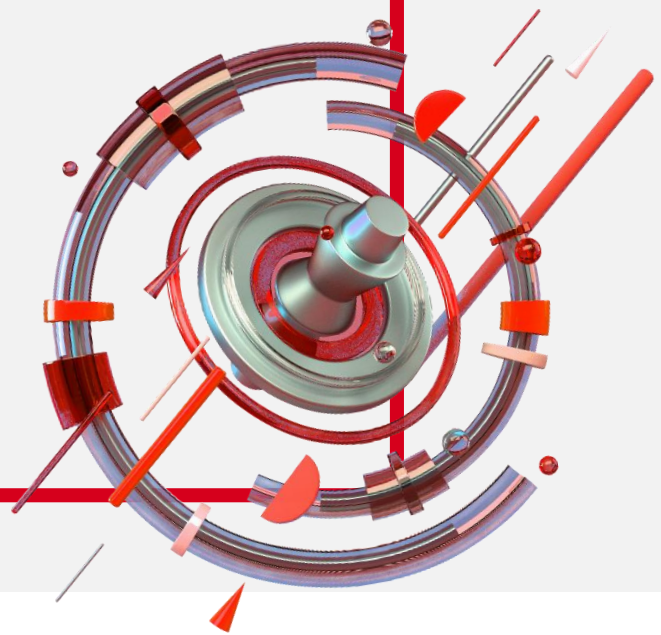
GitOps Fundamentals With Argo CD



Cao Thi Hoang Le

July 2023

Nash Tech.

Agenda

Understanding GitOps.

- What GitOps is.
- Getting Hooked on GitOps
- GitOps with Kubernetes
- Traditional workflow deployments for Kubernetes
- Challenges of traditional workflow deployment
- How GitOps Helps
- GitOps workflow deployments for Kubernetes
- GitOps Use Cases.
- GitOps Operators.
- GitOps Decision Points.

Agenda

Progressive Delivery

- What is Progressive Delivery?
  - Blue/Green Deployment.
  - Canary Deployment.
- Introduction to Argo Rollouts.
- Blue/Green with Argo Rollouts.
- Canaries with Argo Rollouts.
- Automated rollbacks with metrics.

Agenda

Argo CD

- Introduction to Argo CD
- GitOps workflow with Argo CD
- Argo CD installation.
- Creating an application.
- Sync Strategies.

Agenda

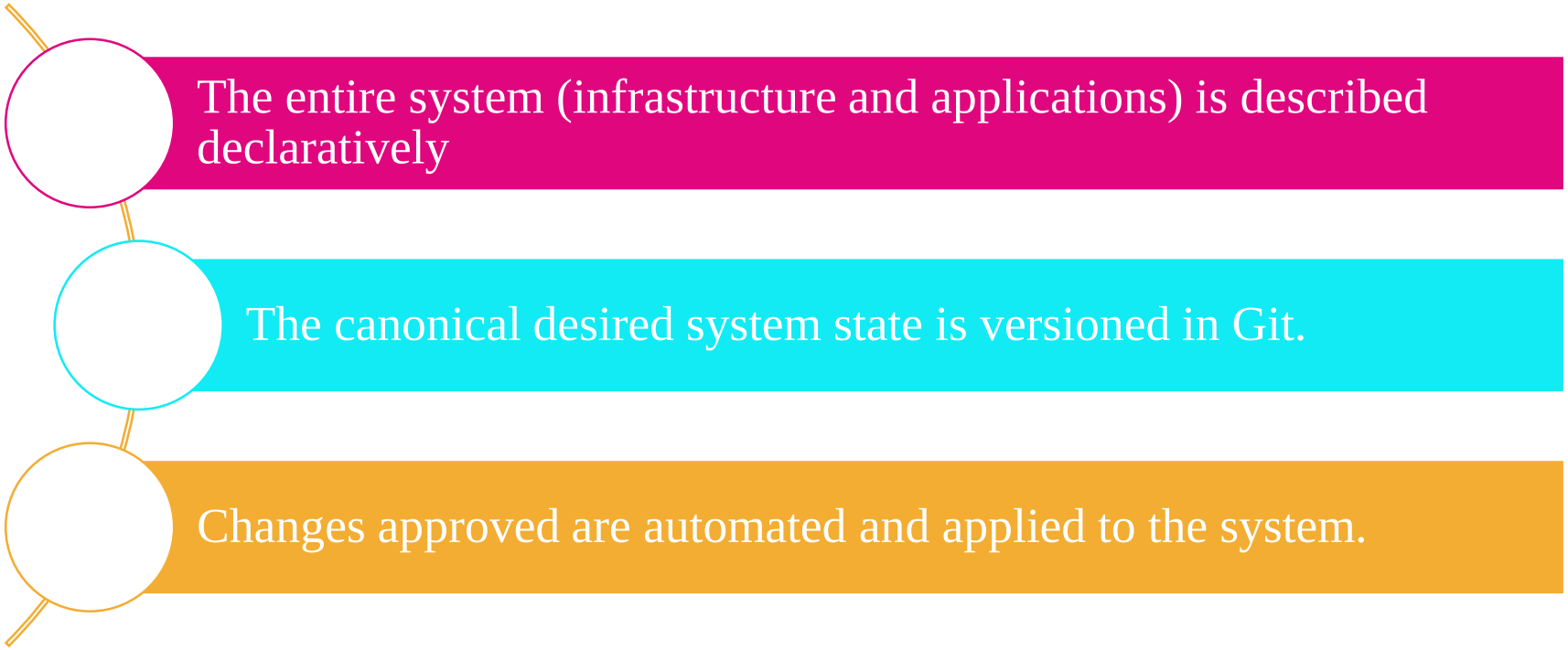
Demo

01

Understanding GitOps.

What GitOps is.

- GitOps is an operating model pattern, a set of best practices where the entire code delivery process is controlled via Git, including infrastructure and application definition as code and automation to complete updates and rollbacks.
- The Key GitOps Principles:



GitOps with Kubernetes

GitOps treats each software or infrastructure component as one or more files located in a version control system, with an automated process to synchronize state between version control and the runtime environment

An orchestration system like Kubernetes is very important for achieving this. Without Kubernetes, the overall infrastructure can be too complex to manage, with multiple, incompatible technologies.

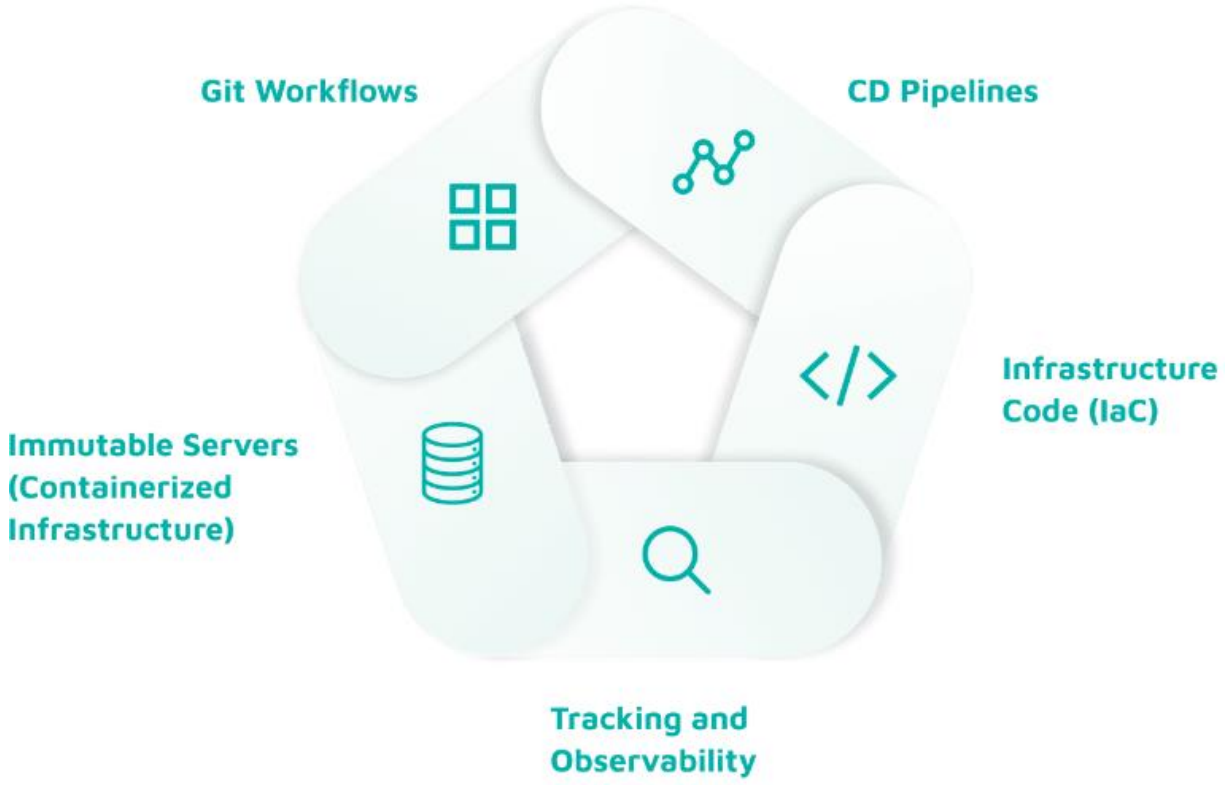
kubernetes

GitOps is not limited to Kubernetes. You can use GitOps with any system that can be observed and described declaratively. Currently the majority of pull-based GitOps operators were built for Kubernetes.

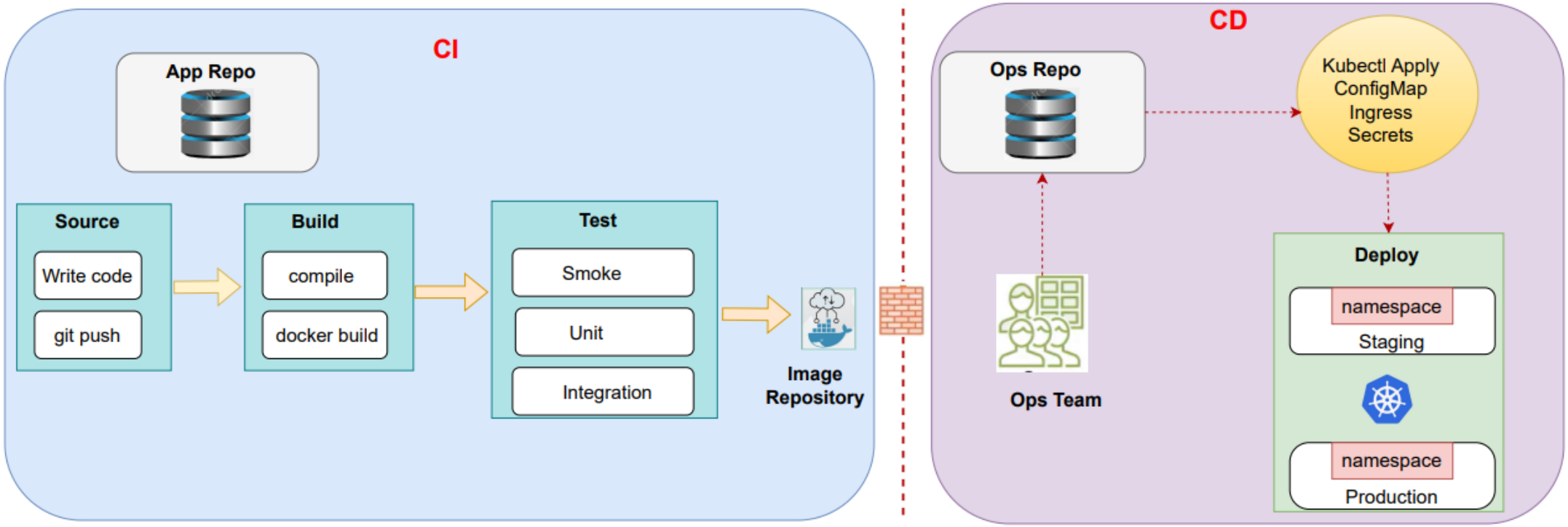
If my project does not have Kubernetes. Does GitOps still apply to?

Getting Hooked on GitOps

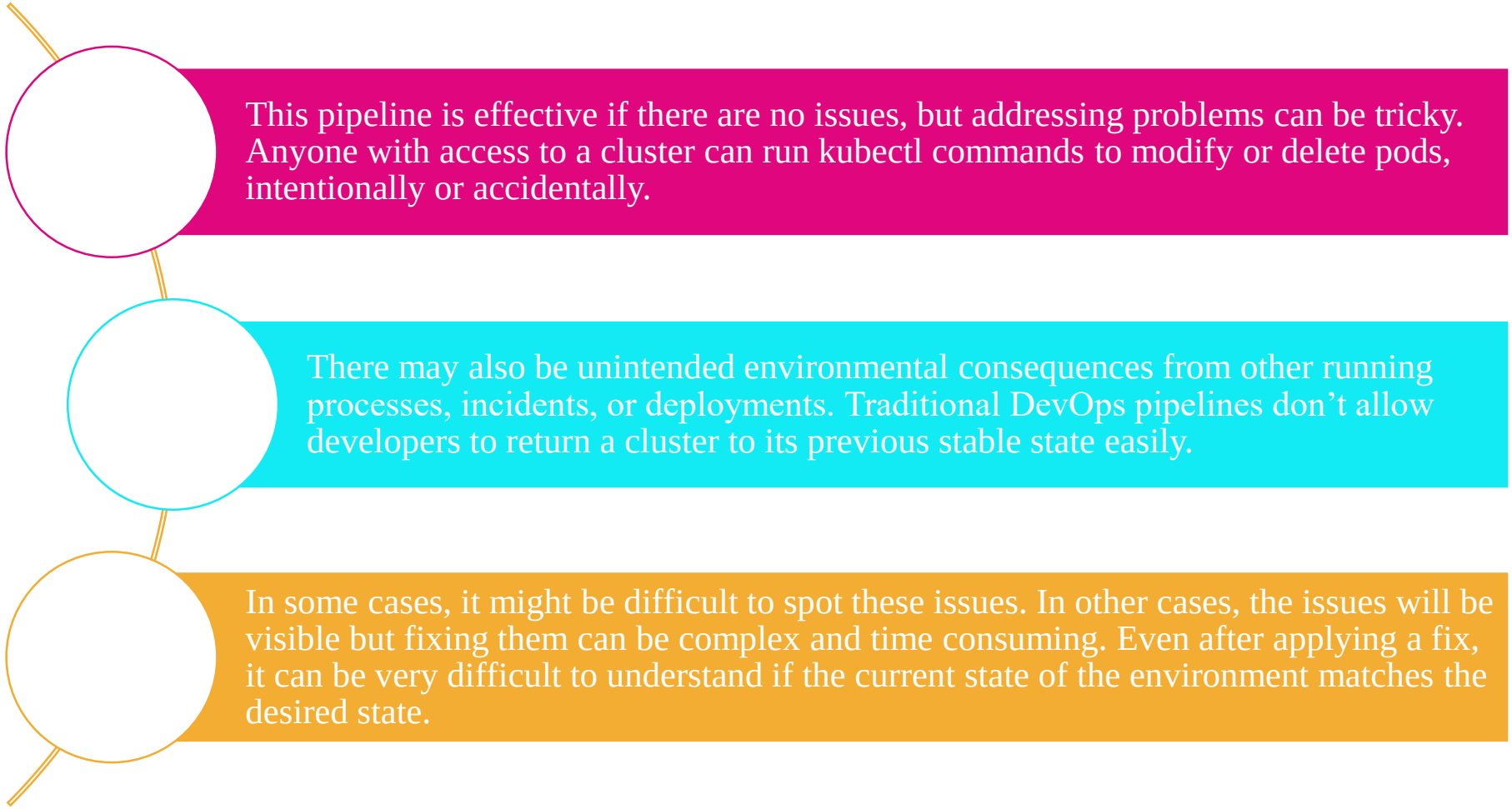
- GitOps is a model developed on pre-existing practices



Traditional workflow deployments for Kubernetes



Challenges traditional workflow deployments



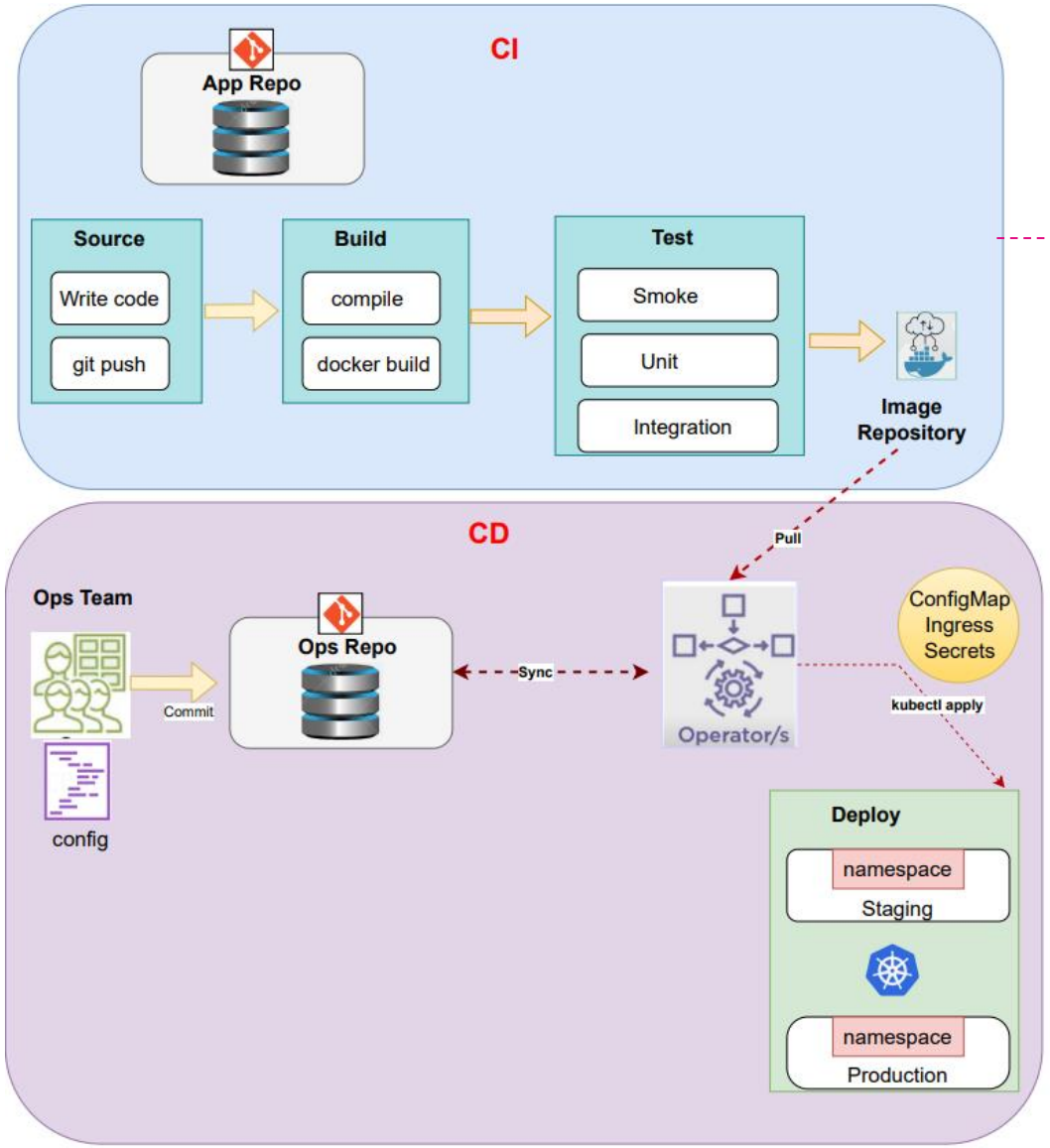
How GitOps helps

GitOps helps address this issue by handling the CD step. The automated CI steps are often the same, a GitOps tool like Argo CD runs in the cluster and keeps track of the manifest files in the code repository, ensuring the cluster remains in sync.

With GitOps, a tool regularly synchronizes the running clusters with the desired target state in the code repository (usually GitHub). GitOps enhances the security of a cluster by separating permissions for CI and CD tasks

In a GitOps pipeline, a tool like Argo regularly compares the cluster state in Kubernetes to the desired state in the manifest files. If someone edits the cluster or deploys pods outside the CI/CD process, the tool detects the changes and can automatically restore the cluster to the target state. Git is the single source or truth, ensuring applications are always in sync.

GitOps workflow deployments for Kubernetes



Until this point, GitOps is identical to a traditional workflow.

1. A deployment automator identifies the change to the image repository and pulls it from the registry.
2. A GitOps agent identifies a change in the cluster and pulls it from the configuration repository, updating the cluster with the relevant changes.

GitOps Use Cases.

Continuous deployment of applications

- Adopting GitOps is easy if you have already adopted Kubernetes and handle applications declaratively. A Git repository should already be set up for your source code.

Continuous deployment of cluster resources

- Once you adopt GitOps for your applications, extending the same principles to supporting applications, managed but not developed by you, is logical. In Kubernetes, these supporting applications include metrics, networking agents, service meshes, databases, and more.
- Applying GitOps to those applications streamlines cluster management and brings GitOps benefits, including easy rollback, to wider audiences beyond the development team.

Continuous deployment of infrastructure

- Adopting GitOps for applications is a great step, and you can extend this paradigm to other infrastructure layers. By defining resources declaratively, it becomes easy to apply GitOps principles to the infrastructure supporting your applications.
- Familiarity with IaC makes transitioning to GitOps easier. Additionally, newer tools like Crossplane enable managing infrastructure similar to Kubernetes applications.

Detecting/Avoiding configuration drift

- GitOps eliminates the need for manual changes with kubectl in Kubernetes. Even if changes are made via kubectl, GitOps agents promptly notify you. You can choose to discard or accept these changes, committing them back to Git as needed.
- This power of detecting configuration drift early is one of the major differences between GitOps and traditional deployment solutions

Multi-cluster deployments

- If you have multiple clusters at different geographical locations or with different configurations, you need a way to track them and understand how they are different.
- This is one of the cases where GitOps really shines. Because the state of all clusters is stored in Git, it is very easy to know what is installed where, who installed it, and when just by looking at the Git history.

GitOps Operator

Cloud Infrastructure

(Infrastructure Automation)

- ✓ Terraform for setup/upgrade/modify
- ✓ Cloud IaaS, IAM, Kubernetes Cluster, Namespaces, Ingress Controller, Service Mesh, Monitoring

Kubestack

Terraform GitOps Framework for building Kubernetes on any platform

https://www.kubestack.com

Jenkins X

Kubernetes pipeline automation with built-in GitOps

https://jenkins-x.io

Cluster

(Application Automation)

- ✓ K8s manifests / helm charts
- ✓ For the application: Deployment, Pods, Services, ConfigMaps, Namespaces etc

Flux

Kubernetes GitOps Operator

https://fluxcd.io

Argo CD

Kubernetes GitOps Operator with visual approach

https://argoproj.github.io/argo-cd

GitOps Decision Points.

Where to host Git Repository?

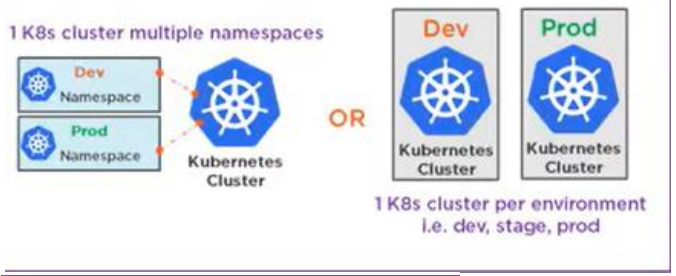
Azure DevOps, GitHub, GitLab, Bit Bucket etc.

Git Repository structure?

level of separation for repositories

i.e. App and Config separate repos, a repo per team, repo per environment etc.

Runtime (K8s) Environment structure?



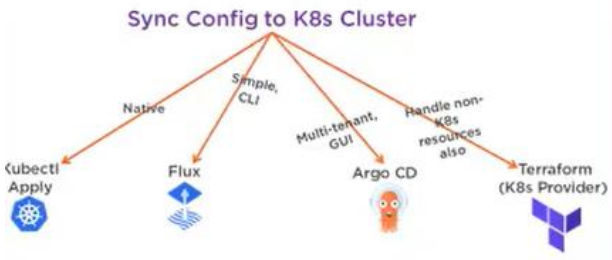
Namespace structure?

Namespace - per environment (dev/prod), per app, service, per engineer, per build, ect.

Operator to use?

Flux, ArgoCD, Kubectl apply, Terraform K8s provider, Jenkins X, etc.

Sync Method?



02

Progressive Delivery

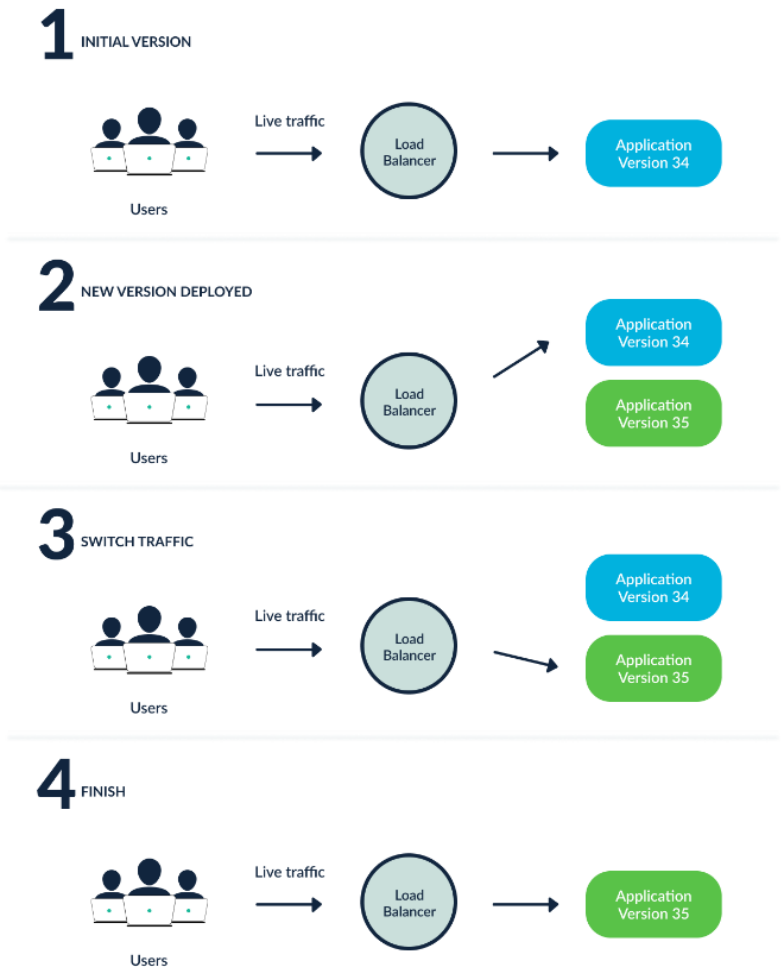
What is Progressive Delivery

- Progressive Delivery is the practice of deploying an application in a gradual manner allowing for minimum downtime and easy rollbacks. There are several forms of progressive delivery such as blue/green, canary, a/b and feature flags.

By: Orit Golowinski

What is Progressive Delivery

- **Blue Green Deployments**
  - Blue/Green deployments are one of the simplest ways to minimize deployment downtime.



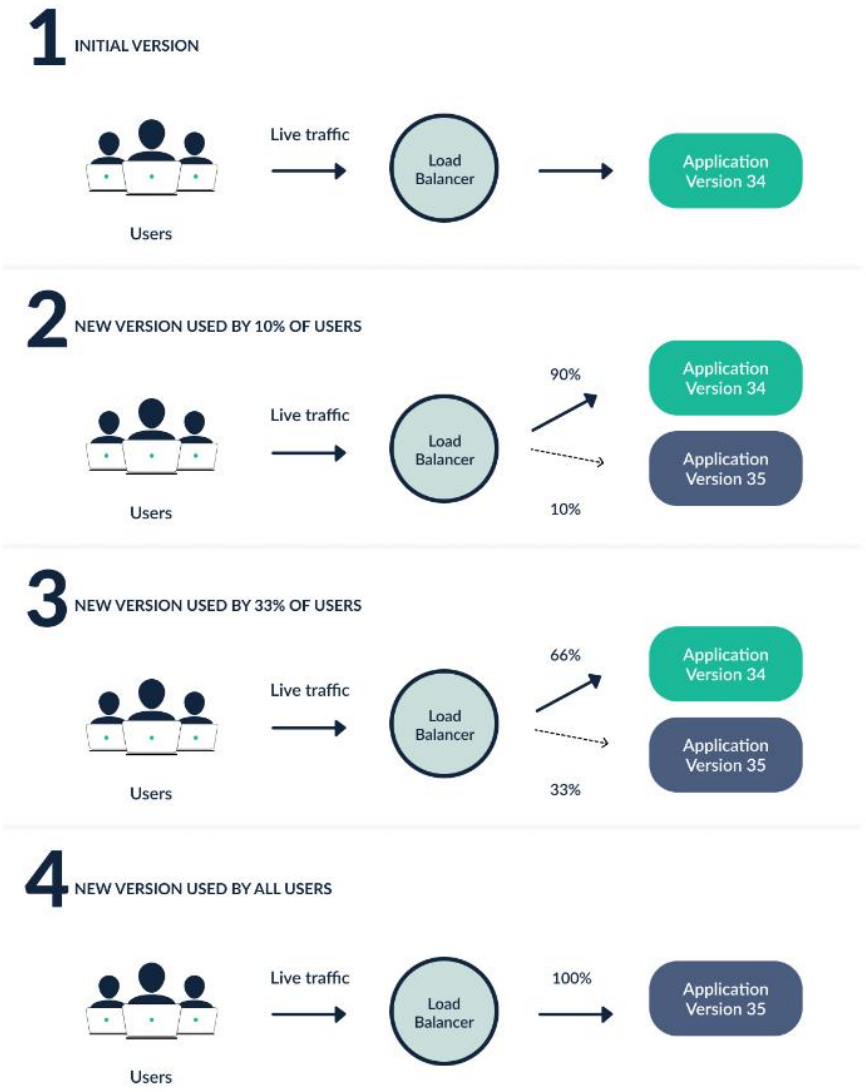
What is Progressive Delivery

- **Canary Deployments**
  - Blue/Green deployments are great for minimizing downtime after a deployment, but they are not perfect. If your new version has a hidden issue that manifests itself only after some time (i.e. it is not detected by your smoke tests), then all your users will be affected because the traffic switch is all or nothing.

  - An improved deployment method is **canary deployments**. This functions like blue/green, but instead of switching 100% of live traffic all at once to the new version, you can instead move only a subset of users.

What is Progressive Delivery

- Canary Deployments



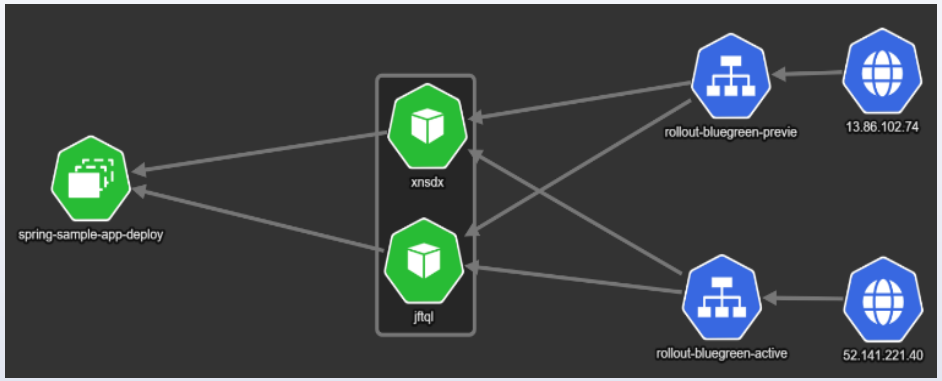
Introduction to Argo Rollouts

- Argo Rollouts is a progressive delivery controller created for Kubernetes. It allows you to deploy your application with minimal/zero downtime by adopting a gradual way of deploying instead of taking an “all at once” approach.
- Argo Rollouts supercharges your Kubernetes cluster and in addition to the rolling updates you can now do:
  - Blue/green deployments
  - Canary deployments
  - A/B tests
  - Automatic rollbacks
  - Integrated Metric analysis

Blue/Green with Argo Rollouts

Initial state of a deployment.
The example uses 2 pods (shown as *xnsdx* and *jftql* in the diagram).

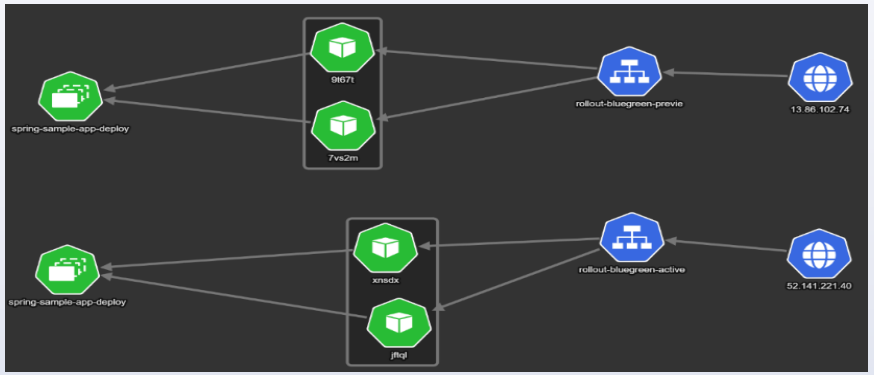
1



Once a deployment starts, a new “color” is created. We have 2 new pods that represent the next version deployed (9t67t and 7vs2m).

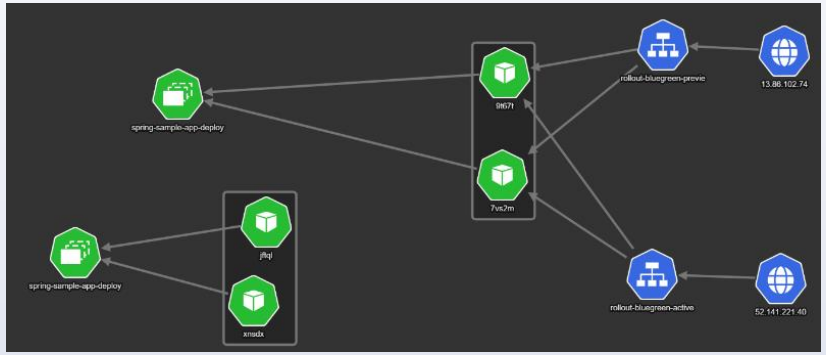
.

2



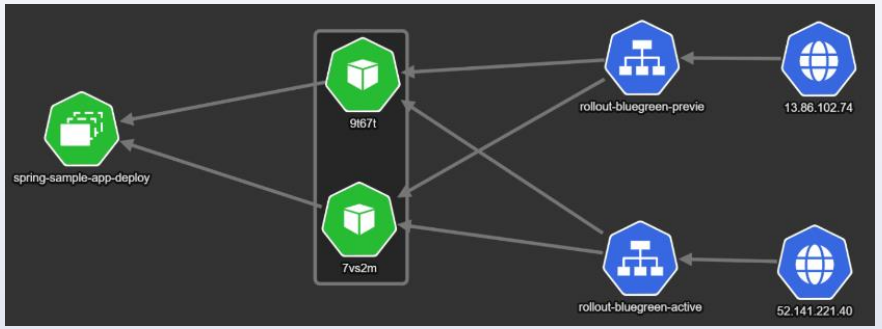
If everything goes well, the next version is promoted to be the active version..

3



Now back to the same configuration as the initial state, and the next deployment will follow the same sequence of events.

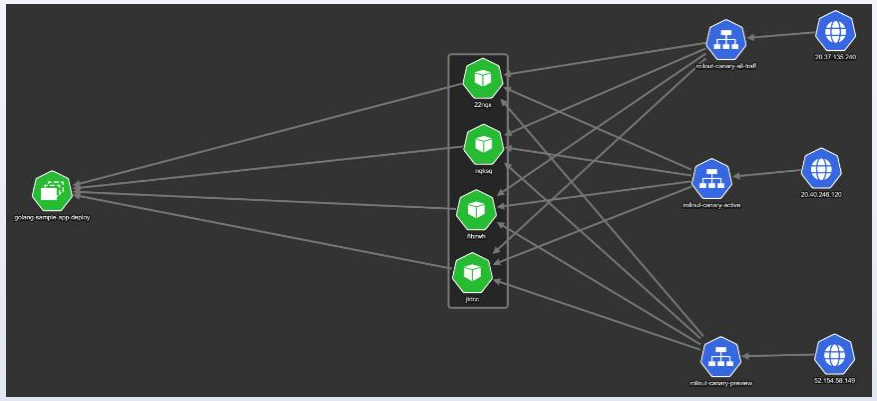
4



Canaries with Argo Rollouts

Initial state of a deployment.
The example uses 4 pods (*22nqx*, *nqksq*, *8bzwh* and *jtdcc*).

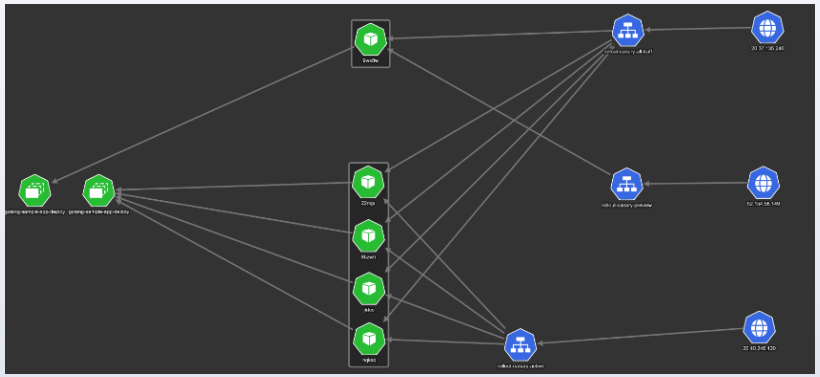
1



Once a deployment starts, a new “color” is created. We have 1 new pod that represent the next version deployed (9wx8w).

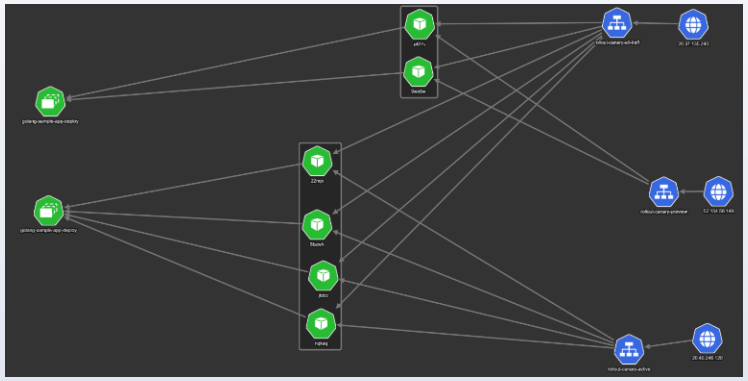
.

2



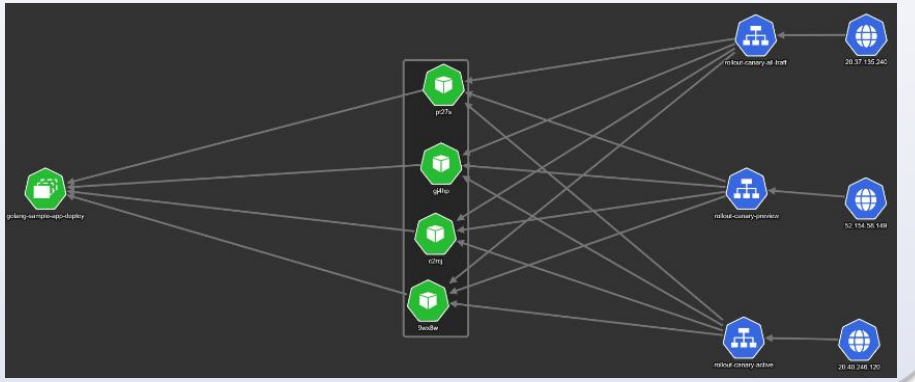
If everything goes well, and you are happy with how the canary works, we can redirect some more traffic to it.

3



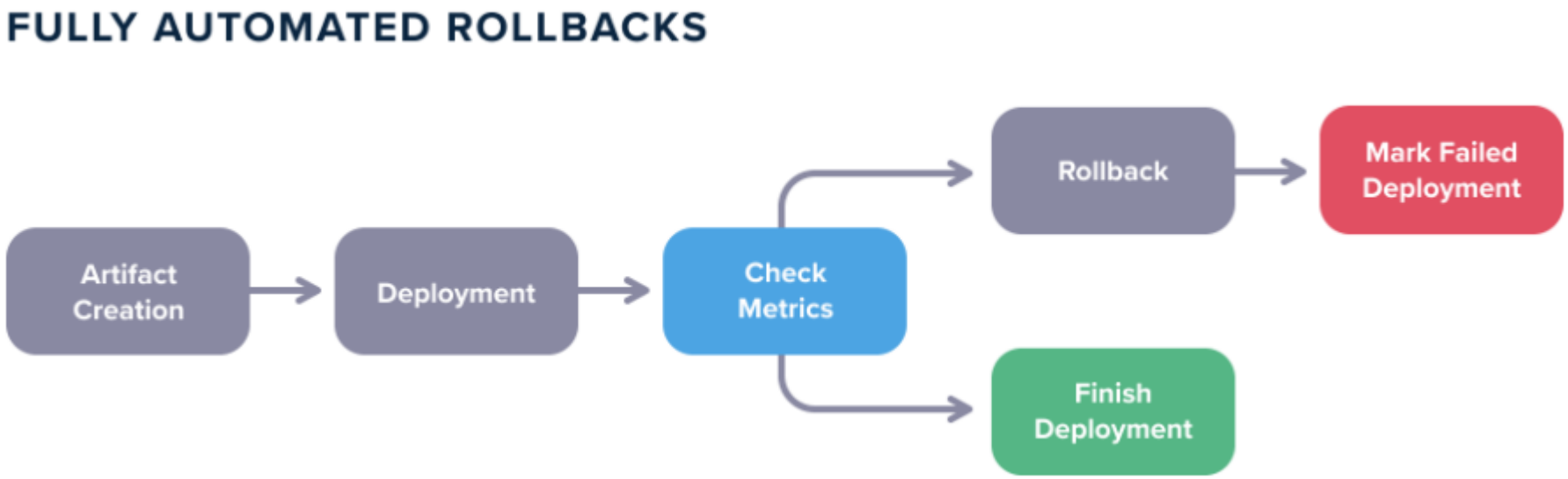
Two more pods are launched for the canary (for a total of 4), and finally we can shift 100% of live traffic to it

4



Automated Rollbacks with Metrics

- While you can use canaries with simple pauses between the different stages, Argo Rollouts offers the powerful capability to look at application metrics and decide automatically if the deployment should continue or not.
- The idea behind this approach is to completely automate canary deployments. Instead of having a human running manual smoke tests, or looking at graphs, you can set different thresholds that define if a deployment is “successful” or not.



Argo CD

Introduction to Argo CD

Argo CD is a GitOps tool implemented as a Kubernetes controller that continuously tracks applications and compares their current state to the target state defined in a Git repository.

When it detects that a deployed application’s live state deviates from its desired state, it marks the application as out of sync

Argo CD executes the later GitOps stages and ensures the correct deployment of new configurations to Kubernetes clusters. It is a Kubernetes-native tool for implementing continuous deployment. It differs from external CD tools, which only support push-based deployment because it can pull the updated code from a Git repository and deploy it directly to a Kubernetes resource

Argo CD is one member of the Argo family of projects. These are:

- Argo CD (GitOps controller)
- Argo Rollouts (Progressive Delivery controller)
- Argo Workflows (Workflow engine for Kubernetes)
- Argo Events (Event handling for Kubernetes)

GitOps workflow with Argo CD

A developer creates a new feature in the source code Git repository.

The CI system picks the source code change, packages the code, and runs unit tests or security scans as needed..

The developer makes a pull request and automatically or manually changes the Kubernetes manifest, stored in a separate Git repository that holds only Kubernetes manifests.

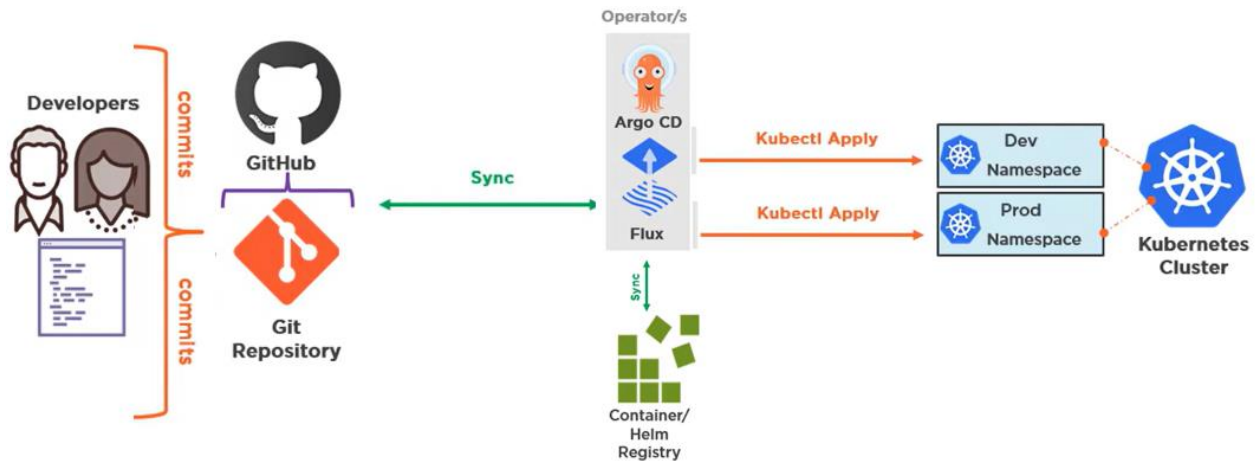
The GitOps team reviews the pull request and merges approved changes to the main branch, triggering a webhook that notifies Argo CD of the change.

Argo CD copies the repository and compares the desired application's state with the Kubernetes cluster's current state, applying the necessary changes to the cluster configuration.

Kubernetes controllers reconcile the required changes to the cluster resources until it reaches the target configuration.

Argo CD tracks the change's progress and reports the application as "in sync" when the cluster matches the desired state.

ArgoCD can also work in the opposite direction to monitor changes to the Kubernetes cluster, rolling them back if they do not match the current Git configuration.



Argo CD Installation

- For quick demos and experimentation, you can deploy ArgoCD by directly using the manifests
  - *kubectl create namespace argocd*
  - *kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml*
- For a production setup we suggest you use Autopilot, a companion project that not only installs Argo CD, but commits all configuration to git so Argo CD can manage itself using GitOps

Creating an application

- An application can be created in Argo CD from the UI, CLI, or by writing a Kubernetes manifest that can then be passed to kubectl to create resources.
- **Creating an Argo CD application in the UI**
  - First, navigate to the +NEW APP on the left-hand side of the UI. Next, add the following to create the application:
  - **General Section:**
    - Application Name: TBD
    - Project: default
    - Sync Policy: Automatic
  - **Source Section:**
    - Repository URL/Git: this is the GitHub repository URL
    - Branches: main
    - Path: TBD
  - **Destination Section:**
    - Cluster URL: select the cluster URL you are using
    - Namespace: default

Then, click CREATE and you have now created your Argo CD application.

Creating an application

- **Creating an Argo CD application with Argo CD CLI**

  *argocd app create {APP NAME} *
  *--project {PROJECT} *
  *--repo {GIT REPO} *
  *--path {APP FOLDER} *
  *--dest-namespace {NAMESPACE} *
  --dest-server {SERVER URL}

  - **{APP NAME}** is the name you want to give the application
  - **{PROJECT}** is the name of the project created or "default"
  - **{GIT REPO}** is the url of the git repository where the gitops config is located
  - **{APP FOLDER}** is the path to the configuration for the application in the gitops repo
  - **{DEST NAMESPACE}** is the target namespace in the cluster where the application will be deployed
  - **{SERVER URL}** is the url of the cluster where the application will be deployed. Use https://kubernetes.default.svc to reference the same cluster where Argo CD has been deployed

- Once this completes, you can see the status and configuration of the application
  - *argocd app list*
- For a more detailed view of the application configuration, run
  - *argocd app get {APP NAME}*

Sync Strategies

- There are 3 parameters that you can change when defining the sync strategy:
  - **Manual or automatic sync.**
    - Manual or automatic sync defines what Argo CD does when it finds a new version of your application in Git. If set to automatic, Argo CD will apply the changes then update/create new resources in the cluster. If set to manual, Argo CD will detect the change but will not change anything in the cluster
  - Auto-pruning of resources - this is only applicable for automatic sync.
    - Auto-pruning defines what Argo CD does when you remove/delete files from Git. If it is enabled, Argo CD will also remove the respective resources in the cluster as well. If disabled, Argo CD will never delete anything from the cluster.
  - Self-Heal of cluster - this is only applicable for automatic sync
    - Self-heal defines what Argo CD does when you make changes directly to the cluster (via kubectl or any other way). Note that doing manual changes in the cluster is not recommended if you want to follow GitOps principles (as all changes should pass from Git). If enabled, then Argo CD will discard the extra changes and bring the cluster back to the state described in Git.

Demo

Thank You

Nash Tech.