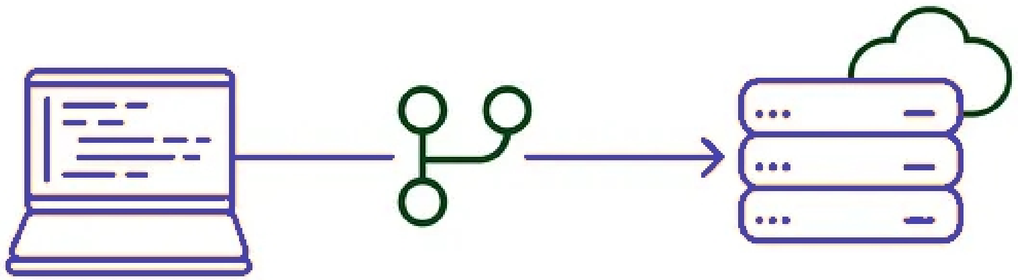
# Argo CD

Modern app of apps
with ApplicationSet

- Infrastructure as code (IaC)
- Introduction to GitOps
- Introduction to ArgoCD
- Application CRD
- Understanding ApplicationSet
- Generators in ApplicationSet
- ApplicationSet Templates
- Understanding ApplicationSet
- Generators in ApplicationSet
- ApplicationSet Templates
- GitOps Workflow with ApplicationSet
- Cluster bootstraping with App of apps
- App of apps with AppSet
- Best Practices for GitOps and ApplicationSet

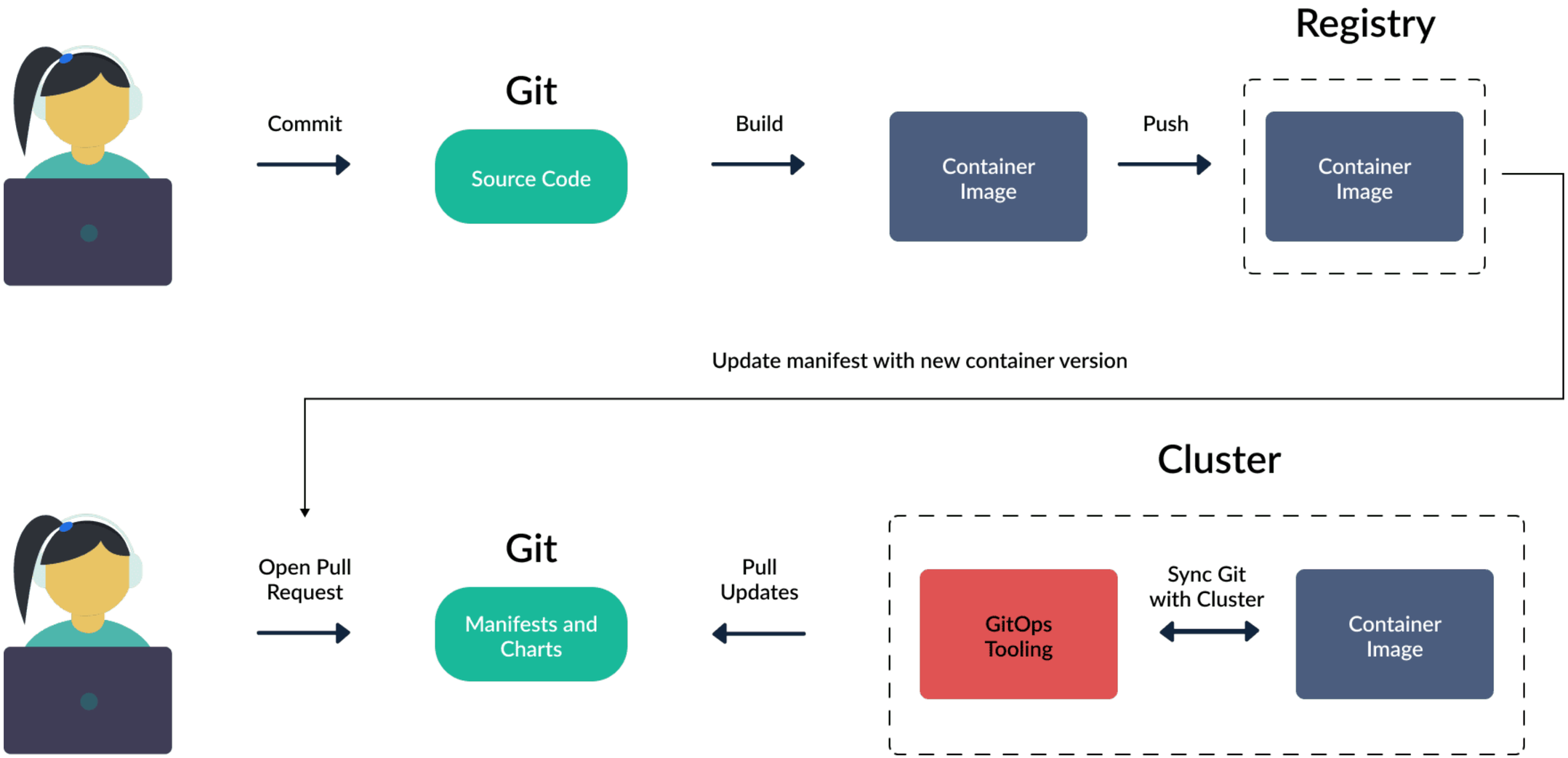
# Infrastructure as code (IaC)

Infrastructure as code (IaC) is the ability to provision and support your computing infrastructure using code instead of manual processes and settings. Any application environment requires many infrastructure components like operating systems, database connections, and storage. Developers have to regularly set up, update, and maintain the infrastructure to develop, test, and deploy applications.



Manual infrastructure management is time-consuming and prone to error—especially when you manage applications at scale. Infrastructure as code lets you define your infrastructure's desired state without including all the steps to get to that state.

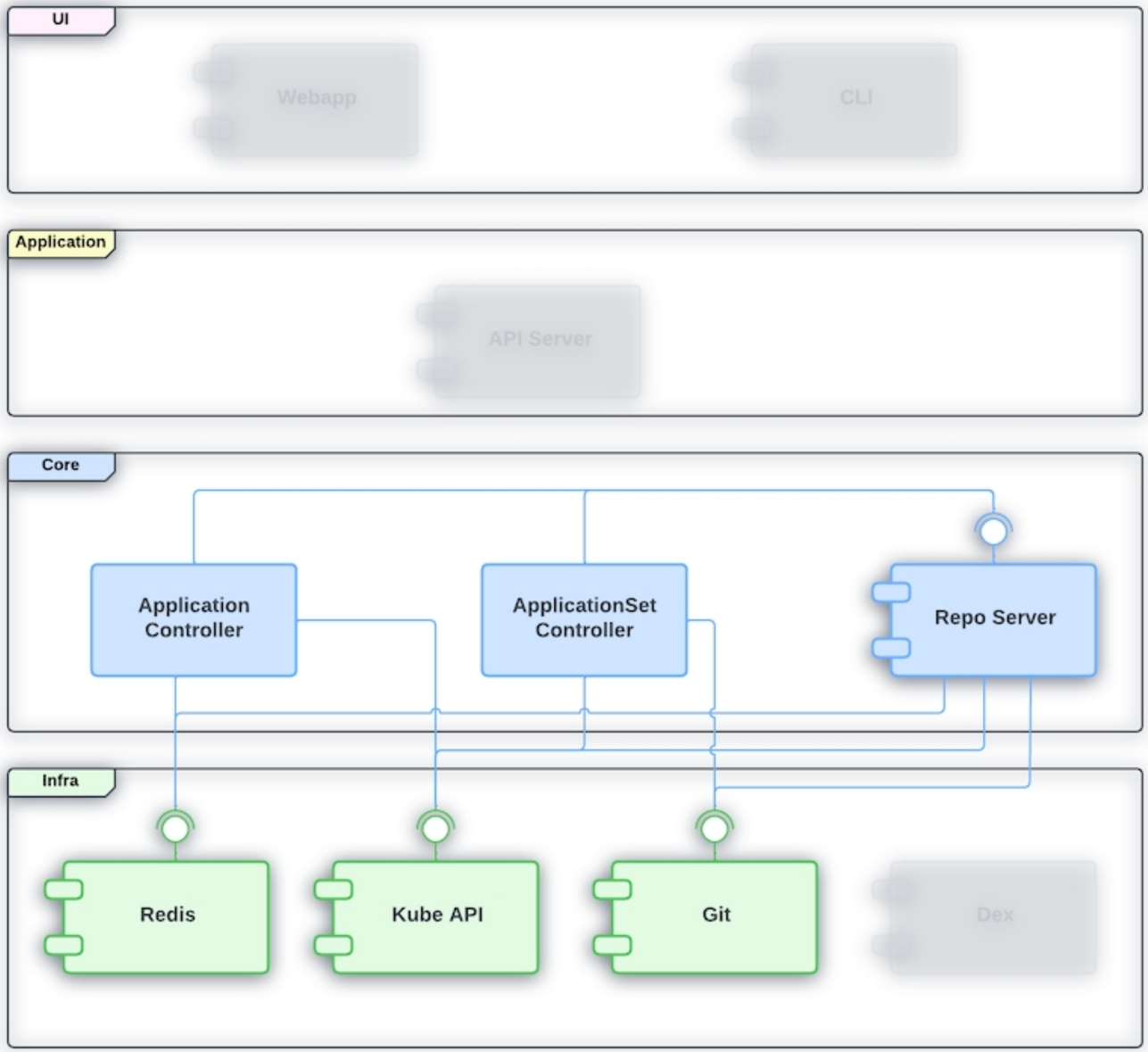
# Introduction to GitOps



# Introduction to ArgoCD

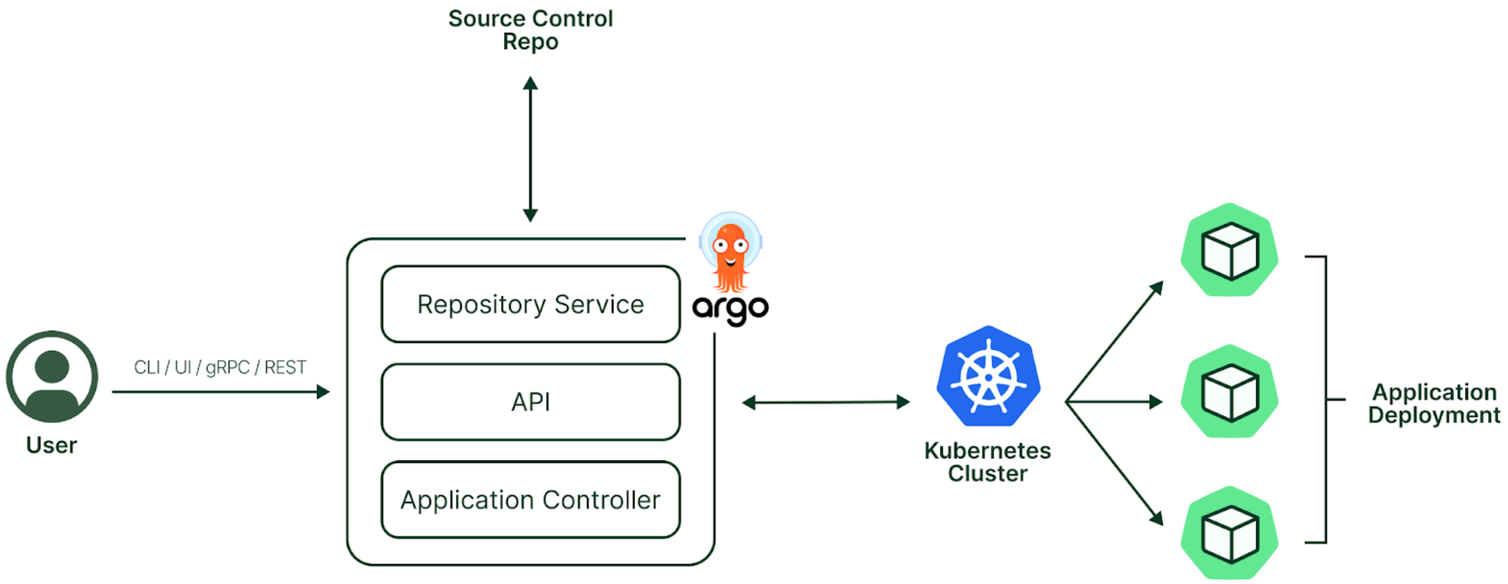
Argo CD is a declarative, GitOps continuous delivery tool for Kubernetes.

Application definitions, configurations, and environments should be declarative and version controlled. Application deployment and lifecycle management should be automated, auditable, and easy to understand.



# Application CRD

The application controller is a Kubernetes controller which continuously monitors running applications and compares the current, live state against the desired target state (as specified in the repo). It detects OutOfSync application state and optionally takes corrective action. It is responsible for invoking any user-defined hooks for lifecycle events (PreSync, Sync, PostSync)

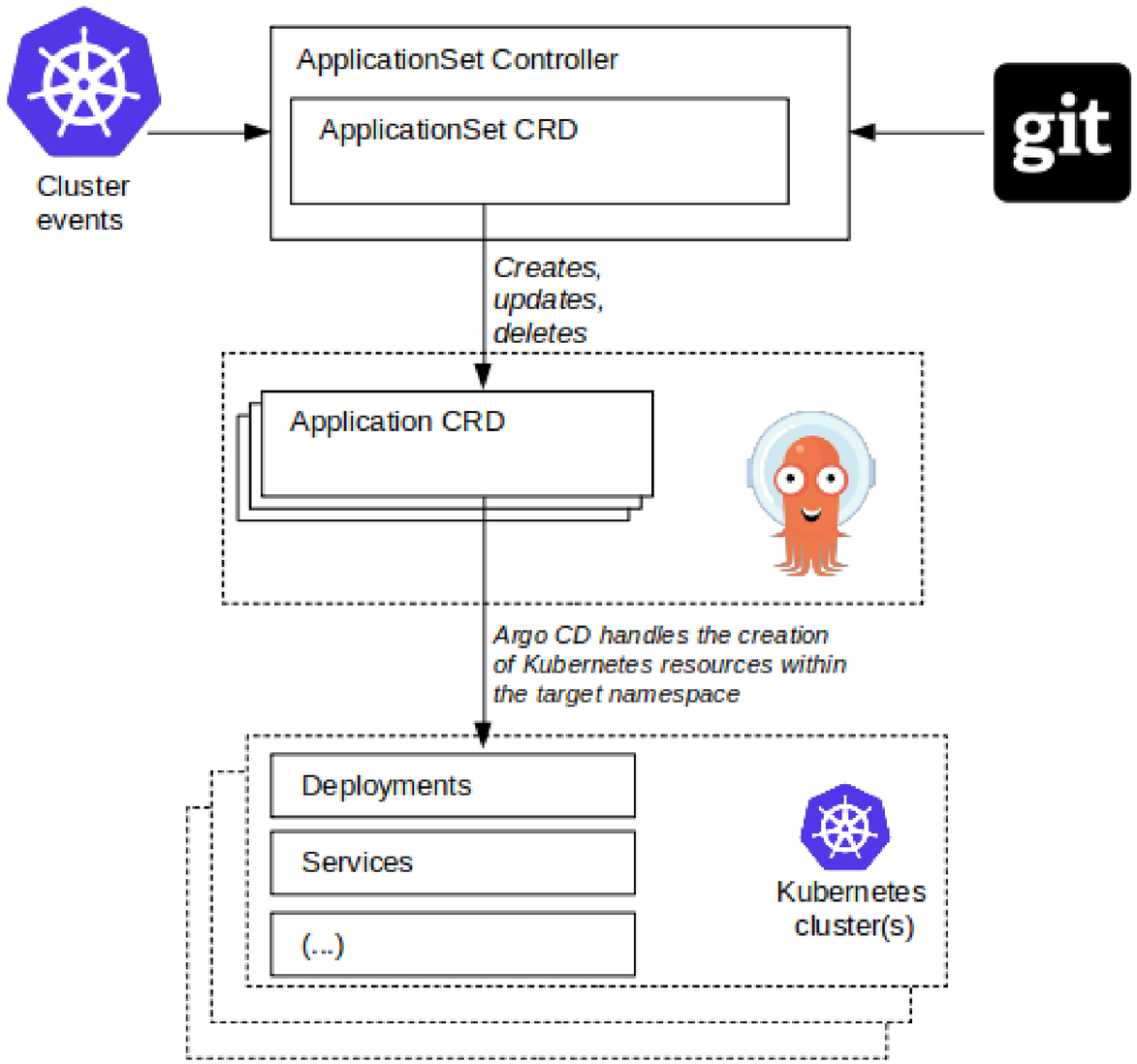


# Understanding ApplicationSet

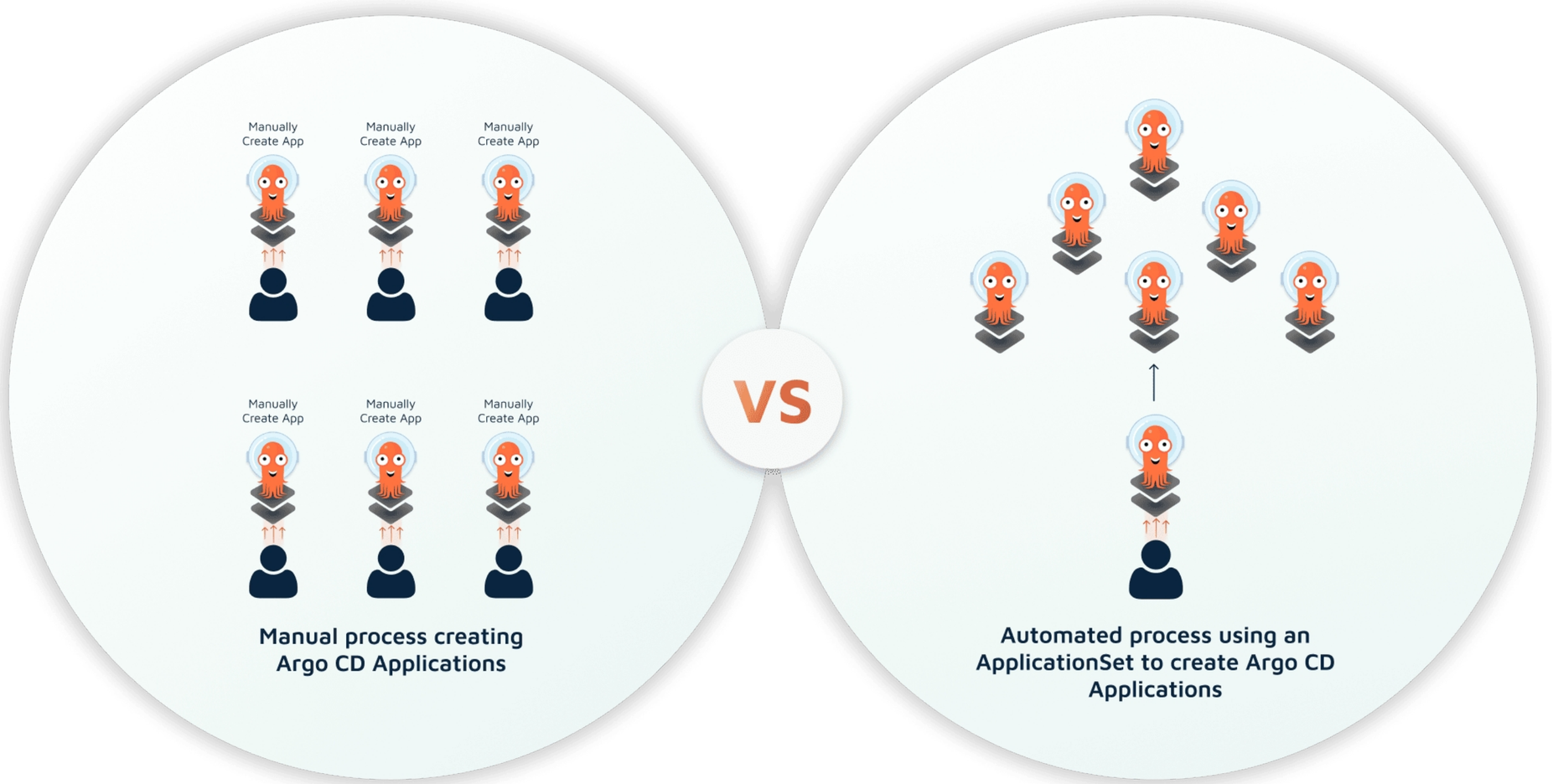
The ApplicationSet controller manages multiple Argo CD Applications as a single ApplicationSet unit, supporting deployments to large numbers of clusters, deployments of large monorepos, and enabling secure Application self-service.

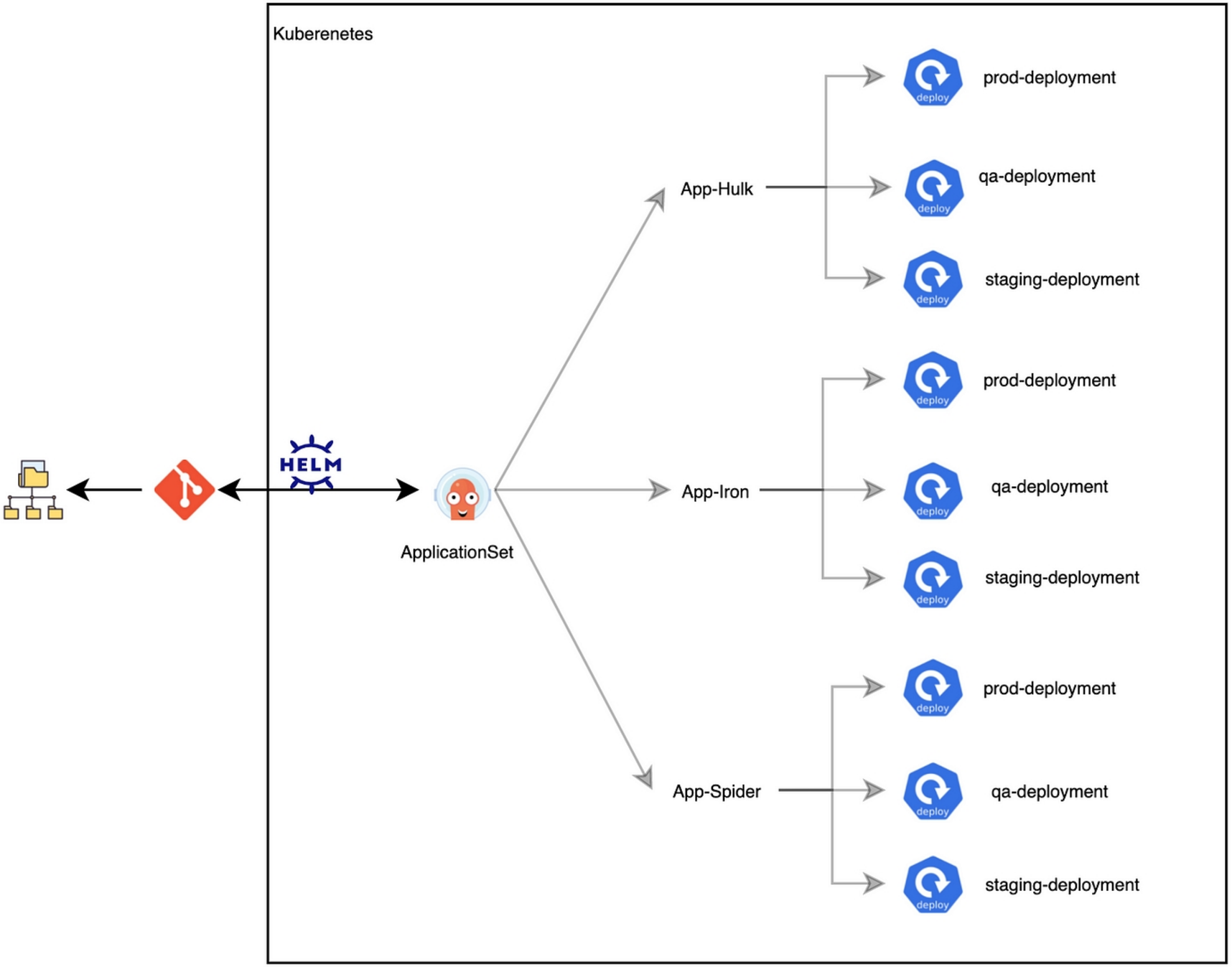
Key concepts:

- Generators
- Templating



# Understanding ApplicationSet




Kuberenetes
HELM
ApplicationSet
App-Hulk
App-Iron
App-Spider
prod-deployment
qa-deployment
staging-deployment
prod-deployment
qa-deployment
staging-deployment
prod-deployment
qa-deployment
staging-deployment

# ApplicationSet generators

Generators are responsible for generating parameters, which are then rendered into the “ template: ” fields of the ApplicationSet resource.

Generators are primarily based on the data source that they use to generate the template parameters. For example: the List generator provides a set of parameters from a literal list, the Cluster generator uses the Argo CD cluster list as a source, the Git generator uses files/directories from a Git repository, and so.

## ApplicationSet generators list

- List generator
- Cluster generator
- Git generator
- Matrix generator
- Merge generator
- SCM Provider generator
- Pull Request generator
- Cluster Decision Resource generator
- Plugin generator

# ApplicationSet Template

The template fields of the ApplicationSet spec are used to generate Argo CD Application resources.An Argo CD Application is created by combining the parameters from the generator with fields of the template (via {{values}}), and from that a concrete Application resource is produced and applied to the cluster.ApplicationSet is able to use Go Text Template. To activate this feature, add goTemplate: true to your ApplicationSet manifest.

- **Fasttemplate**

  Simple and fast template engine for Go. Fasttemplate performs only a single task - it substitutes template placeholders with user-defined values.

- **Go Text Template**

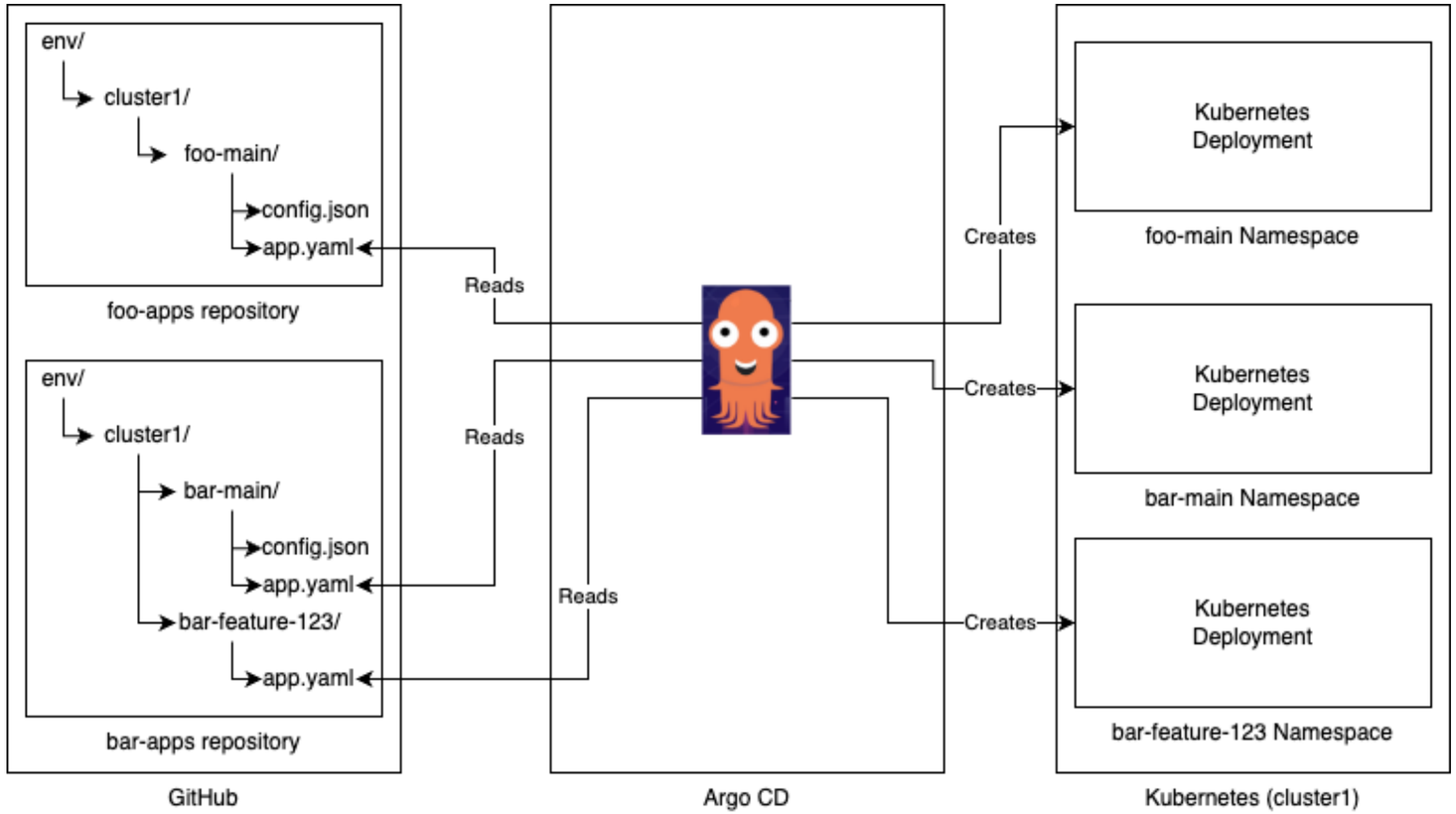
  implements data-driven templates for generating textual output. Templates are executed by applying them to a data structure. The text/template package is used for creating and rendering text-based templates. It’s commonly used for generating dynamic content, such as configuration files.

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: color-applicaitonset
  namespace: argocd
spec:
  generators:
  - list:
      elements:
      - namespace: dev
      - namespace: test
      - namespace: uat
  template:
    metadata:
      # will generate applications with different names for example (dev-color-app, test-color-app, etc..)
      name: '{{namespace}}-color-app'
    spec:
      # applications will be in default project for argocd
      project: default
      source:
        repoURL: https://github.com/AmrAlaaYassen/ArgoCD-ApplicationSet-Demo.git
        targetRevision: HEAD
        path: list-generator-example/deploymen
      destination:
        # default cluster as destination, you can define multiple clusters in ArgoCD UI
        server: https://kubernetes.default.svc
        # will deploy to different namespaces as different destinations
        namespace: '{{namespace}}'
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: argo-projects
spec:
  generators:
  - git:
      # git repository to get the variables from
      repoURL: https://github.com/AmrAlaaYassen/ArgoCD-ApplicationSet-Demo.git
      # branch used to get the variables from
      revision: HEAD
      directories:
      # path to the directory that includes directories to read as variables
      - path: git-dir-generator-example/argo-projects/*
  template:
    metadata:
      # basename is the name of the directory not the full path
      name: '{{path.basename}}'
    spec:
      project: default
      source:
        # source repo, in this example both repositories are the same
        repoURL: https://github.com/AmrAlaaYassen/ArgoCD-ApplicationSet-Demo.git
        targetRevision: HEAD
        # path to read manifests from, here it's the full path not only the name of the directory
        path: '{{path}}'
      destination:
        server: https://kubernetes.default.svc
        # different namespaces named after the directories names to be used as destinations
        namespace: '{{path.basename}}'
```

```yaml
apiVersion: argoproj.io/v1alpha1
kind: ApplicationSet
metadata:
  name: demo-helm-common-chart
  namespace: argocd
spec:
  generators:
    - git:
        repoURL: https://github.com/AmrAlaaYassen/ArgoCD-ApplicationSet-Demo.git
        revision: HEAD
        directories:
          # here it will read the paths under configs and the subpaths under the config
paths
          - path: demo/configs/*/*
  template:
    metadata:
      namespace: argocd
      # you can use the path strings with / delimiter as variables
      name: "{{path[2]}}-{{path.basename}}"
    spec:
      project: default
      source:
        helm:
          # specify different values file based on directories variables
          valueFiles:
            - "configs/{{path[2]}}/{{path.basename}}/values.yaml"
            - "configs/{{path[2]}}/values.yaml"
        path: demo
        repoURL: https://github.com/AmrAlaaYassen/ArgoCD-ApplicationSet-Demo.git
        targetRevision: HEAD
      syncPolicy:
        automated:
          prune: true
      destination:
        # specify different destinations
        namespace: "{{path.basename}}"
        server: https://kubernetes.default.svc
```

# GitOps Workflow with ApplicationSet

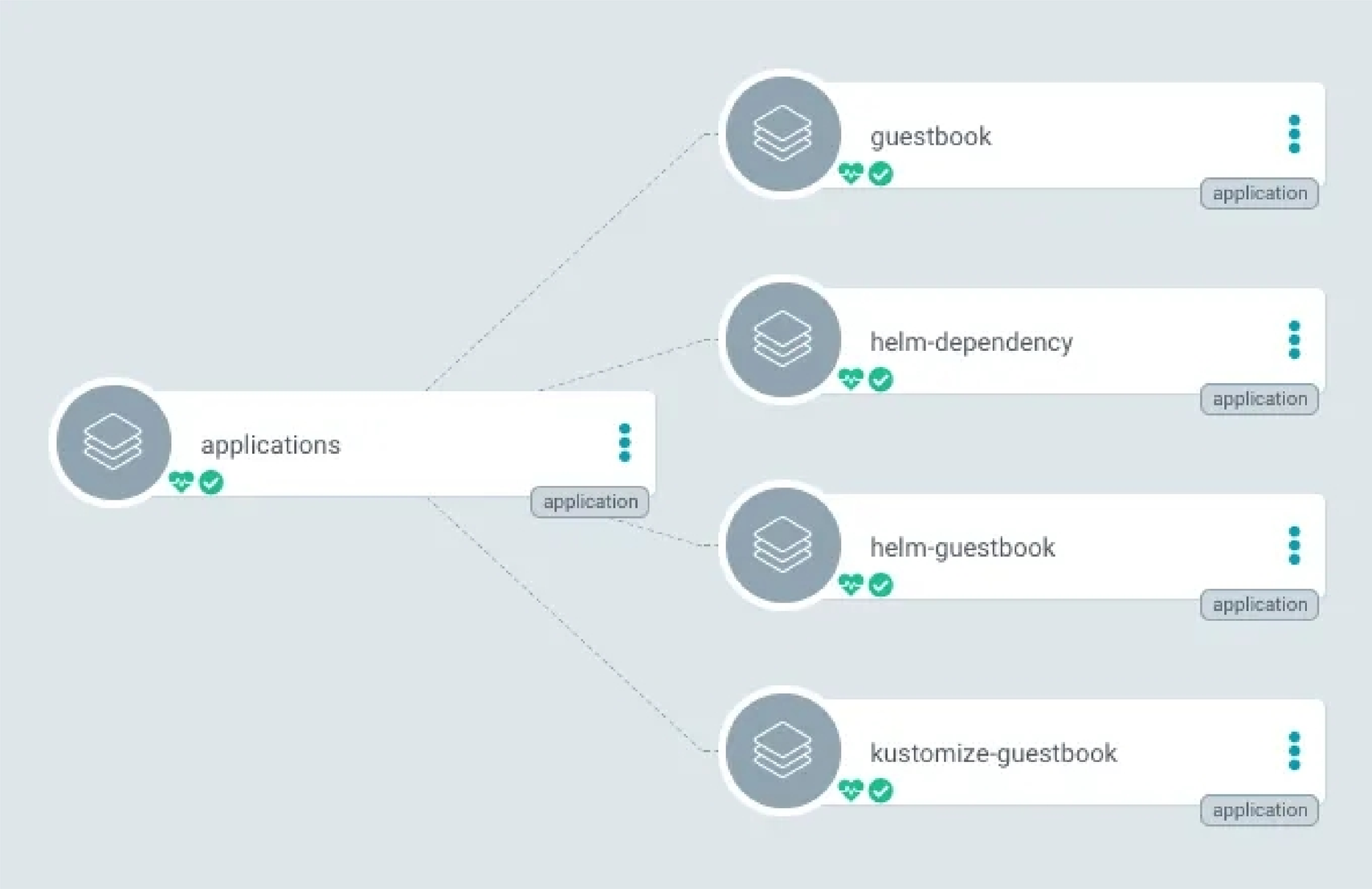


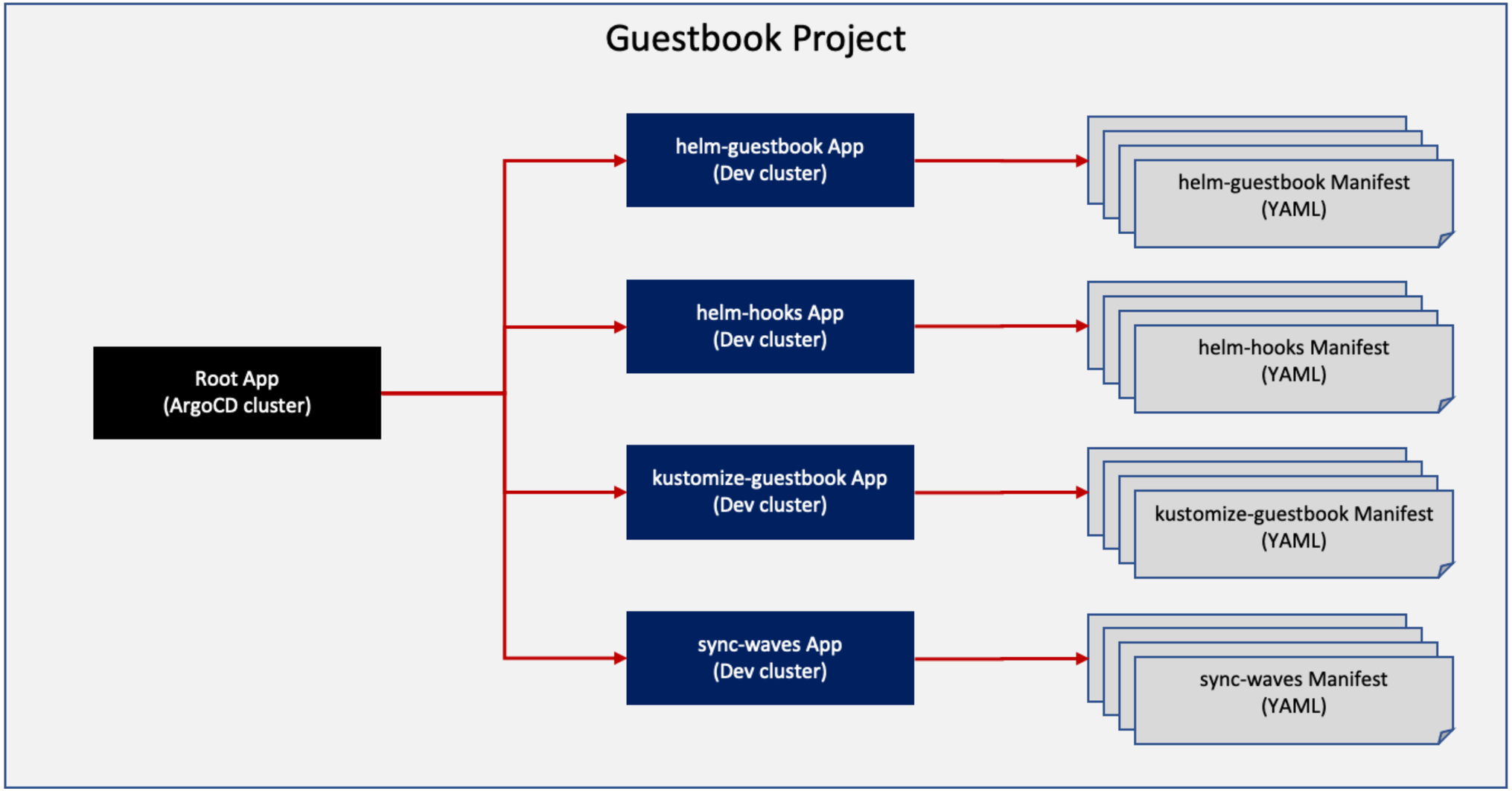
# Cluster bootstraping

# App of apps

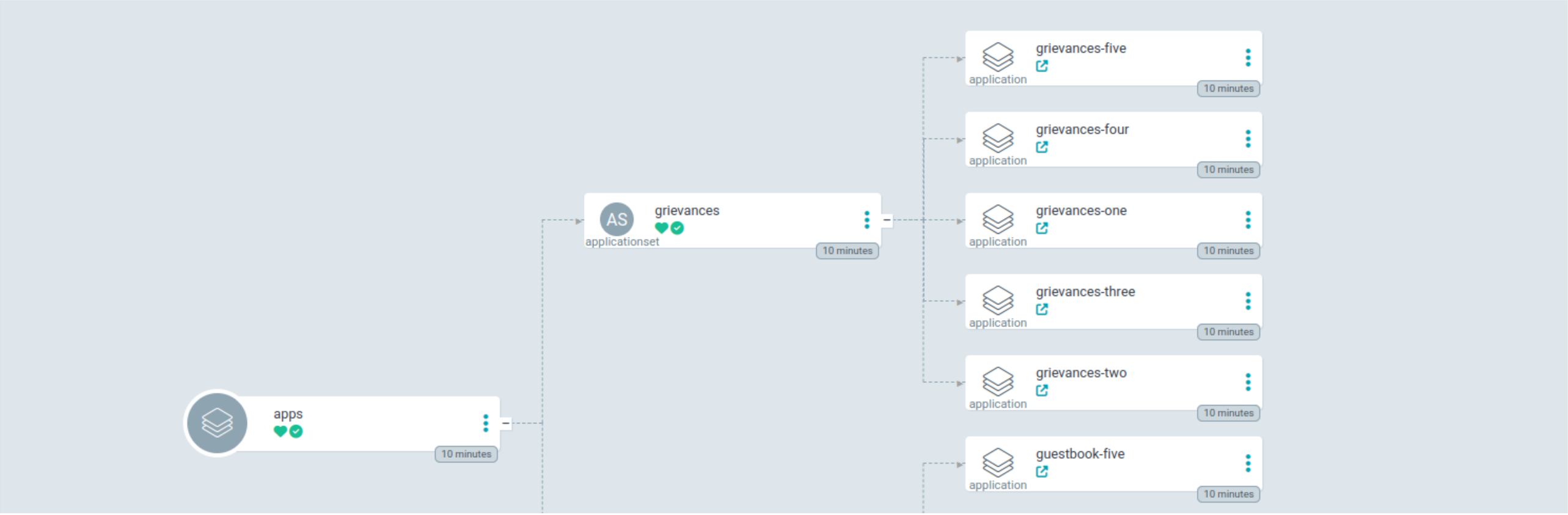
This guide is for operators who have already installed Argo CD, and have a new cluster and are looking to install many apps in that cluster.

There's no one particular pattern to solve this problem, e.g. you could write a script to create your apps, or you could even manually create them.
However, users of Argo CD tend to use the app of apps pattern.
Declaratively specify one Argo CD app that consists only of other apps.

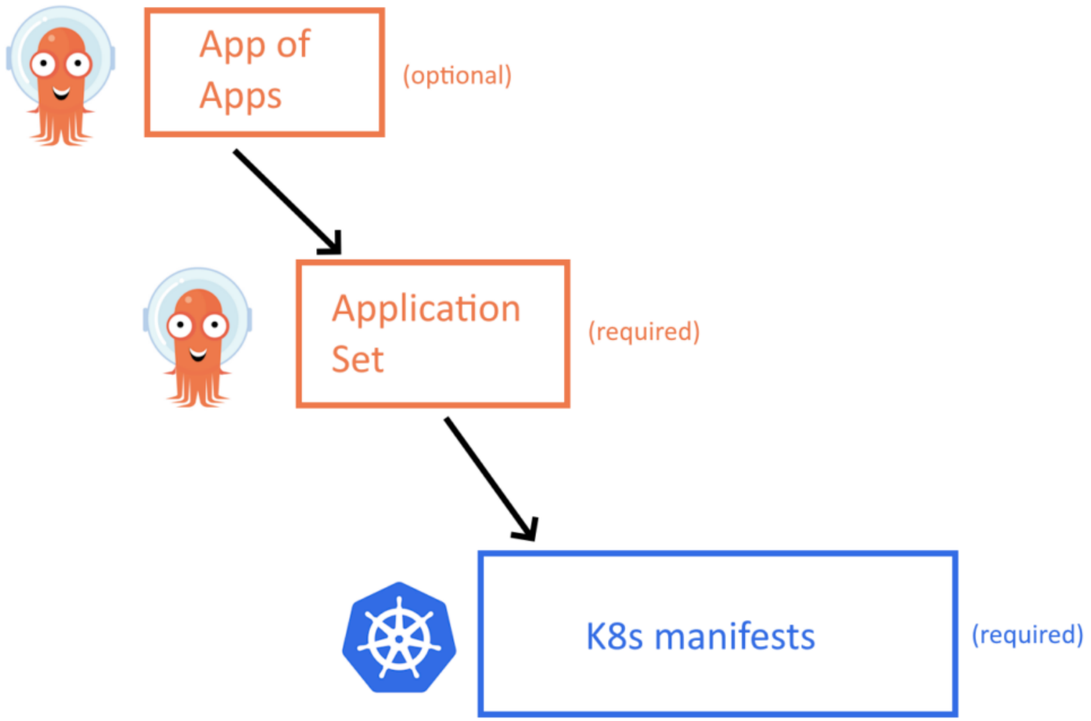


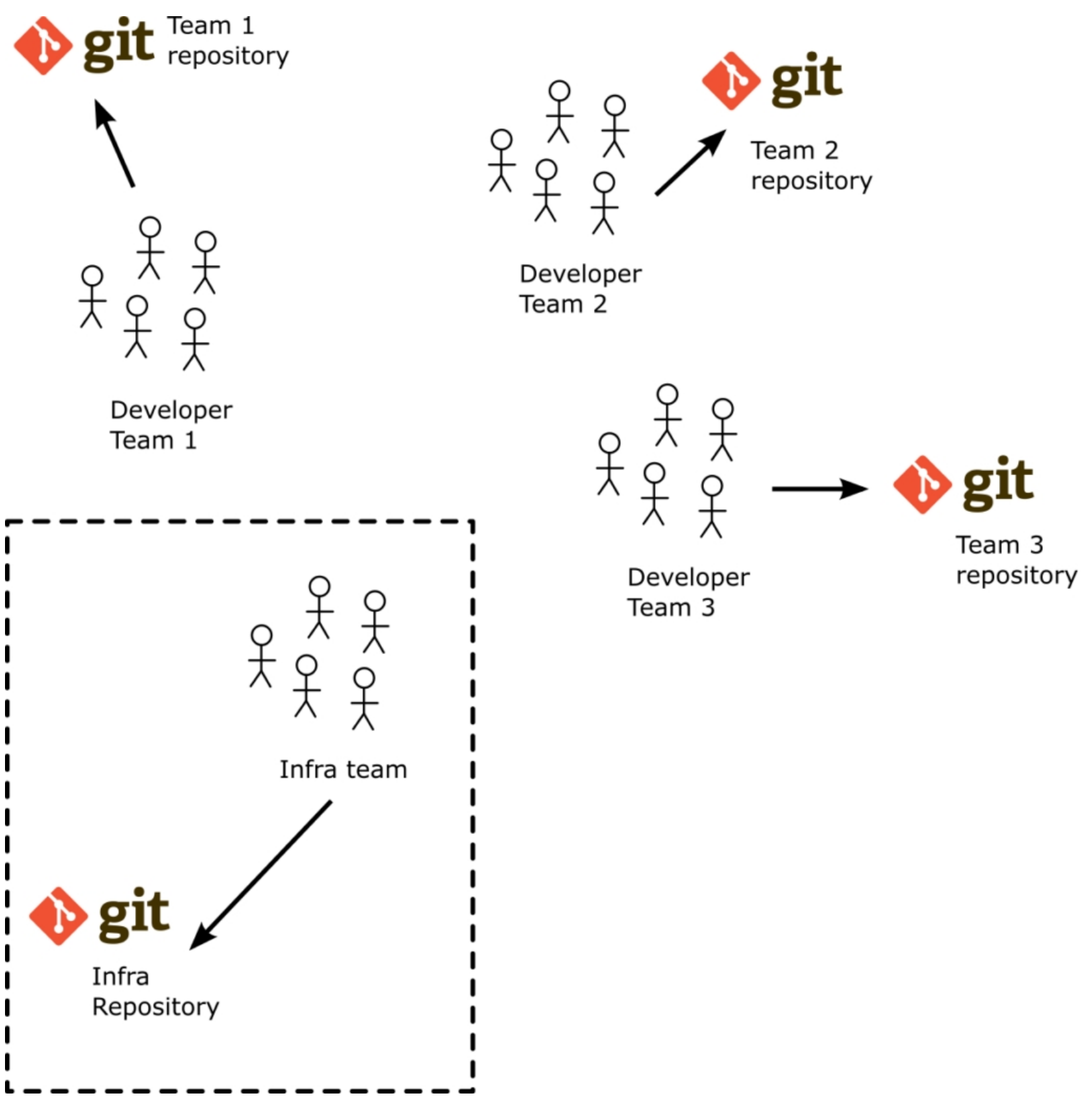

Guestbook Project
Root App
(ArgoCD cluster)
helm-guestbook App
(Dev cluster)
helm-hooks App
(Dev cluster)
kustomize-guestbook App
(Dev cluster)
sync-waves App
(Dev cluster)
helm-guestbook Manifest
(YAML)
helm-hooks Manifest
(YAML)
kustomize-guestbook Manifest
(YAML)
sync-waves Manifest
(YAML)

# App of apps with AppSet



# Best Practices for GitOps and ApplicationSet




git
Team 1
repository
Developer
Team 1
git
Team 2
repository
Developer
Team 2
Developer
Team 3
git
Team 3
repository
Infra team
git
Infra
Repository

# References


https://aws.amazon.com/what-is/iac/

https://argo-cd.readthedocs.io/en/stable/operator-manual/applicationset/

https://codefresh.io/blog/how-to-structure-your-argo-cd-repositories-using-application-sets/

https://skip.kartverket.no/blog/introducing-apps-repositories#what-are-applicationsets

https://codefresh.io/blog/argo-cd-best-practices/