Understanding GitOps

GitOps

What is the difference between GitOps and CiCd / DevOps / ..... ??

- ***It is Ci/Cd Process used my devs BUT for App Config and Infra Config***

- Declarative description of the system is stored in Git (configs, monitoring, etc...)
  - ***Config in GIT, NOT THE CLUSTER is the Source of Truth !!!***
- Changes to the state are made via pull requests
  - ***BASICALLY, do what developers do for their source code.***
  - Application Config
    - **Kubernetes Admin calls this the App, but not exactly the app, it is the YAML Description of the app**
      - **myService.js, MyClass.java VS. Service.yaml, Deployment.yaml**
  - Cluster Config
    - Namespaces, secrets,
  - Shared tools
    - provisioned tools (ArgoCD, Jenkins), specific operators (Storage, Cloud Pak).
- Git push reconciled with the state of the running system with the state in the Git repository

GitOps Principals

The definition of our systems is described as code

- The configuration for our systems can be treated as code, so we can store it and have it automatically versioned in Git, our single source of truth.
- That way we can rollout and rollback changes in our systems in an easy way.

The desired system state and configuration is defined and versioned in Git

- Having the desired configuration of our systems stored and versioned in Git give us the ability to rollout / rollback changes easily to our systems and applications.
- On top of that we can leverage Git's security mechanisms in order to ensure the ownership and province of the code.

Changes to the configuration can be automatically applied using PR mechanisms

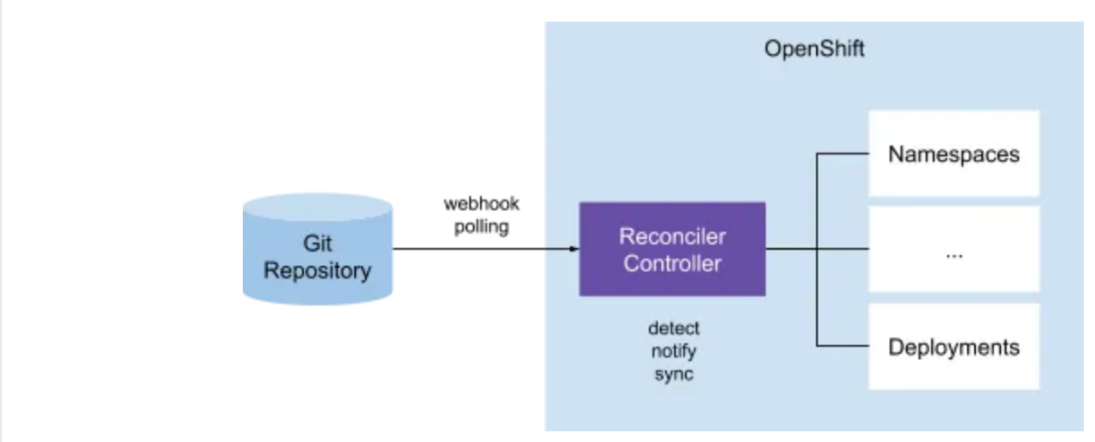
- Using Git Pull Requests we can manage in an easy way how changes are applied to the stored configuration, you can request reviews from different team members, run CI tests, etc.
- On top of that you don't need to share your cluster credentials with anyone, the person committing the change only needs access to the Git repository where the configuration is stored.

There is a controller that ensures no configuration drifts are present

- As the desired system state is present in Git, we only need a software that makes sure the current system state matches the desired system state. In case the states differ this software should be able to self-heal or notify the drift based on its configuration.

On-Cluster Resource Reconciler

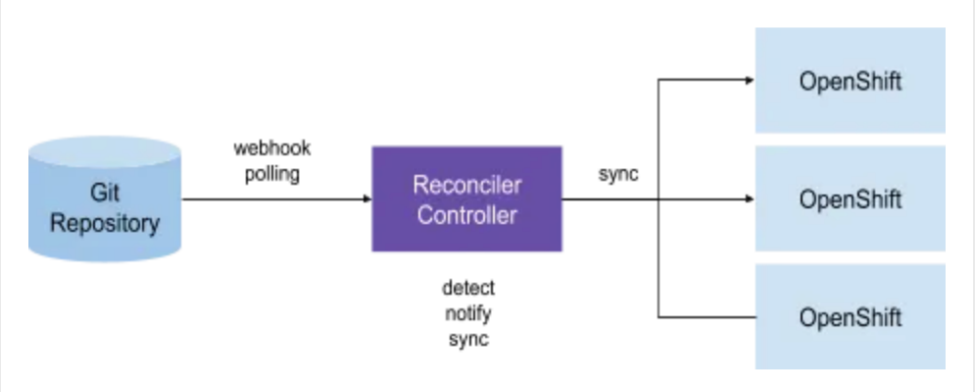
- Various ways to reconcile what's in Git and the cluster.
  - Ci/Cd tools: Jenkins, Tekton, MCM Cloud Pack, etc....
  - Argo CD
    - Specializes in GitOps on Kub.
  - Can be used with a Single Cluster.



- A controller on the cluster is responsible for comparing the Kubernetes resources (YAML files) in the Git repository that acts as the single source of truth, with the resources on the cluster.
- When a discrepancy is detected, the controller would send out notifications and possibly take action to reconcile the resources on Kubernetes with the ones in the Git repository

External Resource Reconciler (Push)

- A variation of the previous pattern is that one or more controllers are responsible for keeping resources in sync between pairs of Git repositories and Kubernetes clusters.
- The difference with the previous pattern is that the controllers are not necessarily running any of the managed clusters.
- The Git-k8s cluster pairs are often defined as a CRD which configures the controllers on how the sync should take place.
- The controllers in this pattern would compare the Git repository defined in the CRD with the resources on the Kubernetes cluster that is also defined in the CRD and takes action based on the result of the comparison.
- ArgoCD is one of the solutions that uses this pattern for GitOps implementation.



Use Git Ops for ?

Cluster Use Cases

- Ensure clusters have similar state (configs, monitoring, storage, etc)
- Recreate (or recover) clusters from a known state
- Create new clusters with a known state
- Rollout a change to multiple OpenShift clusters
- Rollback a change to multiple OpenShift clusters
- Associate templated configuration with different environments

Application Configuration Use Cases

- Promote applications (binary, config, etc) across clusters (e.g. dev, stage, etc)
- Rollout application changes (binary, config, etc) to multiple OpenShift clusters
- Rollback application changes to previous known stages

Some Git Op Use Stories

- **Apply configs from Git**
  As a cluster admin, I want to store OpenShift cluster configs in a Git repository and have the cluster to apply them automatically, so that I can install a new cluster and bring it to an identical known state from the Git repository.
- **Sync with Secret Manager**
  As a cluster admin, I want to keep OpenShift secrets in sync with a secret manager like Vault, so that I can manage secrets in a secret management platform.
- **Detect config drift**
  As a cluster admin, I want OpenShift GitOps to detect and display a warning when cluster configs are not in sync with the designated Git repo, so that I can take action when there is a config drift.
- **Notify config drift**
  As a cluster admin, I want to be notified when OpenShift GitOps detects a config drift, so that I can take action when there is a configuration drift.
- **Sync on config drift**
  As a cluster admin, I want to perform a sync on an OpenShift cluster to sync with the configs stored in a Git repository when a config drift is detected, so that the OpenShift cluster would come back to a known state.
- **Auto-sync on config drift**
  As a cluster admin, I want to configure an OpenShift cluster to automatically sync with the configs stored in a Git repository when a config drift is detected, so that the OpenShift cluster would always be in sync with the config in the designated Git repository.

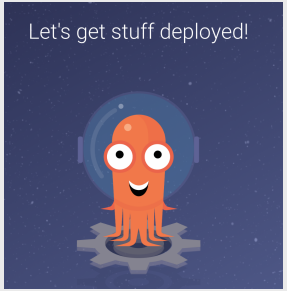
Some Git Op Use Stories

- **Multiple clusters in one registry**
  As a cluster admin, I want to define multiple OpenShift cluster config in a single Git repository and apply to clusters selectively, so that I can manage all cluster configs form a single source of truth.
- **Cluster config hierarchy (Inheritance)**
  As a cluster admin, I want to define a hierarchy of cluster configs (stage, prod, app portfolio, etc with inheritance) in a Git repository, so that I can define configs that apply to a single or multiple clusters.
- **Templating and Overriding configs**
  As a cluster admin, I want to override a subset of inherited configs and their values, so that I can adjust the config for the specific clusters they are being applied to.
- **Granularity include and exclude configs**
  As a cluster admin, I want to define when a certain config should apply or not apply to clusters with certain characteristics, so that I can have granular control over including or excluding cluster configs.
- **Application configs**
  As a cluster admin, I want to define when a certain config should apply or not apply to clusters with certain characteristics, so that I can have granular control over including or excluding cluster configs.
- **Templating support**
  As a developer, I want to have a choice on how to define my application resources (Helm Chart, pure k8s yaml etc), so that I can pick the right format based on my application needs.

Example Git Op Tool Examples

ArgoCD

ArgoCD follows the External Resource Reconcile pattern, it has a central UI that can orchestrate one to many clusters and multiple Git repositories. One weakness is that if ArgoCD goes down, application management cannot be done. (You can make Toold HA)



Flux

Flux follows the On-Cluster Resource Reconcile pattern, as there is not a central management of repository definitions, if one cluster goes down. the ability exists still to manage applications. One weakness is that no central UI is provided by the tool.

GitOps Patterns can be built with traditional Ci/Cd tools

Full solution still requires integrating with Ci tools, testing tools, templating tools, etc....

Cloud Pak for MultiCloud Management can be used to drive overall story