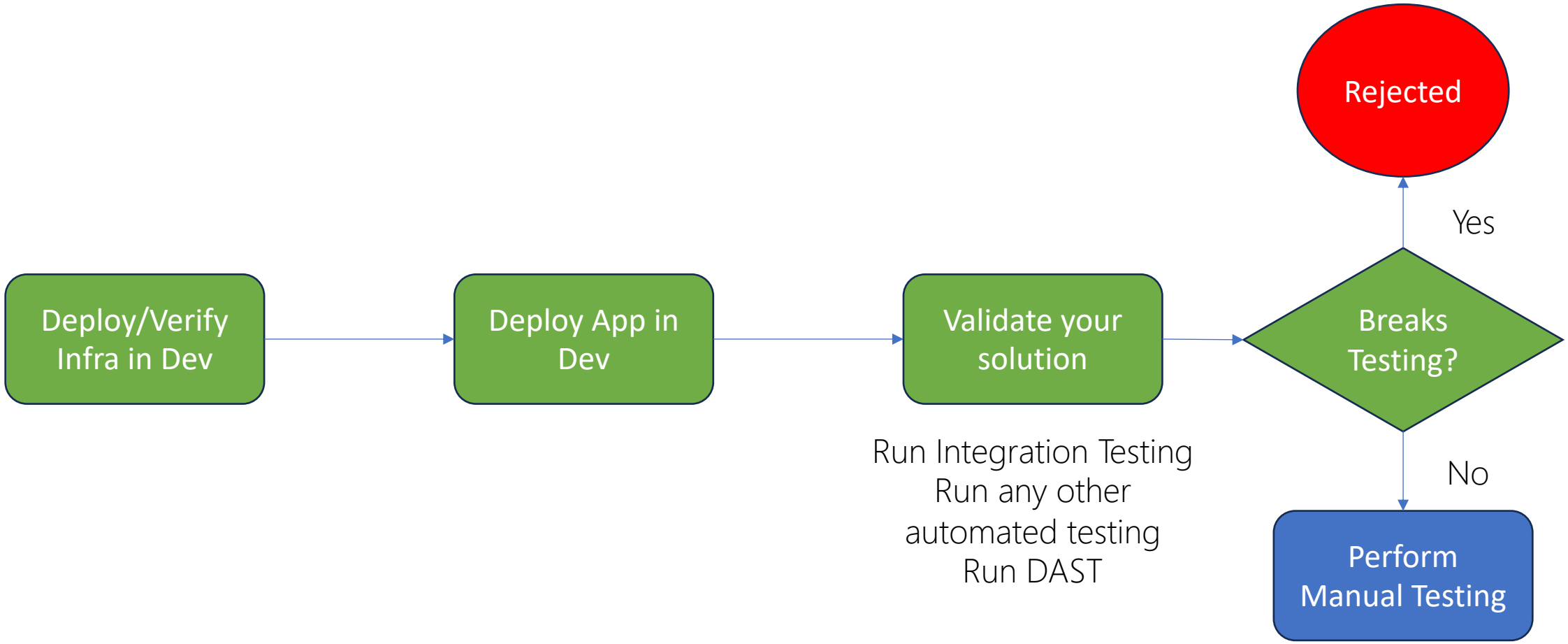
- If possible, you can use disposable environments, to deploy your code
- You can validate your IaC code that creates your environment
- You can run automated integration tests
- You can even make some manual testing and validation, before proceed on your pull request
- Data can be the bigger challenge

# After your pull request being completed…




Compile and build code
Run Unit Testing
Run Static Code Analysys
Run SCA
Create deployable artifact

# With your CD Triggered…



# After running your manual tests…