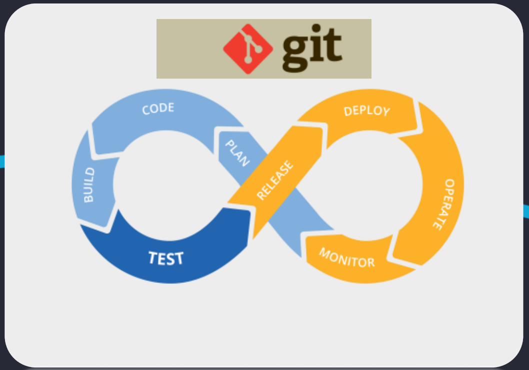
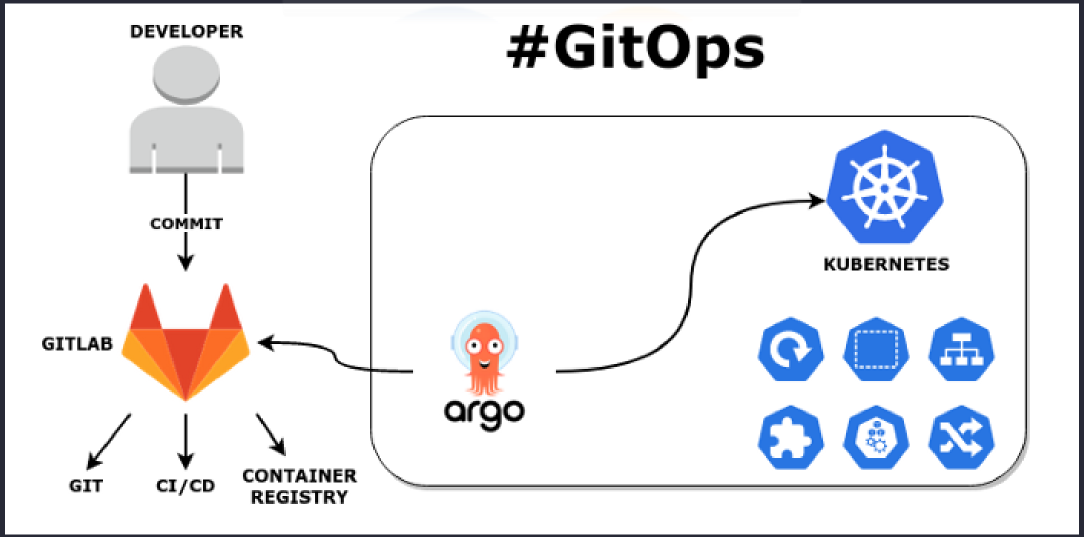




WELCOME TO GITOPS

HOW MANY OF YOU ARE ALREADY FAMILIAR WITH GITOPS AND USING IT?

PLEASE DROP A 'YES' OR 'NO' IN THE CHAT.

AGENDA

- WHAT IS GITOPS
- TRADITIONAL DEPLOYMENT
- WHY DO WE NEED GITOPS
- KEY PRINCIPLES OF GITOPS
- TRADITIONAL VS GITOPS WORKFLOW
- CORE COMPONENTS OF GITOPS ARCHITECTURE
- GITOPS REFERENCE ARCHITECTURE
- SETTING UP GITOPS ENVIRONMENT
- GITOPS ARCHITECTURE BEST PRACTICES
- GITOPS BEYOND KUBERNETES
- KEY TAKEAWAYS
- Q & A

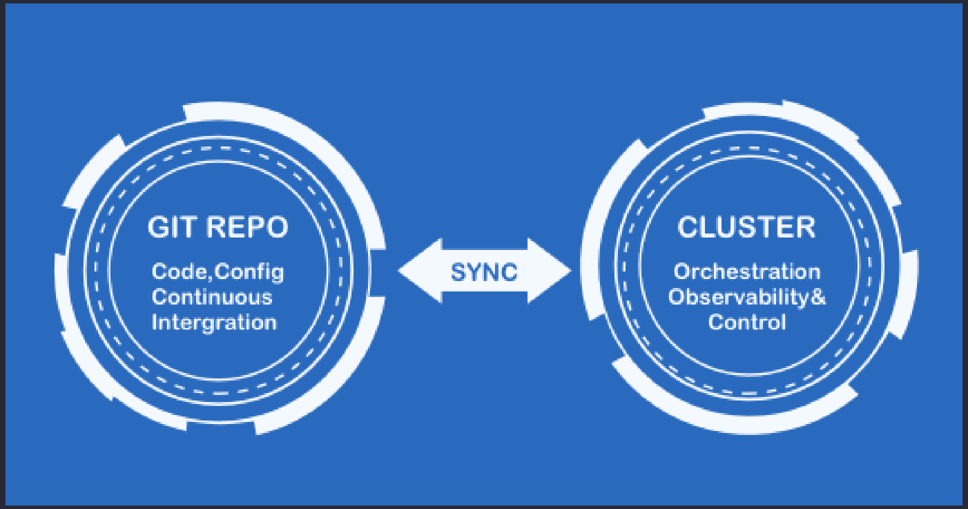
WHAT IS GITOPS

GITOPS IS A MODERN APPROACH TO MANAGING INFRASTRUCTURE AND APPLICATIONS USING GIT AS THE SINGLE SOURCE OF TRUTH. IT BRINGS DEVOPS PRINCIPLES TO INFRASTRUCTURE AUTOMATION, ENABLING CONSISTENCY AND RELIABILITY.

THINK OF GITOPS AS USING GIT TO AUTOMATE YOUR DEPLOYMENT PROCESSES. INSTEAD OF MANUALLY UPDATING CONFIGURATIONS, YOU COMMIT CHANGES TO GIT, AND AN AUTOMATED SYSTEM ENSURES YOUR ENVIRONMENT MATCHES THE STATE DEFINED IN GIT.

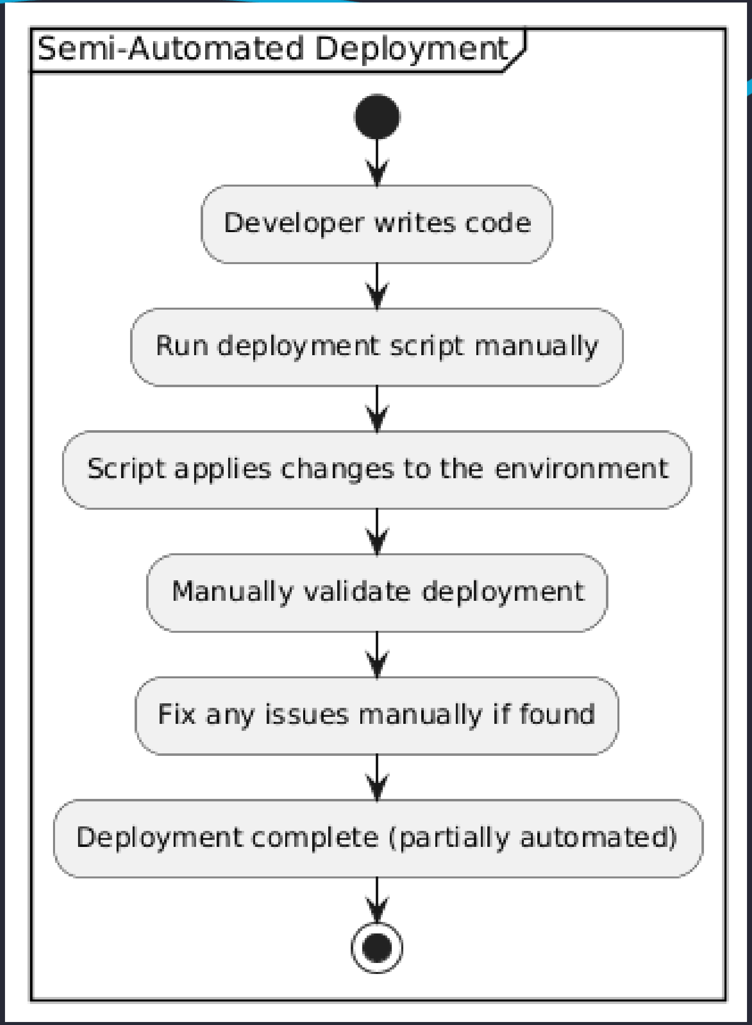
WHY THE NAME "GitOps

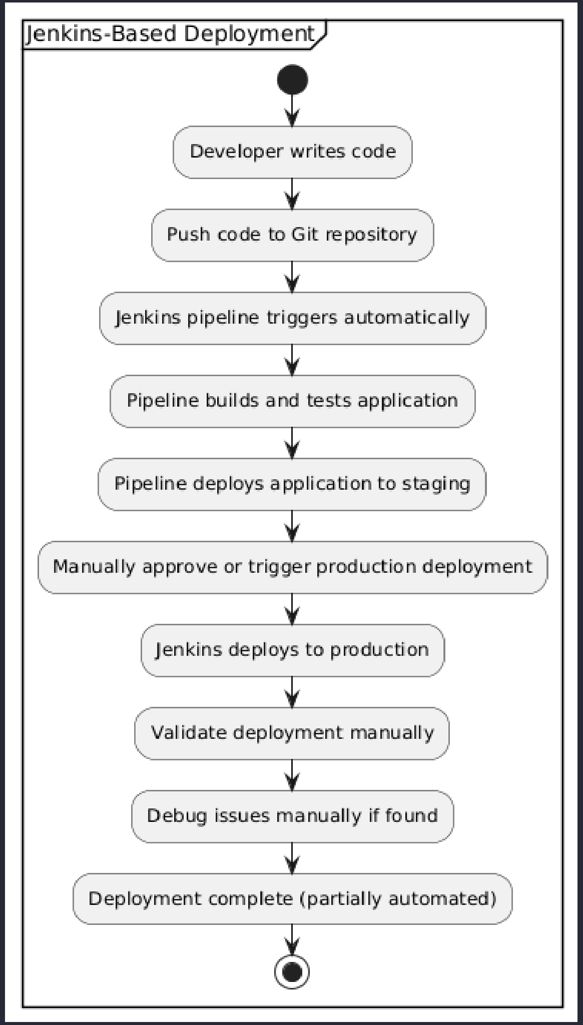
"GIT" BECAUSE IT USES GIT REPOSITORIES



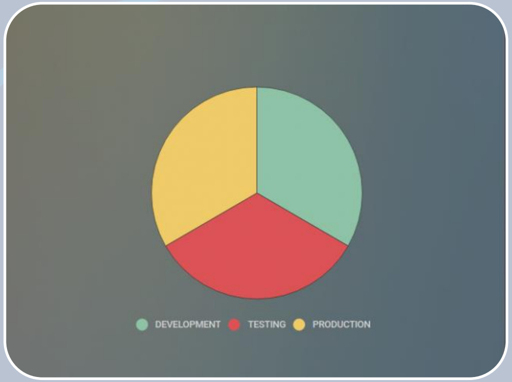
"OPS" BECAUSE IT APPLIES TO OPERATIONS LIKE DEPLOYMENT AND UPDATES.

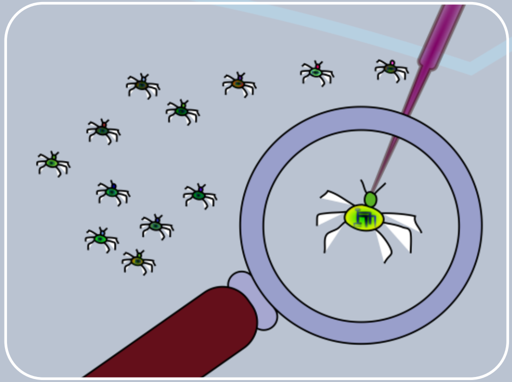
TRADITIONAL DEPLOYMENT WORKFLOWS

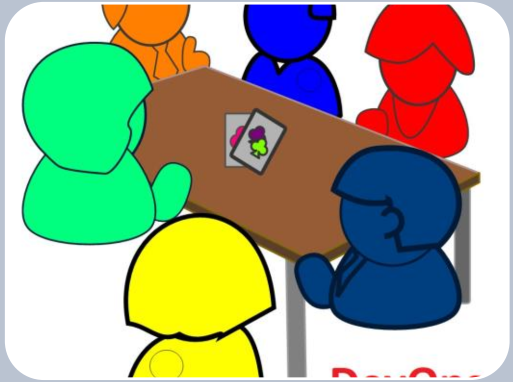

Semi-Automated Deployment
Developer writes code
Run deployment script manually
Script applies changes to the environment
Manually validate deployment
Fix any issues manually if found
Deployment complete (partially automated)


Jenkins-Based Deployment
Developer writes code
Push code to Git repository
Jenkins pipeline triggers automatically
Pipeline builds and tests application
Pipeline deploys application to staging
Manually approve or trigger production deployment
Jenkins deploys to production
Validate deployment manually
Debug issues manually if found
Deployment complete (partially automated)

WHY DO WE NEED GITOPS







Inconsistent Environment

We often have setups that have separate scripts, tools, or manual processes for different environments (development, staging, production).

Debugging & Troubleshooting Complexity

When issues occur in production, it can be challenging to trace back to the root cause because in traditional methods may not always have a single source of truth. Debugging could involve searching through various logs, build scripts, and manual changes made over time.

Manual Processes Sneaking In

Although we try to have automation, manual interventions are still common, especially in the following scenarios: Emergency Fixes, Configuration Updates

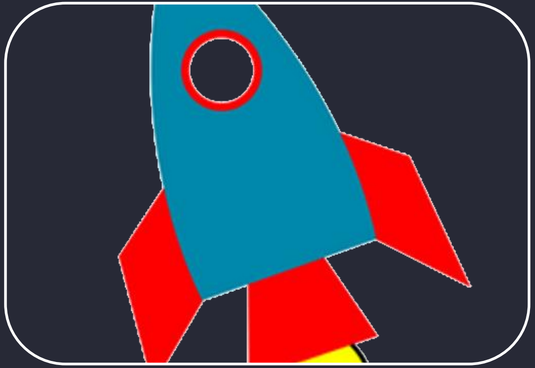
Deployment Delays & Coordination Challenges

In traditional workflows, deployments might require coordination across multiple teams. Even with automated pipelines, deploying new changes often involves review processes, manual approvals, or intervention in case of failures.

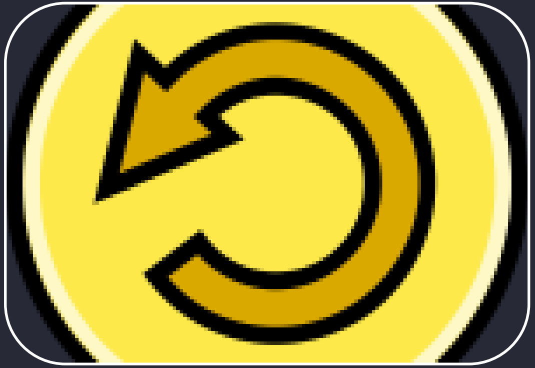
KEY PRINCIPLES OF GITOPS

✓	✓	✓	✓
Declarative Configuration	**Version Control as Source of Truth**	**Automated Deployments**	**Continuous Reconcilliation**
• All infrastructure and application configurations are defined in declarative files, like YAML or JSON.	• Git is the source of truth, tracking changes to infrastructure and application definitions.	• Changes in Git automatically trigger deployment processes using CI/CD pipelines	• An operator constantly monitors the environment to ensure it matches the state defined in Git, reconciling any drift

BENEFITS OF GITOPS



Faster Deployments



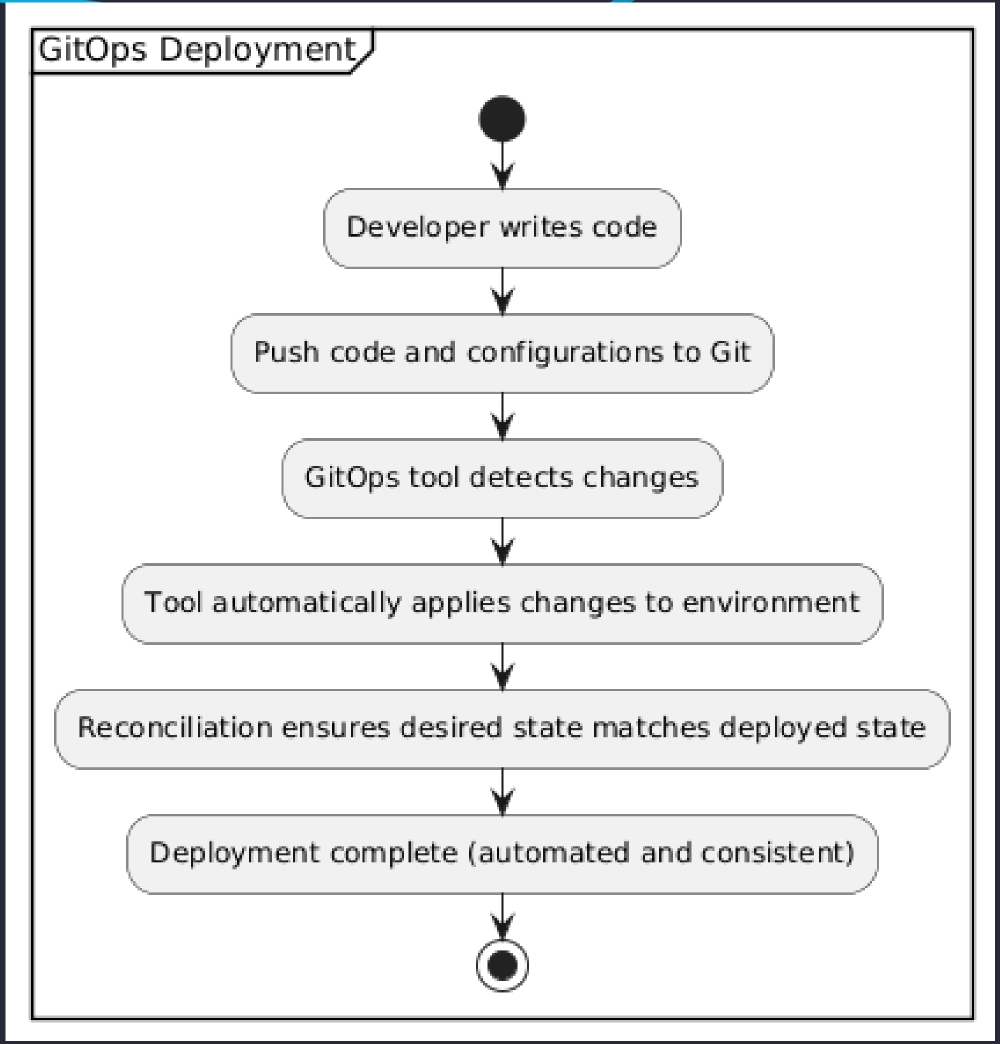
Easy Rollback

Auditability

Collaboration

IT ALSO BRINGS RELIABILITY BY KEEPING YOUR INFRASTRUCTURE CONSISTENT.

GITOPS DEPLOYMENT WORKFLOW


GitOps Deployment
Developer writes code
Push code and configurations to Git
GitOps tool detects changes
Tool automatically applies changes to environment
Reconciliation ensures desired state matches deployed state
Deployment complete (automated and consistent)

CORE COMPONENTS OF GITOPS

The Git repository acts as the source of truth. It stores all the configuration files, infrastructure definitions, and deployment manifests.

GitOps Repository

1

GitHub Bitbucket GitLab

CI/CD Pipelines

2

Actions CI/CD Jenkins

Automate deployment using CI/CD pipelines. They monitor changes in the repository and trigger the necessary actions.

Tools like ArgoCD and Flux are commonly used. They monitor Git repositories for changes and synchronize them with the live environment.

3

GitOps Operator/Agent

argo flux

4

Monitoring/Alerting Tools

Prometheus Grafana

Monitoring tools like Prometheus and Grafana ensure that the actual state is consistent with the desired state defined in Git. If there's a drift, alerts can notify the team.

GITOPS REFERENCE ARCHITECTURE

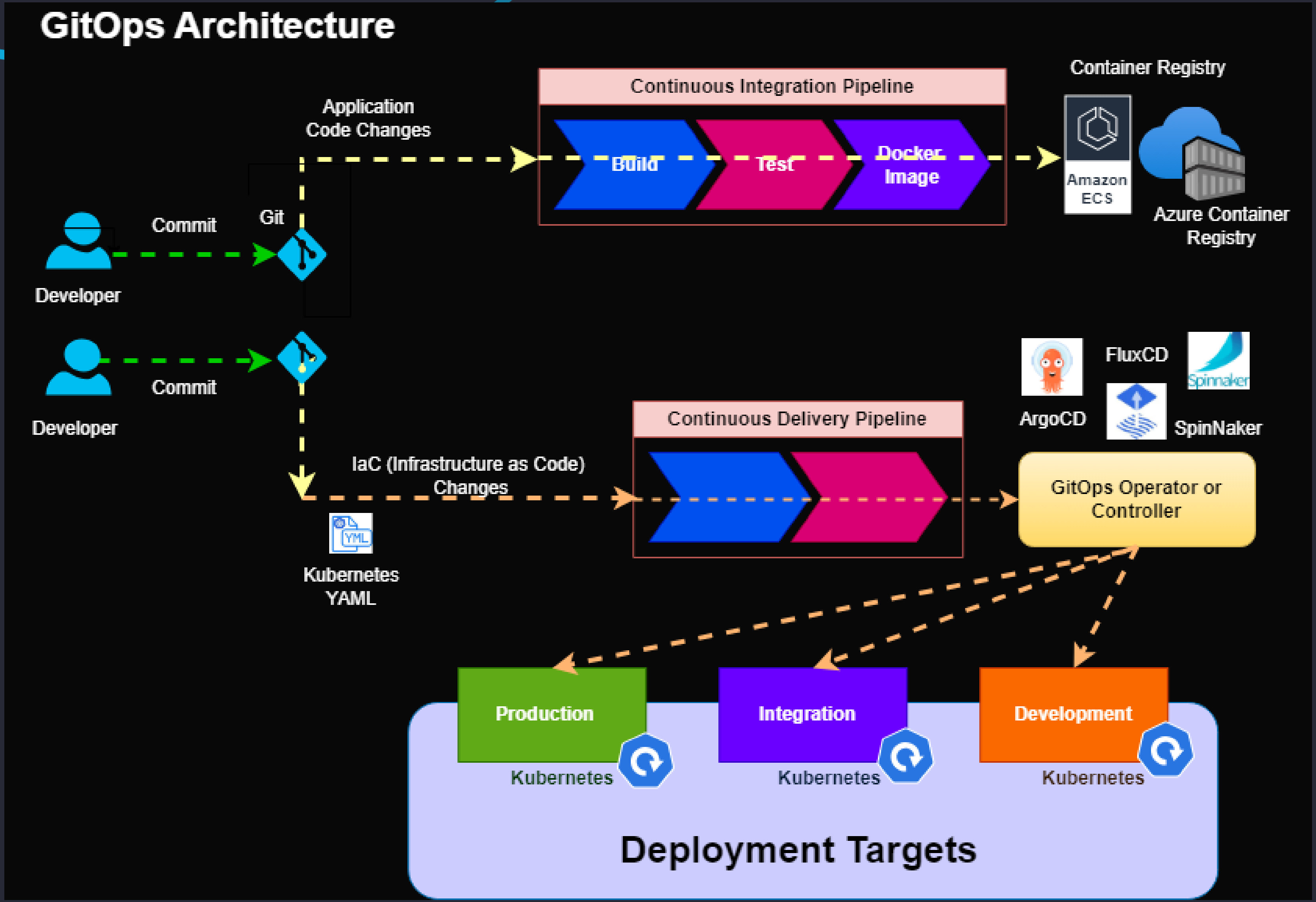
PUSH

In a more traditional setup, you might build a trigger into your pipeline so it executes a command to push your desired infrastructure state into production. (Usually, with Jenkins or a CI/CD tool.)
Push based approach need to expose K8S credentials to CI/CD tool to be able to allow it to make modifications.

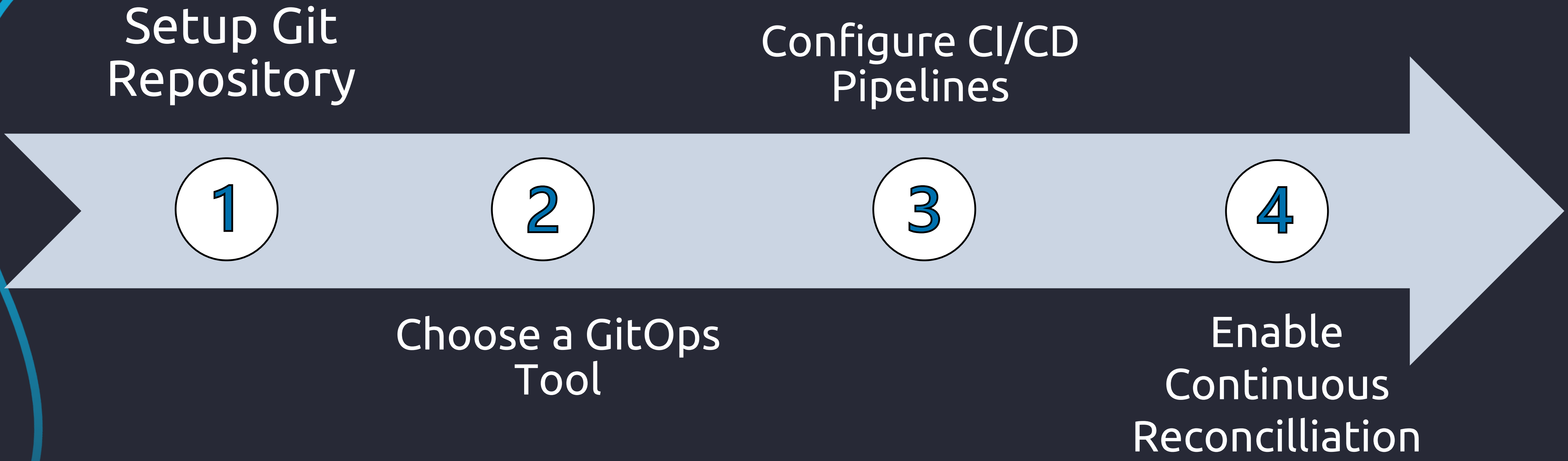
PULL

More resilient way to update your cluster is to install an agent like Argo CD to actively monitor and resolve differences between the desired state in Git and the actual cluster state.
Why

1. Scale
2. Version Control



SETTING UP GITOPS ENVIRONMENT



SETTING UP GITOPS ENVIRONMENT

Setup the GIT Repository

- Create a Git repository with folders for different environments
- Store the configuration files for infrastructure and applications

Choose a GitOps Tool

- Select a GitOps tool based on your requirements.
- ArgoCD is popular for its web interface, while Flux is lightweight and Kubernetes-native

SETTING UP GITOPS ENVIRONMENT

Configure CI/CD Pipeline

- Set up CI/CD pipelines to automate deployments
- Include automated tests in the pipeline to catch configuration errors before deployment

Enable Continuous Reconciliati on

- Configure the GitOps tool to continuously reconcile the cluster state with the Git repository
- Set up alerts for any detected drifts

GITOPS BEST PRACTICES

Adopt a Multi-Repository Strategy

- Consider using multiple repositories for different components of your infrastructure and applications.
- This helps manage complexity, improve security, and enable more fine-grained control over changes.

Security Considerations

- Store secrets securely using tools like Sealed Secrets
- Avoid committing sensitive information directly to the repository

Automate Testing

- Add configuration validation steps in your pipeline
- Use tools like kubeval to check Kubernetes manifests

Use declarative tooling

- Emphasize the use of declarative tools like Helm for Kubernetes applications and Terraform for IaC (Infrastructure as a code)

Monitor drift and reconcile frequently

- Use tools like Argo CD or Flux to continuously monitor for drift.
- Ensure frequent reconciliation to maintain consistency and detect issues early.

Rollback Strategies

- Leverage Git history for quick rollbacks
- Configure automated rollback in your pipeline for failed deployments

GITOPS BEYOND KUBERNETES

WHILE GITOPS IS MOST COMMONLY ASSOCIATED WITH KUBERNETES, ITS PRINCIPLES CAN BE APPLIED TO OTHER AREAS OF INFRASTRUCTURE AND APPLICATION MANAGEMENT. HERE ARE SOME USE CASES AND EXAMPLES WHERE GITOPS IS EFFECTIVELY USED BEYOND KUBERNETES.

Cloud Infrastructure Management

- Use GitOps to manage cloud infrastructure provisioning and updates by storing Terraform or CloudFormation templates in a Git repository.

Application Configuration Management

- GitOps can be used to manage application configurations stored in files like .properties, .env, or JSON/YAML configurations.

GITOPS BEYOND KUBERNETES

Serverless Deployments

- Use GitOps to automate serverless deployments by storing function code (AWS Lambda, Azure Functions) and deployment configurations in Git.

Database Schema Management

- Store database migration scripts in a Git repository and use a GitOps approach to automate the execution of migrations when changes are committed.
- Tools like Flyway or Liquibase can be integrated into the pipeline to apply migrations to the target databases.

FEW FACTS ABOUT GITOPS

GitOps Adoption and Popularity

- According to a 2023 industry survey, around 68% of organizations that use Kubernetes for container orchestration have either adopted GitOps or are planning to adopt it within the next 12 months.
- GitOps is considered a top DevOps trend, with increased adoption driven by the shift toward cloud-native and Kubernetes environments.

GitOps and Kubernetes

- GitOps is most commonly associated with Kubernetes, where it provides a natural fit for managing declarative infrastructure.
- Tools like ArgoCD and Flux have grown significantly, with ArgoCD reaching over 11,000 stars on GitHub and being adopted by top companies.

Benefits of GitOps

- GitOps can reduce deployment times by 2-3 times.
- With GitOps, recovery time after a failed deployment can be reduced by 60%, thanks to easy rollbacks using Git's version history.

GitOps Security and Compliance

- GitOps inherently improves security by storing all changes in Git, providing an auditable history of who made changes and when. This aligns with best practices for compliance frameworks like SOC 2.
- By keeping secrets and configurations in version control (with proper encryption), GitOps enables end-to-end traceability and allows teams to manage security policies as code.

FEW FACTS ABOUT GITOPS

Challenges Addressed by GitOps

- 70% of DevOps teams report issues with environment drift, where configurations differ between staging, production, and development environments. GitOps solves this by automatically reconciling the desired state.

- Manual configuration errors contribute to over 50% of deployment failures, and GitOps addresses this by ensuring all changes go through a version-controlled workflow.

GitOps Tools Growth and Ecosystem

- GitOps tools are expanding rapidly, with dozens of open-source projects supporting various aspects of GitOps workflows, such as deployment, monitoring, and secret management.

- Companies like Weaveworks, Intuit, and Red Hat are major contributors to GitOps tools, fostering a strong and growing ecosystem.

GITOPS CHALLENGES

- Automating Git commits may create conflicts - Git supports manual conflict resolution and editing, not automatic updates.
- Too many Git repositories - The number of GitOps repositories typically increases with each new environment or application, increasing the repo management burden.
- Limited visibility - Enterprise environments often have many GitOps repositories and config files, making it impractical to comb through plain text files.
- Doesn’t centrally Manage secrets - A Git repository is not suitable for storing secrets requiring encryption and decryption.
- Auditing may be insufficient - it is harder to answer other questions with a Git repo, such as the number of times an application was deployed.

KEY TAKEAWAYS

GitOps Extends DevOps Principles

GitOps builds on DevOps practices by managing infrastructure and application changes through Git, providing a unified, declarative, and automated approach to deployment.

Benefits of GitOps

Improved consistency across environments, faster deployments, easier rollbacks, and enhanced security through version-controlled infrastructure.

Real-World Use Cases Beyond Kubernetes

GitOps is not limited to Kubernetes. It can be applied to cloud infrastructure management, serverless functions, database schema updates, network configuration, and more.

Git as the Single Source of Truth

All infrastructure and application changes are stored in Git, allowing for easy auditing, traceability, and rollbacks.

KEY TAKEAWAYS

Getting Started with GitOps

Start small: Use GitOps for a single service or environment before expanding.

Adopt tools like ArgoCD, Flux, or others to automate reconciliation.

Apply GitOps principles to existing CI/CD pipelines to leverage current workflows.

Challenges and Considerations

GitOps is powerful but requires careful management of Git repositories, secrets, and automation tools. Consider potential pitfalls like handling secrets securely and managing large monorepos.

Thank you for joining the session!

Questions?