Red Hat
Technical Account
Management

The TAM
your tech hour

GitOps for OpenShift Administrators

Simon Krenger

Principal Technical Account Manager , Red Hat

Simon Krenger

Principal Technical Account Manager
OpenShift Container Platform

Located in Bern, Switzerland

What we’ll discuss today

Introduction to common GitOps setups

Declarative GitOps

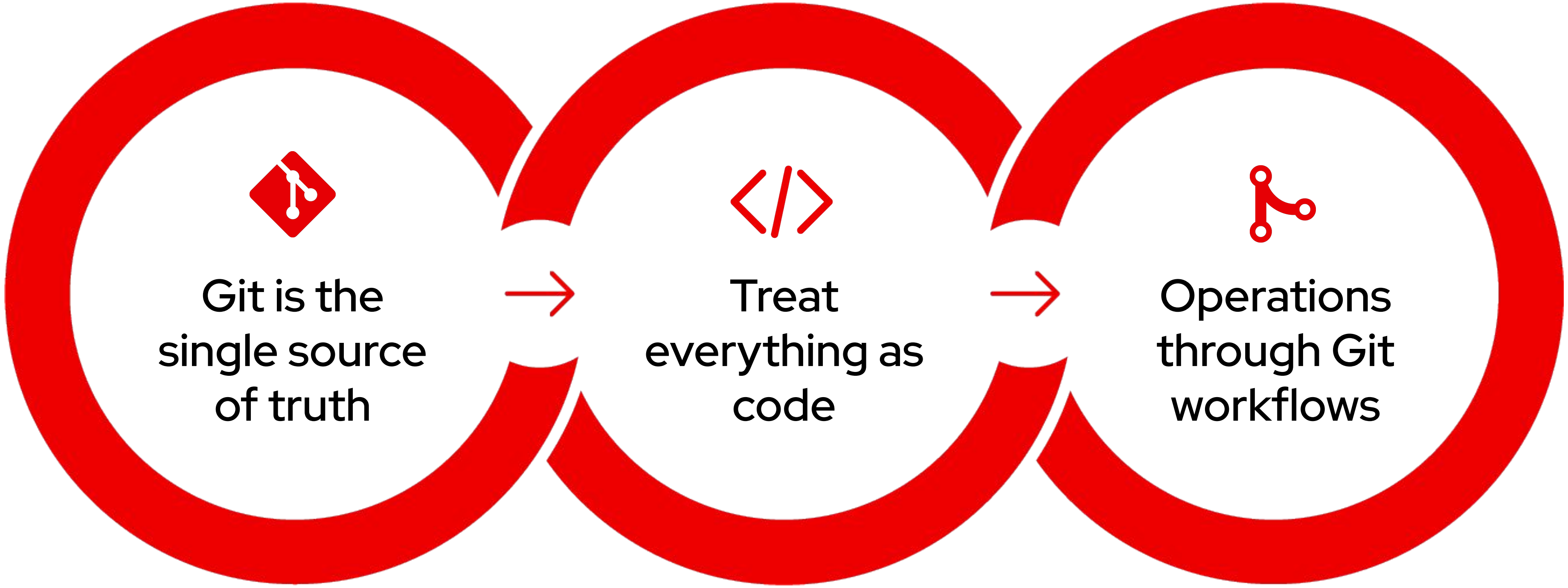
Patterns for managing OpenShift

Best practices for Argo CD

Common GitOps setups

Why do we want to use GitOps?

An developer-centric approach to Continuous Delivery and infrastructure operation



Git is the single source of truth → Treat everything as code → Operations through Git workflows

GitOps Principles

The system is described declaratively

The desired state is versioned in Git

Approved changes can be applied automatically

A controller exists to detect and act on drift

Common questions from customers

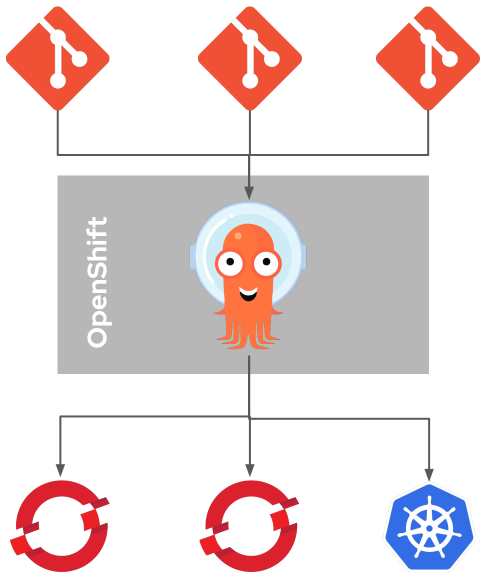
... and what we'll discuss today

How can I use GitOps as an OpenShift administrator?

What are others doing?

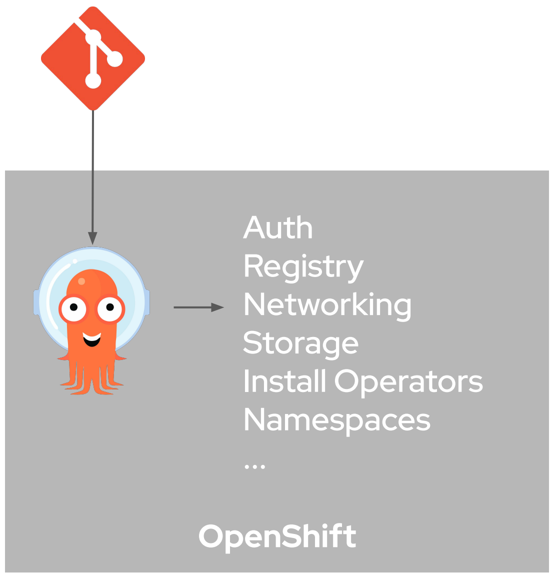
What are common setups?

GitOps Deployment Strategies



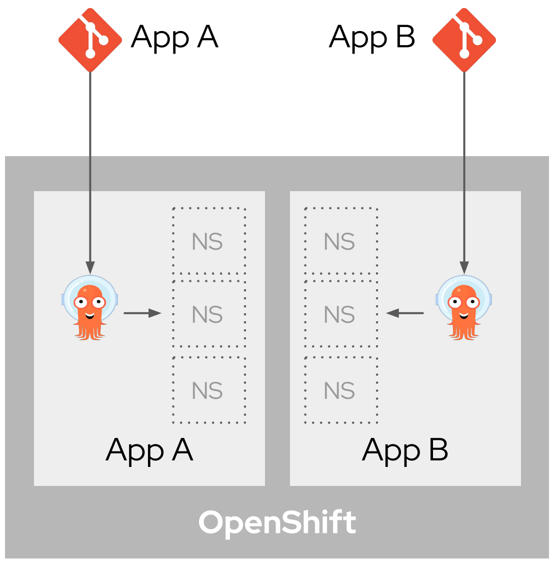
Central Hub (Push)

A central Argo CD pushes Git repository content to remote OpenShift and Kubernetes clusters



Cluster Scoped (Pull)

A cluster-scope Argo CD pulls cluster service configurations into into the OpenShift cluster



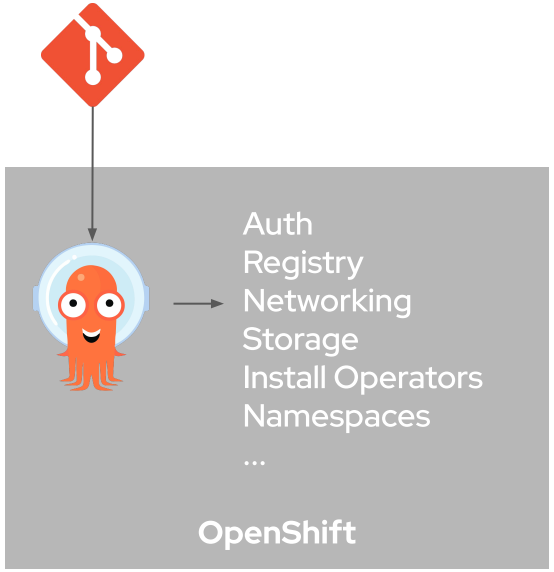
Application Scoped (Pull)

An application scoped Argo CD pulls application deployment and configurations into app namespaces

How do Red Hat customers use ArgoCD?

Common setups for administrators

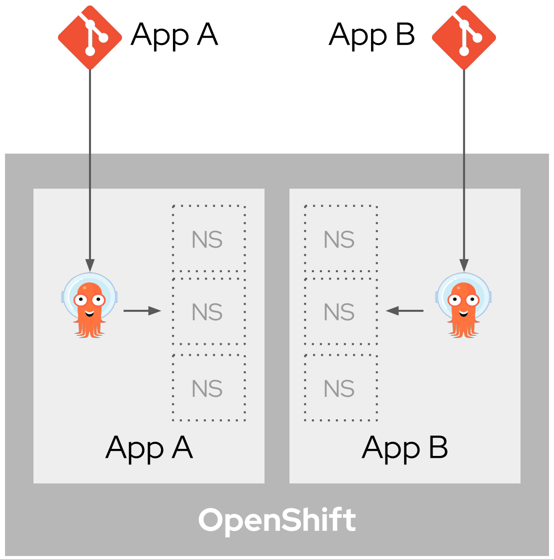
- Built-in Global Argo CD for cluster administration
- Has all the necessary permissions out-of-the-box
- Different `Applications` configure different parts of the cluster
- Administrators deploy their infrastructure applications via Argo CD



How do Red Hat customers use ArgoCD?

Common setups for development teams

- For development teams, we often see different setups:
  - Central application-specific Argo CD instance managing workload namespaces
  - One Argo CD per namespace / team



Red Hat Advanced Cluster Management

RHACM and OpenShift GitOps: Better together

Red Hat
Advanced Cluster
Management
for Kubernetes

Designed for multi-cluster use cases

Customers often use RHACM with GitOps to manage multiple clusters instead of “just” OpenShift GitOps.

Leverages ACM Placement API

Provides predicate selection, taints/tolerations, scoring/prioritizer, spread, and affinity

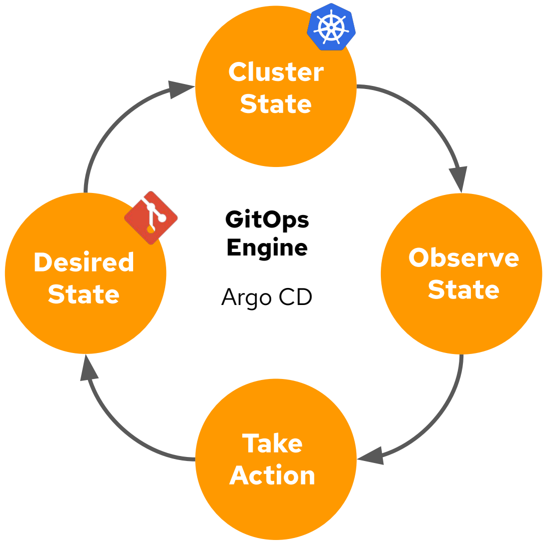
Additional features

Topology view for ApplicationSets, Support for additional usage patterns (push / pull), Cluster lifecycle integration

What else do I need to know?

GitOps principles are not only technical

- To get all the advantages of GitOps, your processes could include:
  - Pull Requests / Merge Requests with approvals
  - All configuration on the cluster via Git only (no admin accounts with write access)
  - Auto prune and self-heal configuration



Declarative GitOps

Declarative GitOps

Managing GitOps using GitOps

OpenShift GitOps Operator

Built-in CustomResourceDefinitions for declarative management

Out-of-the-box CRDs:

- `ArgoCD`
- `Application`
- `ApplicationSet`
- `AppProject`
- `Rollout`
- …

Managing ArgoCD instances

“ArgoCD” CustomResourceDefinition

- Represents an Argo CD / GitOps instance
  - Separate Routes / RBAC / configuration for each instance
  - Useful to set up separate instances for separate teams
- When installing OpenShift GitOps, cluster-wide ArgoCD instance exists out-of-the-box

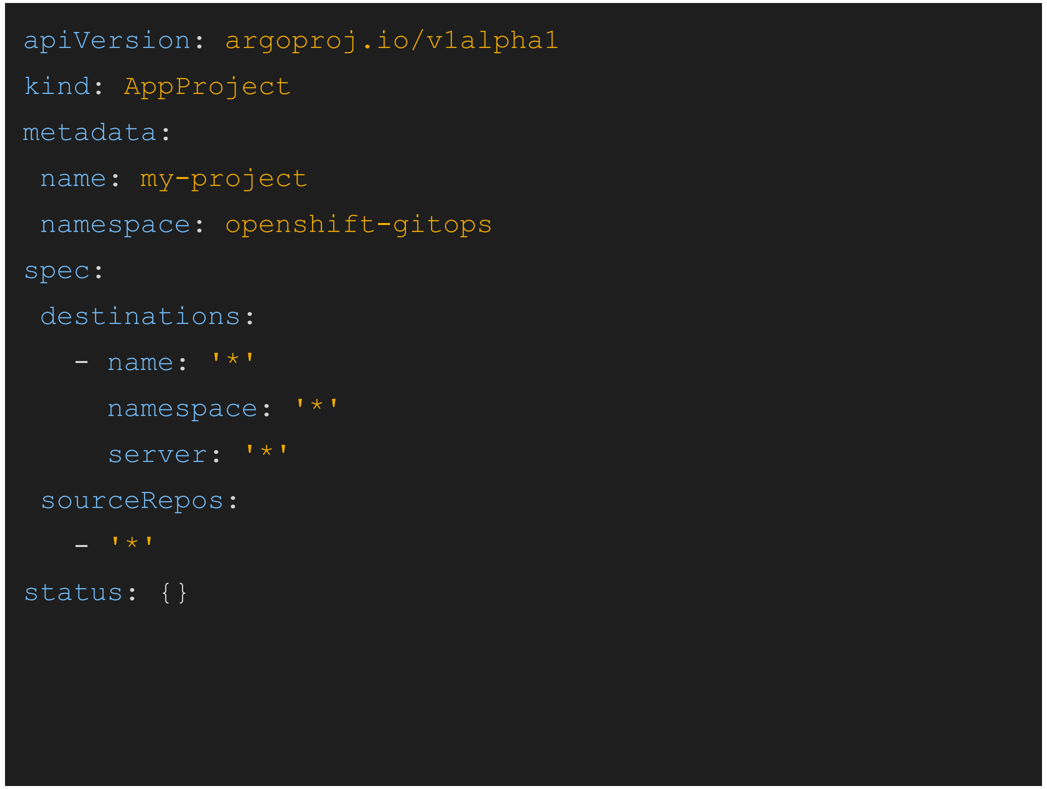
```
apiVersion: argoproj.io/v1beta1
kind: ArgoCD
metadata:
 name: openshift-gitops
 namespace: openshift-gitops
spec:
 server:
   autoscale:
     enabled: false
   grpc:
     ingress:
       enabled: false
   ingress:
     enabled: false
[..]
```

Source:
https://docs.openshift.com/gitops/1.12/argocd_instance/argo-cd-cr-component-properties.html

AppProjects

“AppProject” CustomResourceDefinition

- Represents a project within Argo CD
  - Typically used to restrict Source, Destination, Namespaces
- Only necessary when multi-tenancy is done within Argo CD



```
apiVersion: argoproj.io/v1alpha1
kind: AppProject
metadata:
 name: my-project
 namespace: openshift-gitops
spec:
 destinations:
   - name: '*'
     namespace: '*'
     server: '*'
 sourceRepos:
   - '*'
status: {}
```

Source:
https://argo-cd.readthedocs.io/en/stable/operator-manual/declarative-setup/#projects

Repositories

Secret with “secret-type: repository” label

- Represents a source repository
  - Typically used when credentials need to be supplied
- No CRD, is a standard Secret with the `argocd.argoproj.io/secret-type: repository` label
- Alternative is to use repo-creds credentials

```
kind: Secret
apiVersion: v1
metadata:
 name: repo-2105479437
 namespace: openshift-gitops
 labels:
   argocd.argoproj.io/secret-type: repository
stringData:
 project: default
 type: git
 sshPrivateKey: <REDACTED>
 url: https://github.com/simonkrenger/gitops-basics-demos.git
type: Opaque
```

Source:
https://argo-cd.readthedocs.io/en/stable/operator-manual/declarative-setup/#repositories

Clusters

Secret with “secret-type: cluster” label

- Represents a destination cluster
- No CRD, is a standard Secret with the `argocd.argoproj.io/secret-type: cluster` label
- Used to store credentials for accessing clusters

```yaml
kind: Secret
apiVersion: v1
metadata:
 name: mycluster-example
 namespace: openshift-gitops
 labels:
   argocd.argoproj.io/secret-type: cluster
 annotations:
stringData:
 config: |
  {"bearerToken": "<authentication token>", [..]}
 name: mycluster.example.com
 server: https://mycluster.example.com
type: Opaque
```

Source:
https://argo-cd.readthedocs.io/en/stable/operator-manual/declarative-setup/#clusters

Demo:
Declarative GitOps

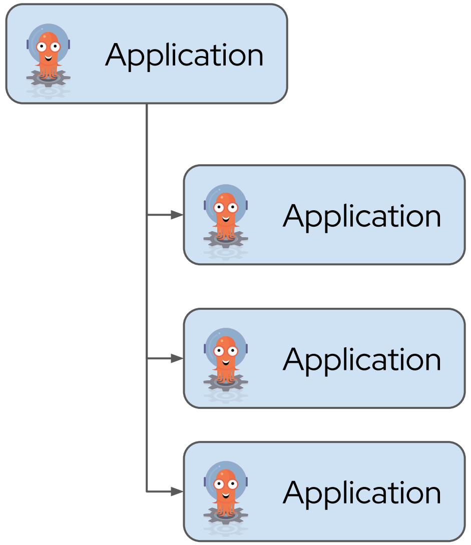
The TAM your tech hour

Patterns for managing OpenShift

App-of-apps

Manage Applications via an Application

- App of Apps is a common pattern where one Argo CD `Application` points to a repo that only contains other Argo CD `Applications`
- Very useful to be able to provision and manage a group of related applications together
- Pattern evolved over time, still seen in many places



ApplicationSets

Standardised way to deploy many Applications

- The ApplicationSet controller automatically generates Argo CD `Applications` based on the contents of an `ApplicationSet` Custom Resource (CR).
- Examples of available generators:
  - List generator
  - Cluster generator
  - Git generator
  - Matrix generator

```
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
 name: guestbook
 namespace: openshift-gitops
Spec:
[..]
 generators:
 - list:
     elements:
     - cluster: dev
       url: https://1.2.3.4

[..]
  template:
    metadata:
      name: '{{.cluster}}-guestbook'
[..]
```

Source: https://argo-cd.readthedocs.io/en/stable/user-guide/application-set/

App-of-apps / ApplicationSets

What to use when

App-of-apps:

- Easier to learn and to use when you already have existing `Applications`
- Available in all Argo CD versions
- Scaling (>100 Applications) may be an issue
  - Maintenance of `Application` YAMLs

ApplicationSets:

- Requires consistent application / repository layout
- Generators give flexibility:
  - Allows dynamic generation of ApplicationSets (git generator)
  - Can become very complex (matrix generators)

Secrets Management

Careful when storing credentials or secrets in Git

- Very common question from customers, Secrets Management is a big topic in GitOps workflows
- Red Hat does not provide any Secrets Management product at this time
- External Secrets Operator support is planned in OpenShift

Red Hat does not provide support or recommendations for specific third-party software. However some customers use:

- Bitnami Sealed Secrets
- Hashicorp Vault
- Mozilla SOPS

Larger GitOps setups

Considerations for large Argo CD instances

- Argo CD has some architectural limitations when it comes to large instances
- This mostly affects namespace-scoped Argo CD instances
- Keep this in mind when designing your Argo CD environment

- Solution 7006291 describes possible tuning options
- QPS, Processors, cluster-scope, resource exclusions

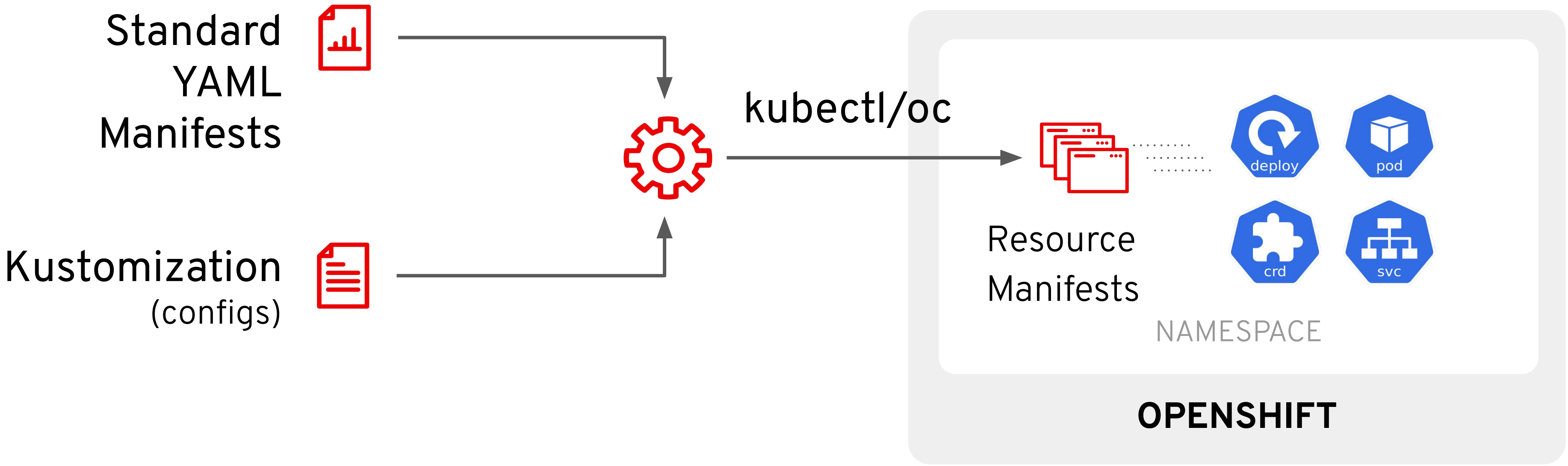
- When managing multiple clusters with Argo CD, dynamic sharding is available

Source:
https://access.redhat.com/solutions/7006291
https://developers.redhat.com/articles/2023/09/26/dynamically-scale-argo-cd-application-controller-openshift-gitops-110

Kustomize / Helm

Kustomize / Helm is built-into Argo CD

Allows for templating your YAML to your destination clusters:



Demo:
Argo CD patterns

The TAM your tech hour

Best Practices for Argo CD

Argo CD Best Practices

From the upstream “Best Practices” documentation

Review Argo CD best practice website

The Best Practices documentation page on the upstream website is helpful

Leaving Room For Imperativeness

You do not necessarily need to define all values in the YAML (for example do not set `replicas` when HPA is being used)

Separating Config Vs. Source Code Repositories

Separating application code and application configuration allows for separate access and auditing

Ensuring Manifests At Git Revisions Are Truly Immutable

Avoid using “latest” for container images and refer to code repositories with a tag or commit SHA

Source:
https://argo-cd.readthedocs.io/en/stable/user-guide/best_practices/

GitOps quality of life tips

GitOps quality of life tips from Red Hats GitOps team

Use annotation tracking

Kubernetes annotations do not have the same limitations as labels (used by default by Argo CD), annotation tracking can help when hitting these limitations

Use kubectl-neat to export clean resources

Use the `neat` plugin for `kubectl` and `oc` to remove certain fields from resource YAMLs

Override automatic sync for App of Apps

Use the `ignoreDifferences` feature in the parent application to simplify the process of changing an `Application` managed by an App-of-apps

Use Global projects

When using `AppProjects`, consider adding a Global project to manage cluster-wide configuration

Source:
https://www.redhat.com/en/blog/gitops-quality-life-tips
https://github.com/itaysk/kubectl-neat

More best practices learned over the years

From Red Hats customers

Regularly ensure “no double management” of resources

A rapidly increasing `resourceVersion` number on an object can indicate that both an Operator and also GitOps manage the same resource

Easily test custom health checks

Use the `argocd admin` command to manually run custom Argo CD health checks

Kustomize for managing ConfigMap contents

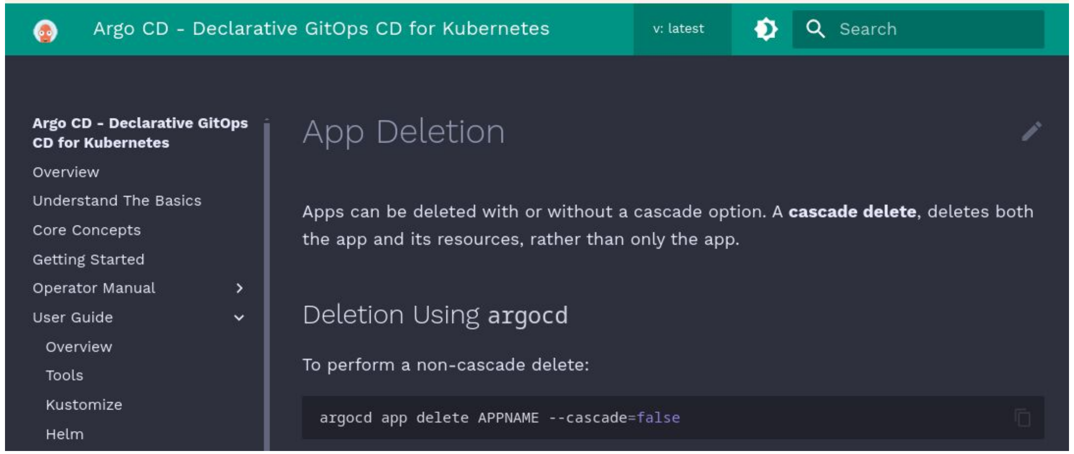
Managing YAML within YAML can be a pain, use kustomize and `configMapGenerator` to generate `ConfigMaps` / `Secrets`

Source:
https://www.redhat.com/en/blog/gitops-quality-life-tips

Deleting Applications: Danger Zone

A word of caution

- When deleting resources, take care to not accidentally delete all resources managed by Argo CD
- One customer deleted all his `MachineSets` when deleting an Argo CD `Application`
- Use Sync Options such as `Prune=false` or `Delete=false` to avoid deleting important objects



App Deletion

Apps can be deleted with or without a cascade option. A **cascade delete**, deletes both the app and its resources, rather than only the app.

Deletion Using `argocd`

To perform a non-cascade delete:

```
argocd app delete APPNAME --cascade=false
```

Demo:
Best practices

Summary

- **Common GitOps Setups**

  Instances for administrators, instances for developers

- **Declarative GitOps**

  Manage GitOps configuration using GitOps

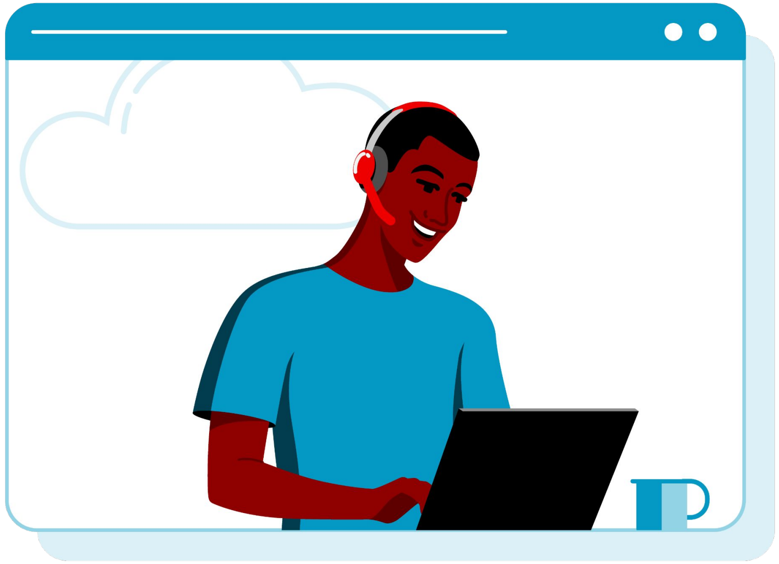
- **Patterns for managing OCP**

  App-of-apps, Secrets Management and Large Instances

- **Best Practices**

  Tips for common issues

Thanks !



- http://linkedin.com/company/Red-Hat
- http://youtube.com/user/RedHatVideos
- http://facebook.com/RedHatinc
- https://twitter.com/RedHat

Red Hat
Technical Account
Management