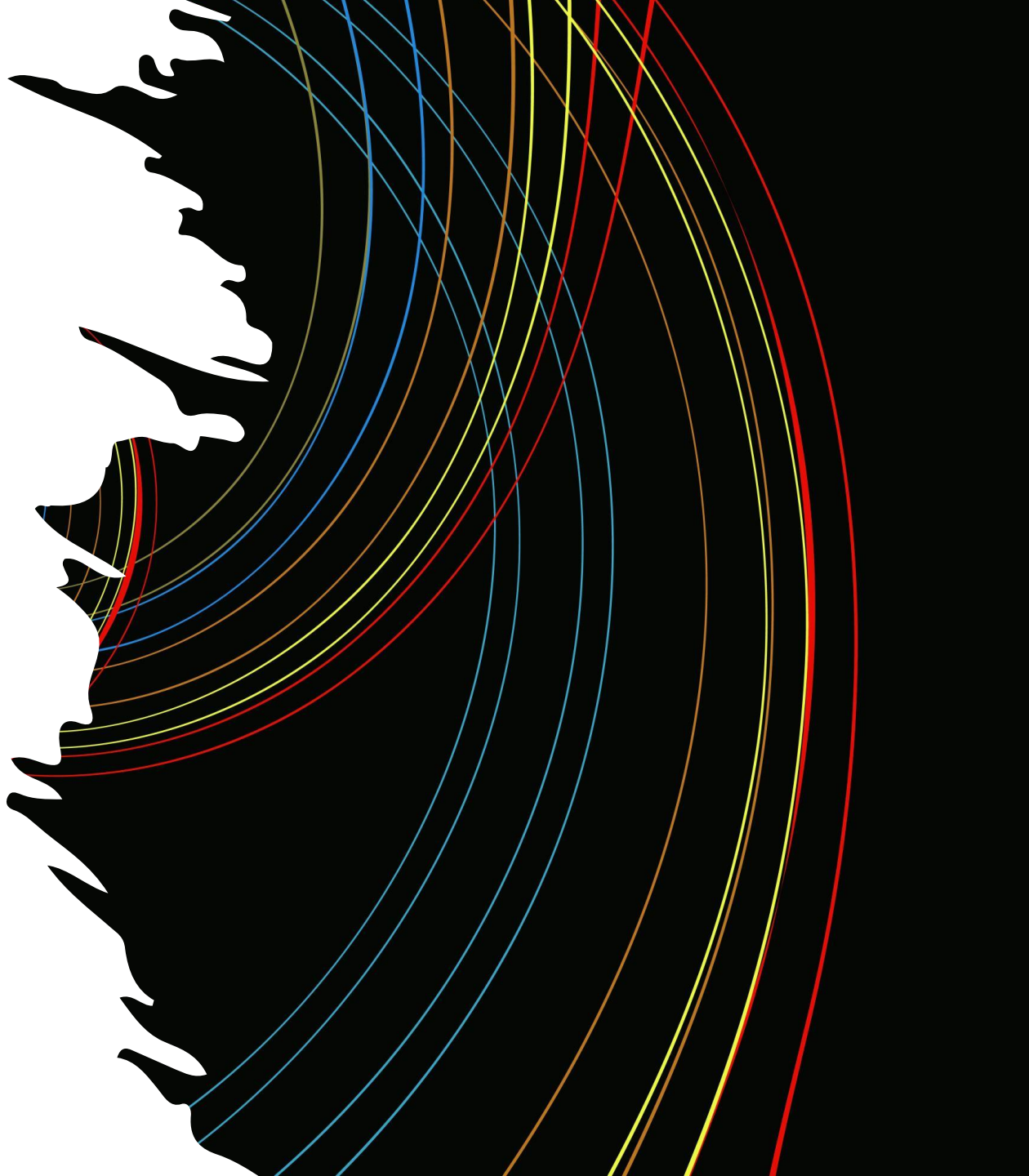
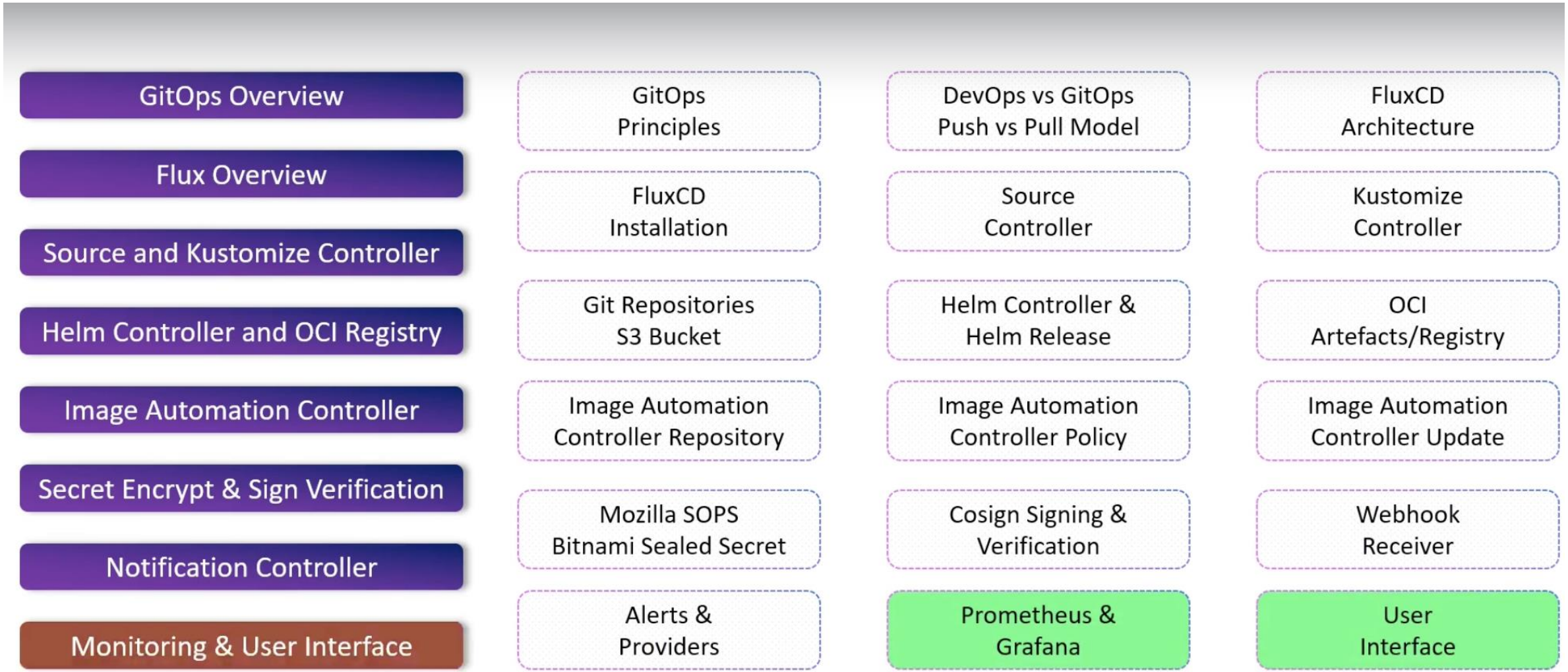
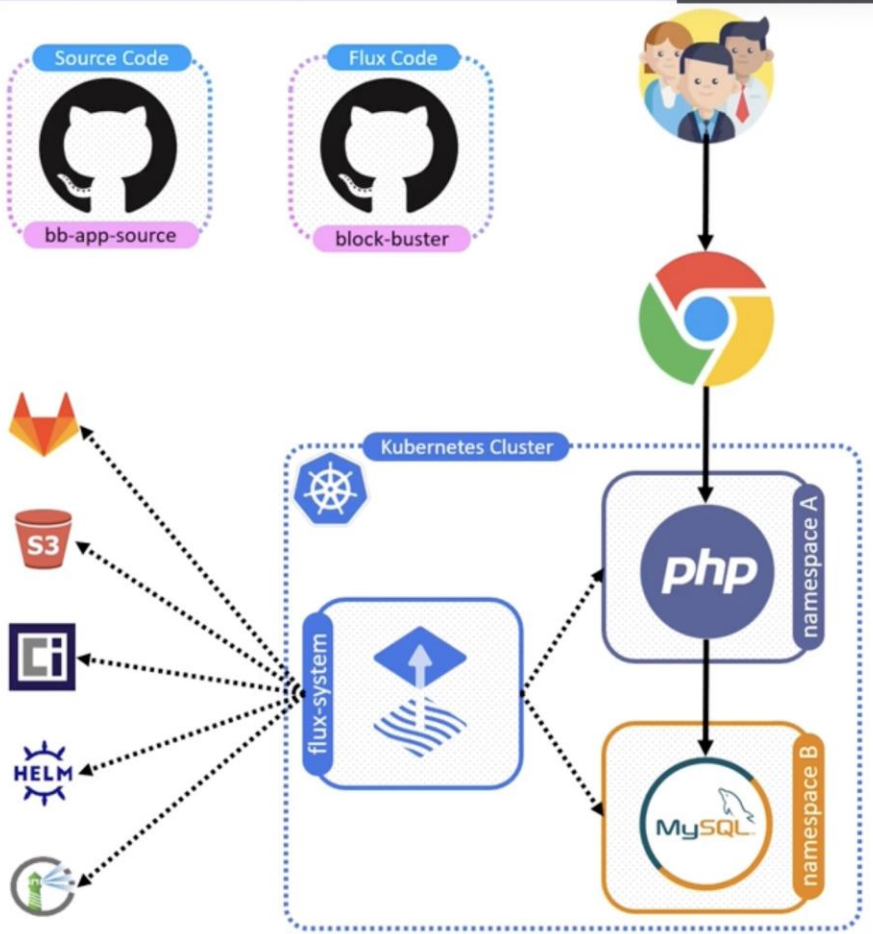
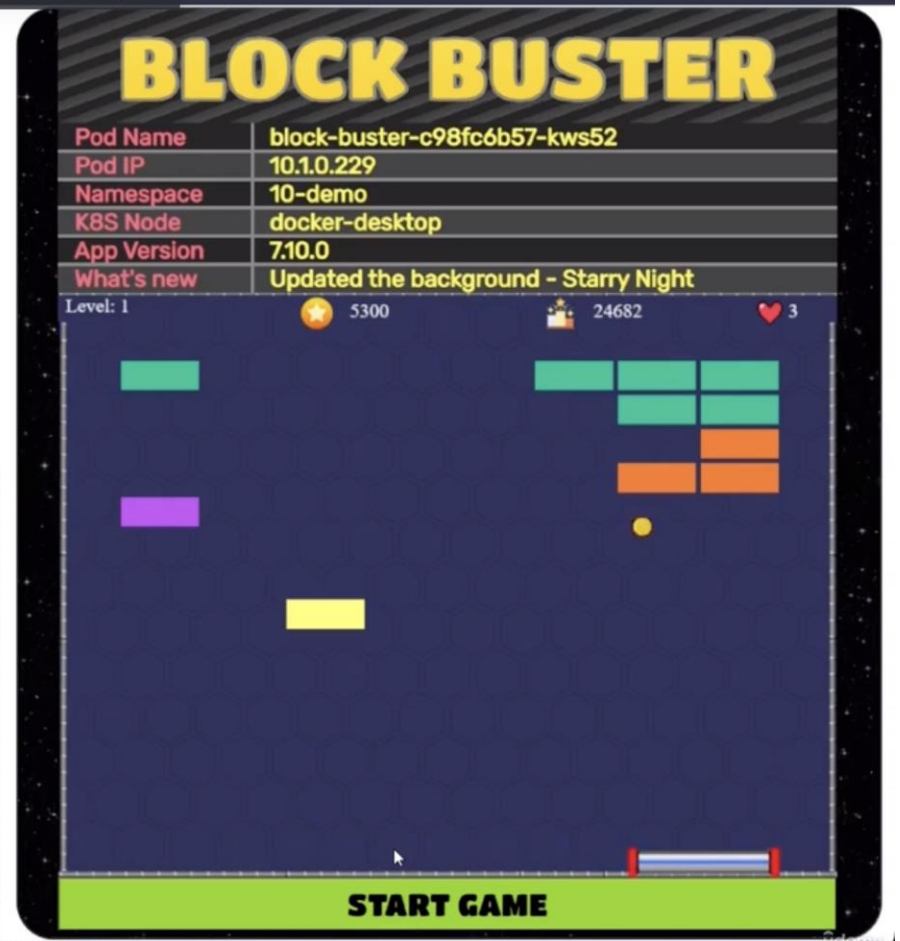
GitOps -FluxCD - Tutorial




GitOps Overview
Flux Overview
Source and Kustomize Controller
Helm Controller and OCI Registry
Image Automation Controller
Secret Encrypt & Sign Verification
Notification Controller
Monitoring & User Interface
GitOps Principles
FluxCD Installation
Git Repositories S3 Bucket
Image Automation Controller Repository
Mozilla SOPS Bitnami Sealed Secret
Alerts & Providers
DevOps vs GitOps Push vs Pull Model
Source Controller
Helm Controller & Helm Release
Image Automation Controller Policy
Cosign Signing & Verification
Prometheus & Grafana
FluxCD Architecture
Kustomize Controller
OCI Artefacts/Registry
Image Automation Controller Update
Webhook Receiver
User Interface

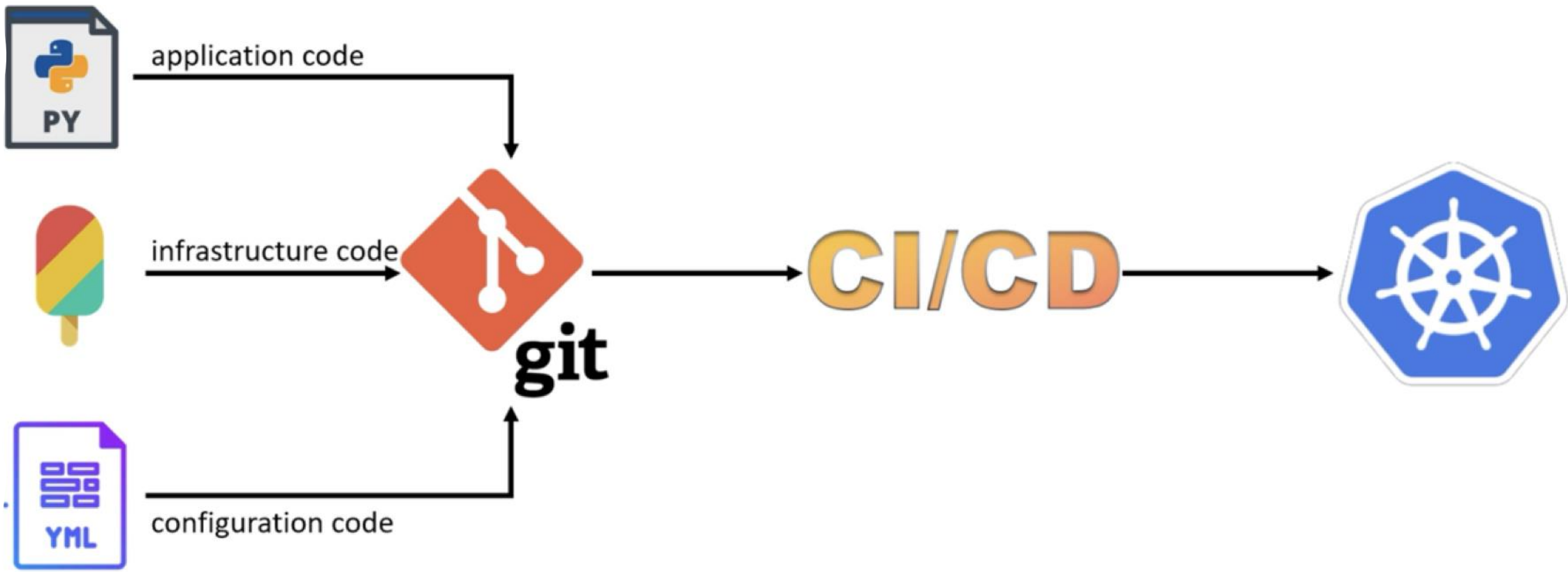
Application Architecture





GitOps is a modern cloud native approach to continuous deployment that leverages the Git version control.

Git acts as a single source of truth for all configuration, infrastructure and application code.



GitOps Principles

1

2

3

4

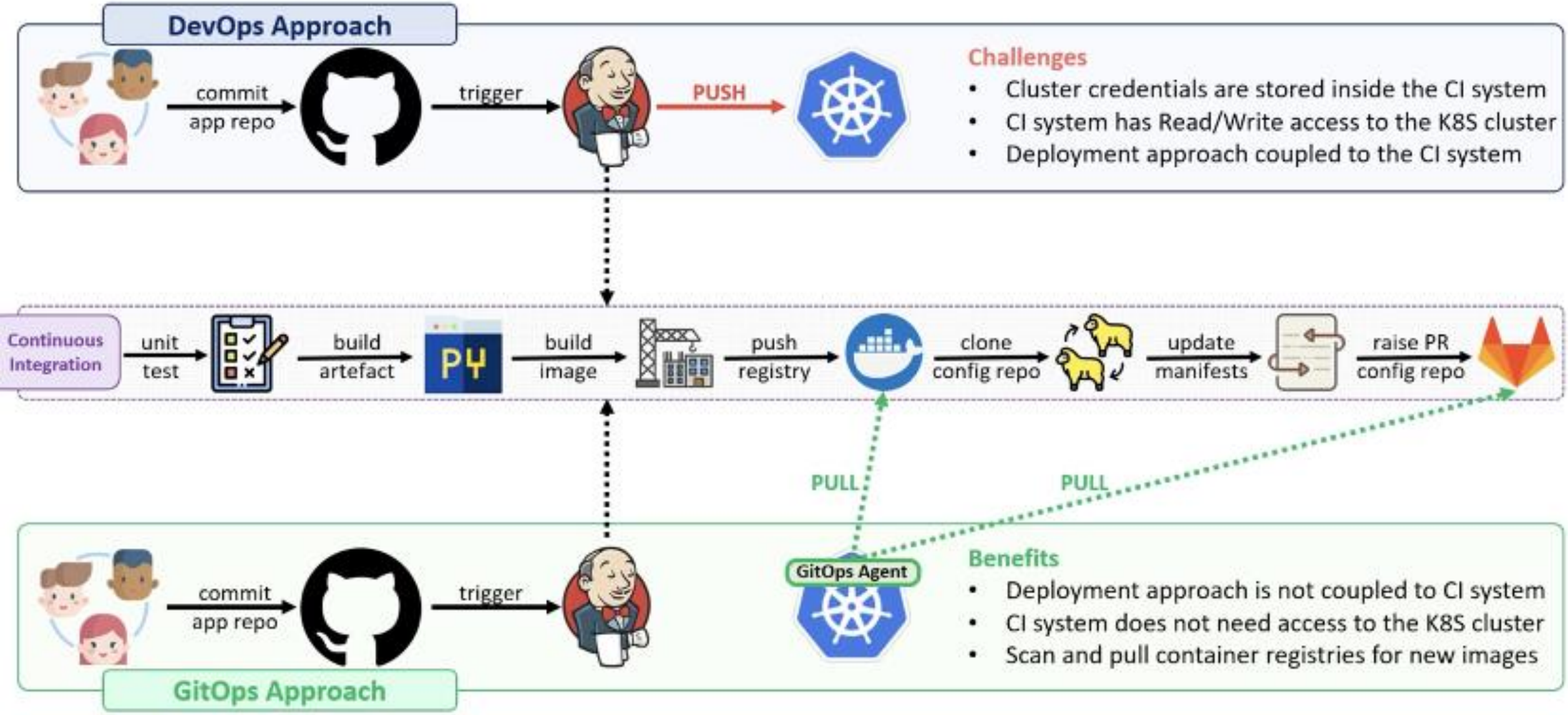
The entire system is described declarativetly

Desired state is versioned in a Git Repository

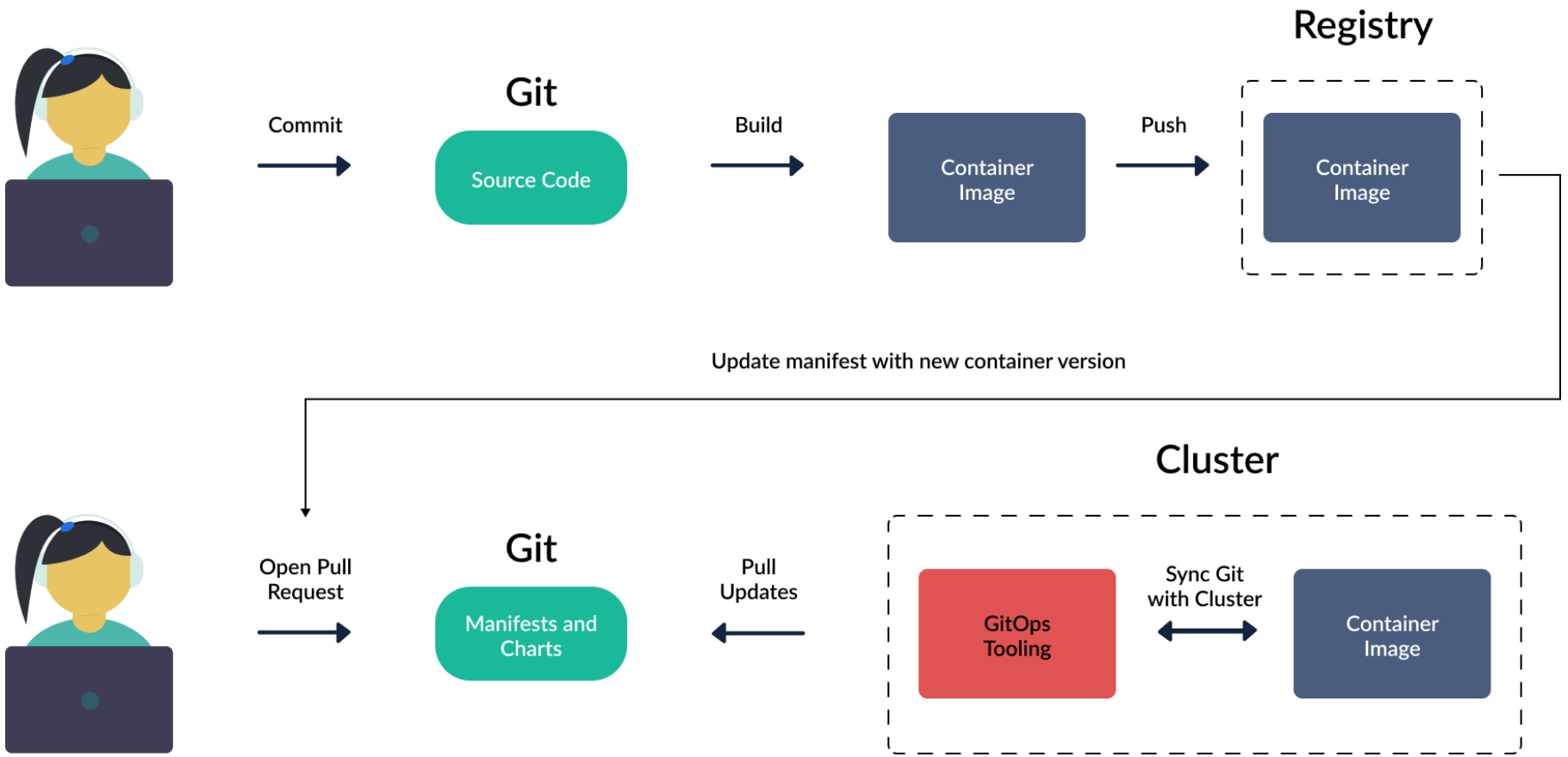
Any changes can be automatically pulled & applied to one or more targets

GitOps agents ensure correctness and reconcile the system to match the desired state

GitOps vs DevOps

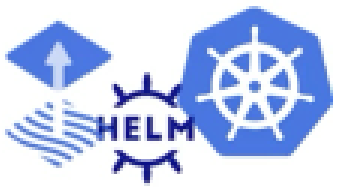


GitOps Workflow for Kubernetes



What/Why/How FluxCD?

What is FluxCD?	Flux is a tool for maintaining Kubernetes clusters in sync with configurations defined on source repositories like Git, Helm, Container Image, S3
	Continuously monitors running applications, comparing their live state to the desired state
	It reports the deviations to help developers manually or automatically synchronize the live state with the desired state



Why use FluxCD?	It extends the benefits of declarative specifications and Git-based configuration management
	It enables application deployment (CD) through automatic reconciliation and provides alerting, notifications
	It can even scan image repositories and update container images automatically before pushing/commiting them back to Git repository

How FluxCD works?	It follows the **GitOps** pattern by using Git repositories as the source of truth for the desired state of applications and infrastructure			
	Bootstrap	Git Repository	Helm Chart/Repository	OCI Repository	S3 Buckets
	Based on the resources it is either going to use Helm Controller or Kustomize Controller to deploy and manage applications			

FluxCD Features

1. Automated deployment (CD) of applications to specified target environment
2. Support for multiple config management/templating tools (Kustomize, Helm, OCI, Jsonnet, plain-YAML)
3. Works with all major Git providers, container registries, CI providers and S3-compatible buckets as a Source
4. Multi-tenancy and manage apps in either the same or other Kubernetes clusters
5. Flux provides alerting to external systems, and can handle external events.
6. Under the weave-gitops project, Weaveworks offers a free and open source GUI for Flux.
7. Out-of-the-box Prometheus metrics and Grafana Dashboards
8. Webhook integration (GitHub, BitBucket, GitLab)
9. CLI based for automation and CI integration
10. Automated or manual reconciliation of applications to its desired state
11. Automated configuration drift detection
12. Flux can manage and deploy any Kubernetes resource along with infrastructure dependencies

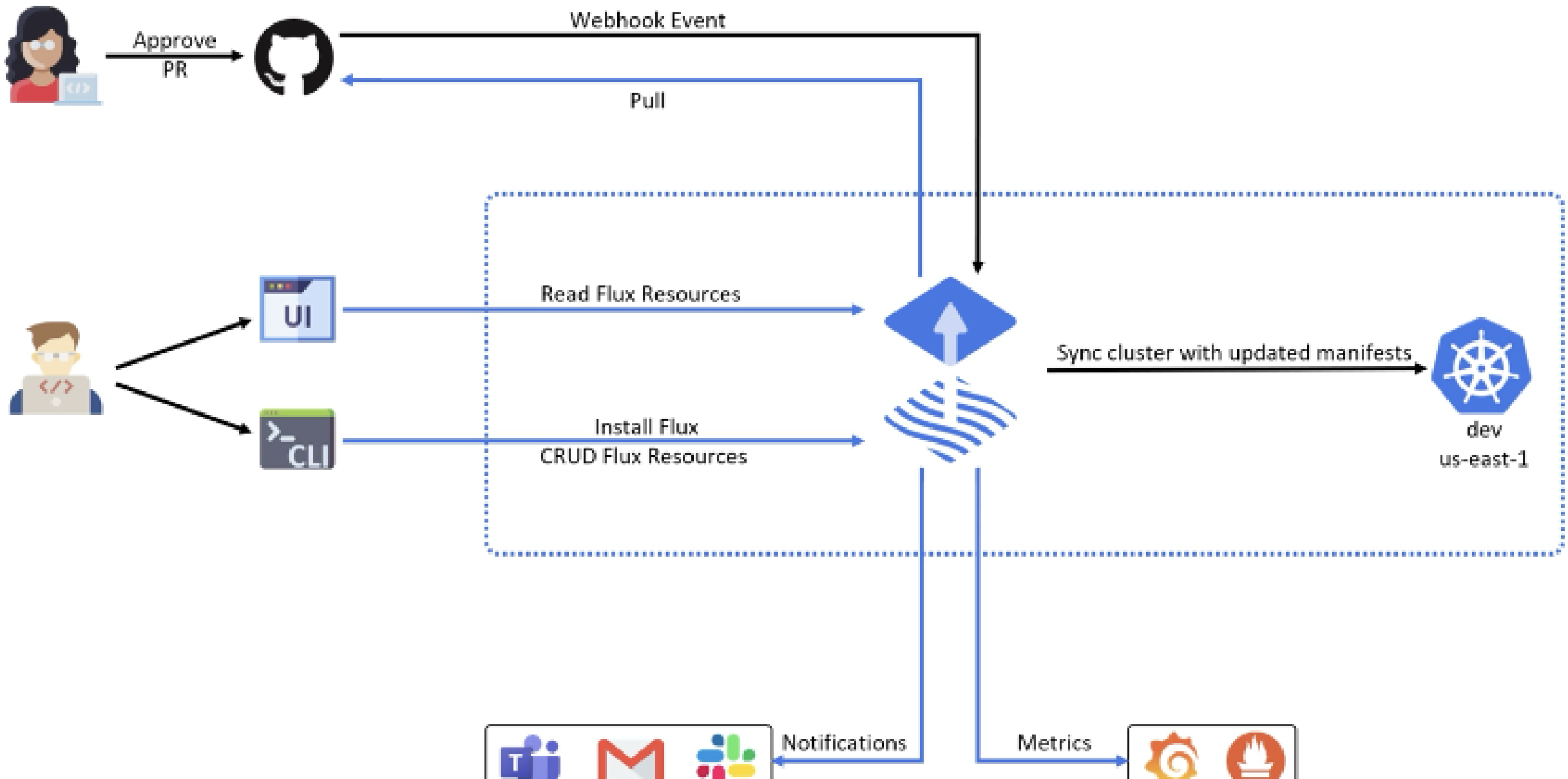
Flux Components

Flux is powered by the **GitOps Toolkit**, a set of composable APIs and specialized tools that enable a wide range of continuous delivery use-cases, from simple Kubernetes deployment pipelines to multi-tenant and multi-environment progressive delivery rollouts.

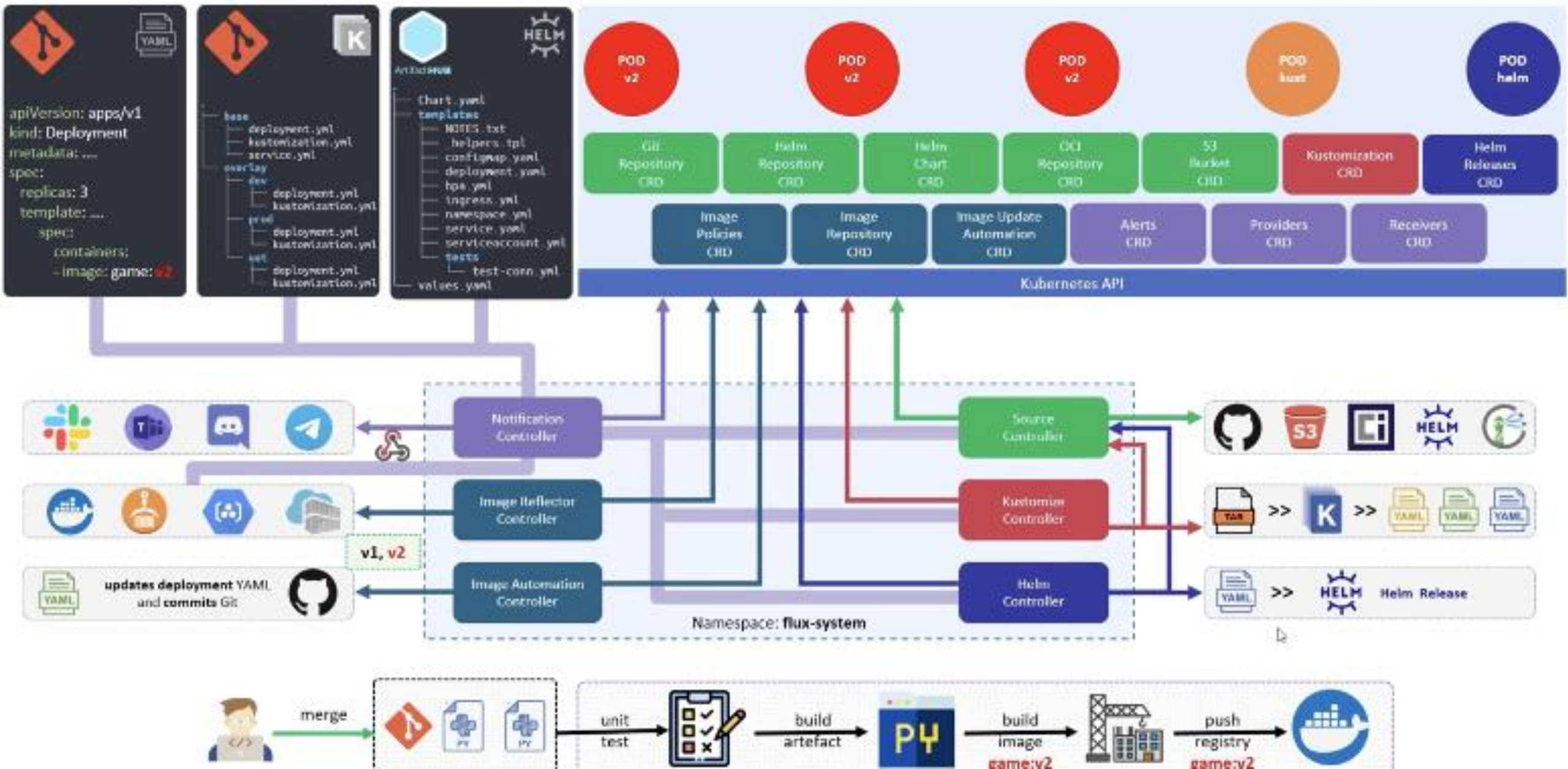
The Flux project is made out of the following components:

- **Flux CLI** - a command-line tool for installing, upgrading, operating, monitoring and debugging the Flux controllers running on Kubernetes clusters.
- **Flux Terraform Provider** - a provider for bootstrapping Flux with Terraform and OpenTofu.
- **Flux APIs** - a set of Kubernetes CRDs that allow defining continuous delivery workflows in a declarative manner.
- **Flux controllers** - a set of Kubernetes controllers that automate all aspects of continuous delivery based on the declarative workflows defined with the Flux APIs.

FluxCD Architecture



FluxCD Architecture



Flux Installation Options


```
► curl -s https://fluxcd.io/install.sh | sudo bash
```


```
► . <(flux completion bash)
```

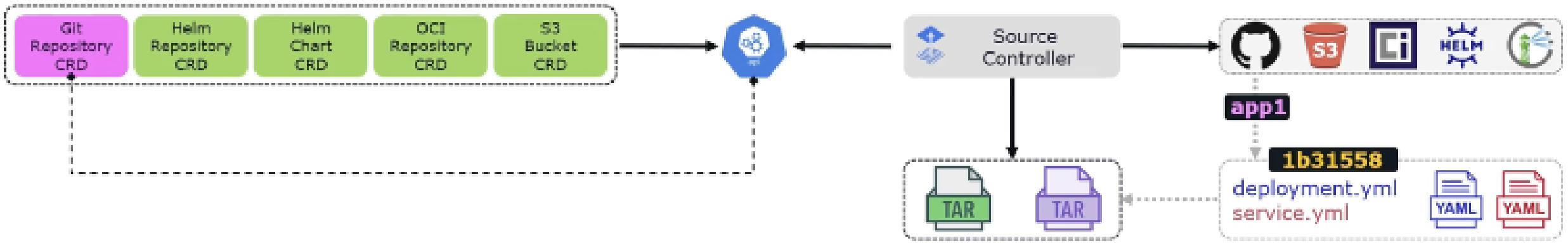
```
► flux bootstrap github \
  --owner sidd-harth \
  --repository brick-breaker \
  --path flux-clusters/dev-cluster \
  --personal true \
  --private false

Please enter GitHub personal access token (PAT):*********
► connecting to github.com
✔ repository created
✚ generating manifests
► installing components in flux-system namespace
deployment "source-controller" successfully rolled out
deployment "kustomize-controller" successfully rolled out
► configuring deploy key
✔ bootstrap finished
```

```
► git clone https://github.com/sidd-harth/brick-breaker.git
► cd brick-breaker
► tree
.
└── flux-clusters
    └── dev-cluster
        ├── flux-system
        │   ├── gotk-components.yaml
        │   ├── gotk-sync.yaml
        │   └── kustomization.yaml
        ├── game-deployment.yml
        ├── game-service.yml
        ├── application2-source.yml
        └── application2-kustomization.yml
```

Customize Flux Manifests	Bootstrap with Terraform	Dev/Testing Installation	Flux Uninstallation
automatic reconciliation	*registry.terraform.io*	*flux install*	*flux uninstall --namespace=flux-system*

Please note that any Helm releases or Kubernetes objects that Flux reconciled on the cluster will remain after using the uninstall command

Source Controller

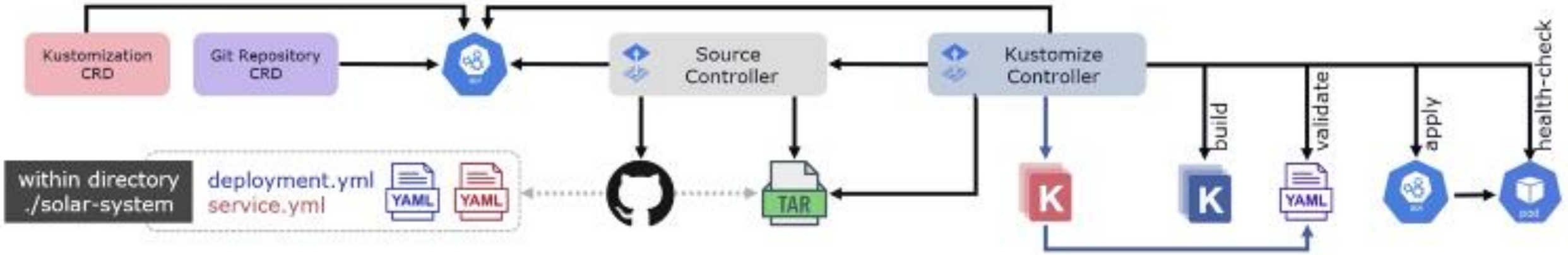



```
► flux create source git source-app1 \
--url https://github.com/sidd-harth/app1\
--interval 10s \
--branch main \
--export

apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: source-app1
  namespace: flux-system
spec:
  interval: 10s
  ref:
    branch: main
  url: https://github.com/sidd-harth/app1
```

```
► flux get sources git
NAME          REVISION          SUSPENDED  READY  MESSAGE
flux-system   main/7e35678      False      True   stored artifact revision 'main/7e35674...'
source-app1   main/1b31558      False      True   stored artifact revision 'main/1b31558...'
```

```
► kubectl -n flux-system exec -it source-controller-549d-mhcb8 -- /bin/sh

~ $ ll data/gitrepository/flux-system/source-app1
1b31558bb1a701b3e454d97a2ec7592652bbc9e3.tar.gz
latest.tar.gz

~ $ tar -tf data/gitrepository/flux-system/source-app1/latest.tar.gz
deployment.yml
service.yml
```

Kustomization Controller

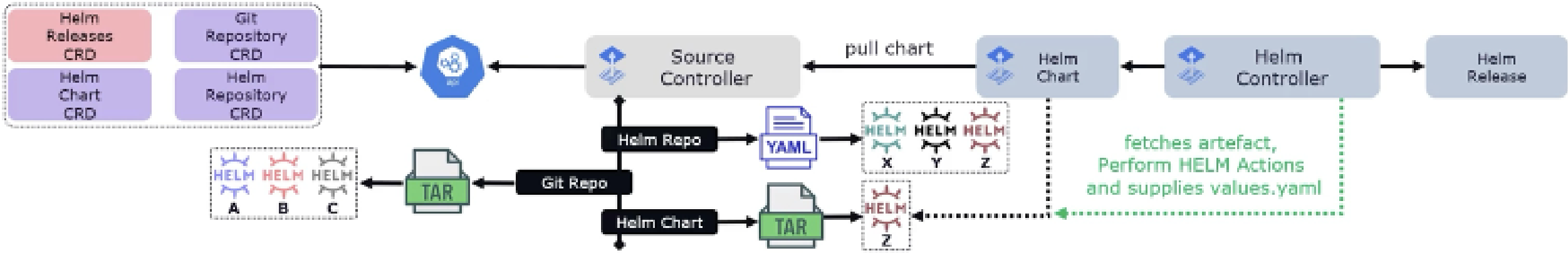



```
► flux create kustomization kustomization-app1 \
--source GitRepository/source-app1 \
--path ./solar-system \
--prune true \
--export
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: kustomization-app1
  namespace: flux-system
spec:
  interval: 1m0s
  path: ./solar-system
  prune: true
  sourceRef:
    kind: GitRepository
    name: source-app1
```

```
► kustomize create --autodetect --recursive  ## spits out a kustomization.yaml
► cat kustomization.yaml
apiVersion: kustomize.config.k8s.io/v1beta1  ## used to configure Kustomize overlay
kind: Kustomization
namespace: default
resources:
  - deployment.yaml
  - service.yaml
```

```
► flux get kustomizations
NAME                REVISION          SUSPENDED  READY  MESSAGE
flux-system         main/7e35678      False      True   Applied revision: 'main/7e35674...'
kustomization-app1  main/1b31558      False      True   Applied revision: 'main/1b31558...'
```

Helm Controller




```
► flux create source git my-helm-charts \
--url https://github.com/sidd-harth/charts \
--branch main
```

```
► flux create source helm bitnami \
--url https://charts.bitnami.com/bitnami \
--cert-file=./cert.crt \
--key-file=./key.crt \
--ca-file=./ca.crt
```

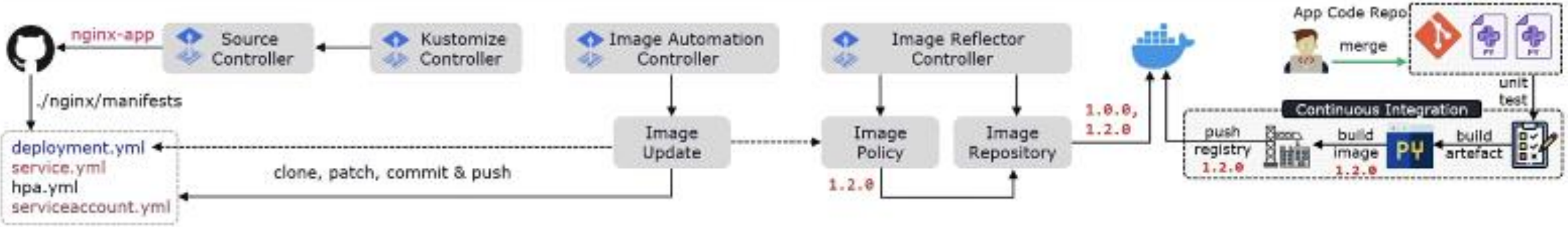
```
► flux create helmrelease chart-z-release \
  --source HelmRepository/bitnami \
  --chart chart-z \
  --chart-version=1.2.3 \
  --values values.yml
```

Flux HelmChart

```
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmChart
metadata:
  name: flux-system-chart-z-release
spec:
  interval: 1m0s
  chart: chart-z
  reconcileStrategy: ChartVersion
  sourceRef:
    kind: HelmRepository
    name: bitnami
  version: '1.2.3'
status:
  artifact:
    path: helmchart/flux-system/flux-system-chart-
          z-release/chart-z-release-1.2.3.tgz
    revision: 1.2.3
    url: http://source-controller.flux-
         system.svc.cluster.local./helmchart/flux-
         system/flux-system-chart-z-release/chart-
         z-release-1.2.3.tgz
```

```
► k -n flux-system exec -it source-controller -- sh

~ $ tree data/
├── gitrepository
│   └── flux-systen
│       └── my-helm-charts
│           ├── 1b31558bb1a701c7592652bbc9e3.tar.gz
│           └── latest.tar.gz
├── helmrepository
│   └── flux-systen
│       └── bitnami
│           ├── index-e6dc924894f5f871db9b968.yaml
│           └── index.yaml
└── helmchart
    └── flux-systen
        └── flux-system-chart-z-release
            ├── chart-z-release-1.2.3.tgz
            └── latest.tar.gz
```

Image Automation Controller



deployment.yaml

```yaml
apiVersion: apps/v1
kind: Deployment
metadata: ....
spec:
 selector: ....
 template: ....
  spec:
   containers:
   - name: nginx
     image: sid/nginx:1.2.0 # {"$imagepolicy":"flux-system:nginx-policy"}
```

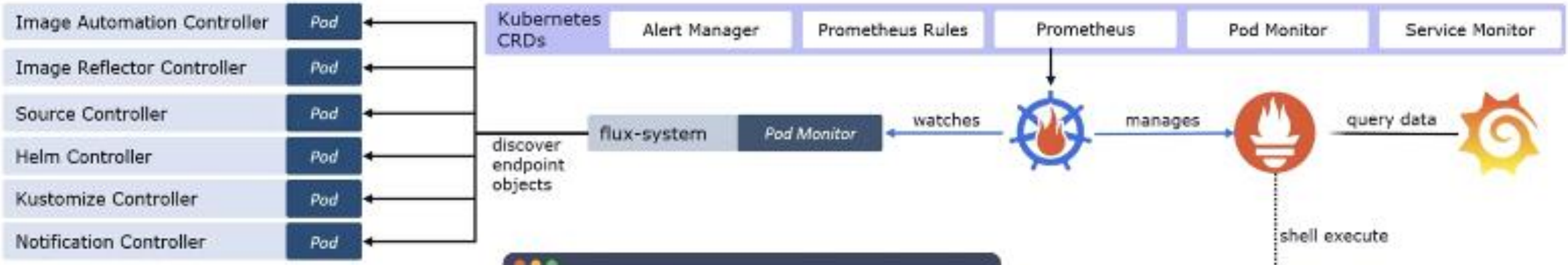
```
► flux create image repository nginx-repo \
--image=docker.io/sid/nginx \
--interval=1m
```

```
► flux create image policy nginx-policy \
--image-ref=nginx-repo \
--select-semver=1.x.0
```

```
► flux create image update nginx-update \
--git-repo-ref=nginx-app \
--git-repo-path="./nginx/manifests" \
--checkout-branch=main \
--push-branch=main \
--author-name=fluxcdbot \
--author-email=fluxcdbot@users.noreply.github.com \
--commit-template="{{range .Updated.Images}}{{println .}}{{end}}"
```

```
► flux get image all
NAME                                    LAST SCAN                       SUSPENDED    READY    MESSAGE
imagerepository/nginx-repo              2022-11-23T14:21:10+05:30       False        True     successful scan, found 2 tags
NAME                                    LATEST IMAGE                                 READY    MESSAGE
imagepolicy/nginx-policy                docker.io/sid/nginx:1.0.1                    False    Latest image tag for 'docker.io/sid/nginx' resolved to: 1.2.0
NAME                                    LAST RUN                        SUSPENDED    READY    MESSAGE
imageupdateautomation/nginx-update      2022-11-23T14:21:04+05:30       False        True     committed and pushed 86a9ac42ba524ca543f07bd4872c357ba to main
```

Monitoring with Prometheus + Grafana




```
► kubectl get pod source-controller-.. -o yaml
apiVersion: v1
kind: Pod
metadata:
  namespace: flux-system
  annotations:
    prometheus.io/port: "8080"
    prometheus.io/scrape: "true"
  labels:
    app: source-controller
  name: source-controller-5495bf4d7d-nws57
spec:
  containers:
    image: ../../source-controller:v0.31.0
    ports:
    - containerPort: 8080
      name: http-prom
      protocol: TCP
```

```
► kubectl get podmonitor flux-system -o yaml
apiVersion: monitoring.coreos.com/v1
kind: PodMonitor
metadata:
...
spec:
  namespaceSelector:
    matchNames:
      - flux-system
  selector:
    matchExpressions:
      - key: app
        operator: In
        values:
          - helm-controller
          - source-controller
          - kustomize-controller
          - notification-controller
          - image-automation-controller
          - image-reflector-controller
  podMetricsEndpoints:
    - port: http-prom
```

```
► kubectl exec -it prometheus-0 -c config-reloader \
-- /bin/sh
~ $ cat /etc/prometheus/config_out/prometheus.env.yaml
global:
  scrape_interval: 30s
rule_files:
  /etc/prometheus/rules/prometheus-0/*.yaml
scrape_configs:
- job_name: serviceMonitor/monitoring/kube-etcd/0
- job_name: serviceMonitor/monitoring/kube-proxy/0
- job_name: serviceMonitor/monitoring/operator0
- job_name: serviceMonitor/monitoring/prometheus/0
- job_name: serviceMonitor/monitoring/coredns/0
- job_name: serviceMonitor/monitoring/apiserver/0
- job_name: serviceMonitor/monitoring/kubelet/0
- job_name: serviceMonitor/monitoring/kubelet/1
- job_name: podMonitor/monitoring/flux-system/0
```

Source Controller

The main role of the source management component is to provide a common interface for artifacts acquisition. The source API defines a set of Kubernetes objects that cluster admins and various automated operators can interact with to offload the Git and Helm repositories operations to a dedicated controller.

Kustomize Controller

The kustomize-controller is a Kubernetes operator, specialized in running continuous delivery pipelines for infrastructure and workloads defined with Kubernetes manifests and assembled with Kustomize.

Flux CD Components

Helm Controller

The Helm Controller is a Kubernetes operator, allowing one to declaratively manage Helm chart releases with Kubernetes manifests.
The desired state of a Helm release is described through a Kubernetes Custom Resource named HelmRelease. Based on the creation, mutation or removal of
a HelmRelease resource in the cluster, Helm actions are performed by the controller.

Notification Controller

The Notification Controller is a Kubernetes operator, specialized in handling inbound and outbound events.
The controller handles events coming from external systems (GitHub, GitLab, Bitbucket, Harbor, Jenkins, etc) and notifies the GitOps toolkit controllers about source changes.
The controller handles events emitted by the GitOps toolkit controllers (source, kustomize, helm) and dispatches them to external systems (Slack, Microsoft Teams, Discord, Rocker) based on event severity and involved objects.

Flux CD Ecosystem

There are many other tools that came to light to extend Flux CD capabilities, including:’

Visual Studio Code Extension

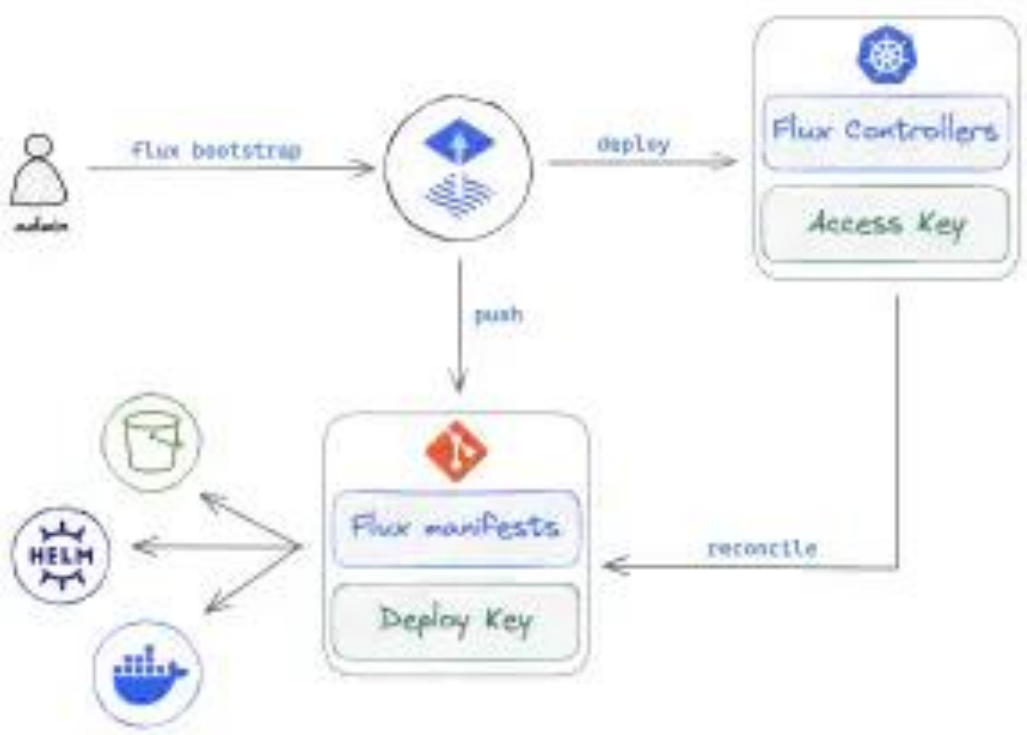
The Visual Studio Code Extension is built on Flux CD and can be used to visualize, configure, and debut Flux objects (sources and workloads) needed for GitOps workflows. This extension dramatically simplifies application deployment for developers, enabling them to deploy code directly from VS Code without needing to be an expert on Kubernetes. Check the **Visual Studio Marketplace** for the complete list of features and installation instructions.

Terraform Controller

The Terraform Controller is a specialized controller integrated with Flux designed to apply GitOps principles to Terraform resources. Leveraging the combined capabilities of Flux and Terraform, TF-controller allows you to adopt GitOps practices for both infrastructure and application resources within the Kubernetes and Terraform ecosystem. Check out the **documentation** and **use cases**.

Flux bootstrap

- Bootstrap is the process of deploying the Flux controllers on a Kubernetes cluster and configuring them to watch a Git repository for changes. The bootstrap repository can contain references to other Git repos, OCI repos, Helm charts, S3-compatible buckets; together all these sources form the desired state of the cluster.
- With Flux running on the cluster, all changes to the desired state are automatically reconciled, including the self-update of the Flux controllers. If the cluster state drifts from the desired state, Flux will automatically correct it, effectively undoing any changes made to the cluster outside of the GitOps workflow.



Pros & Cons

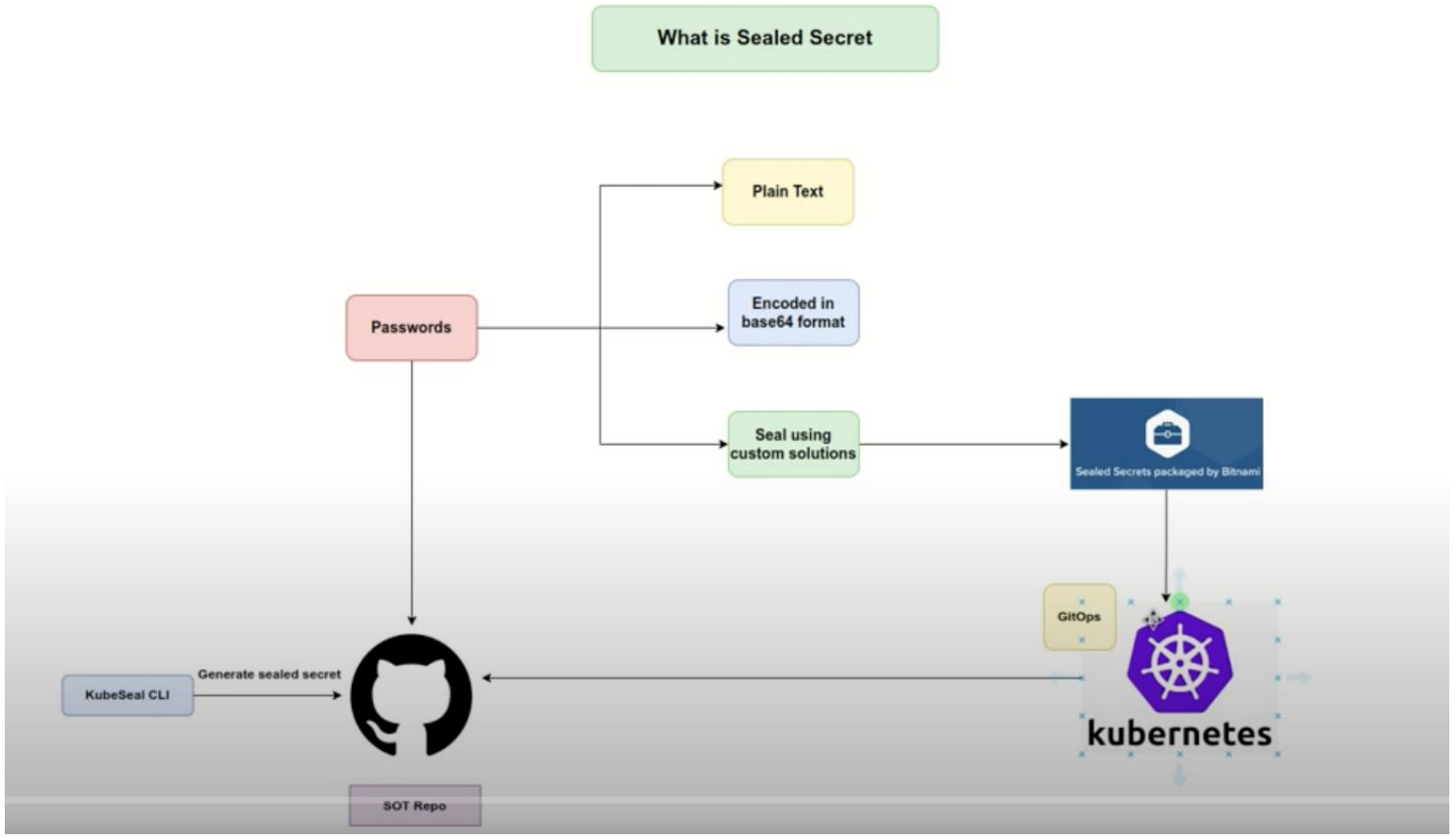
Pros:

- Reduced attack surface, the API server of each cluster doesn't need to be exposed to external systems.
- Reduced blast radius, each cluster is self-sufficient and can operate independently.
- Suitable for hard multi-tenancy and air-gapped environments where clusters can't communicate with each other.

Cons:

- Operational overhead, each cluster needs to be bootstrapped with Flux separately.
- Maintenance overhead, each Flux instance needs to be updated independently.
- Monitoring and observability overhead, each Flux instance needs to be monitored separately, the collected metrics and events need to be aggregated.

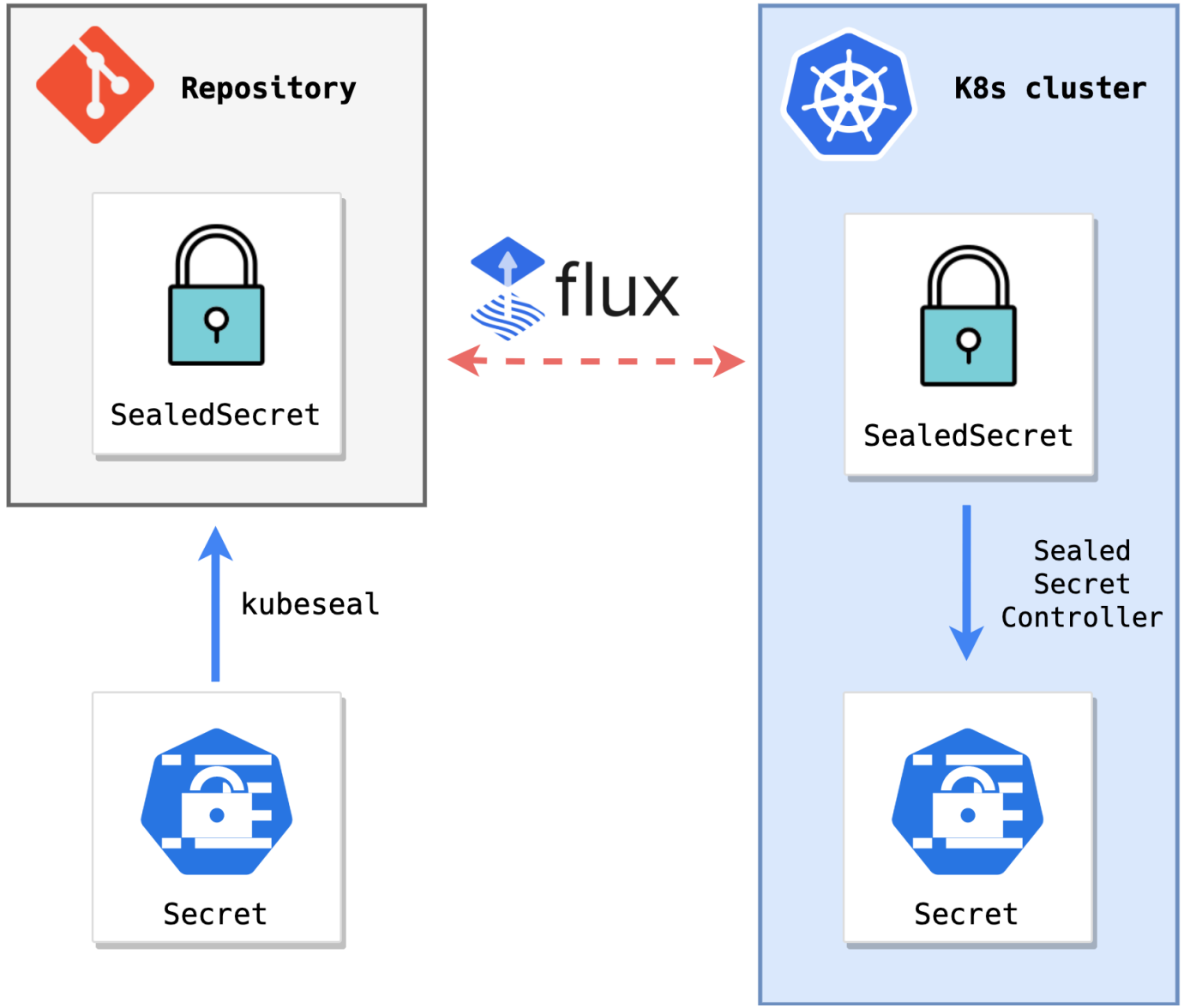
Sealed Secrets

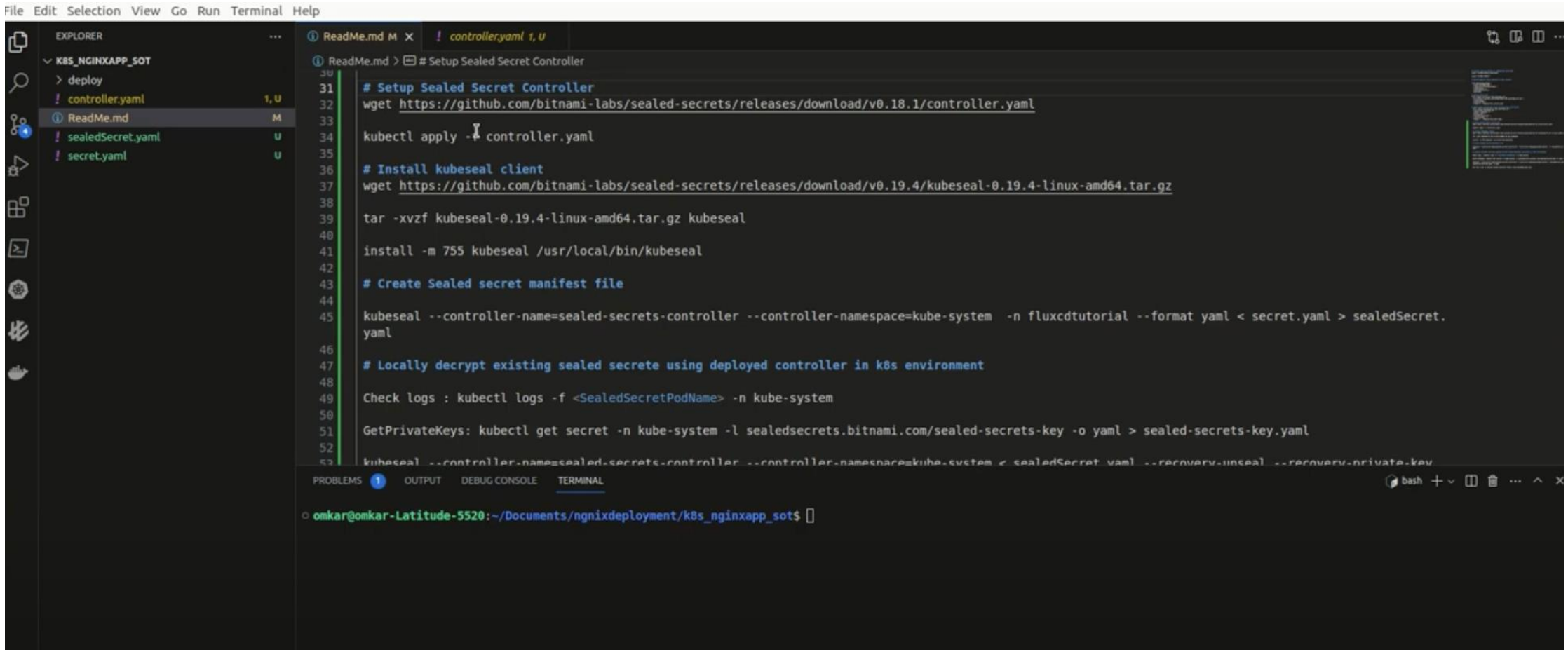


Deploy and use Bitnami Sealed Secrets in Flux

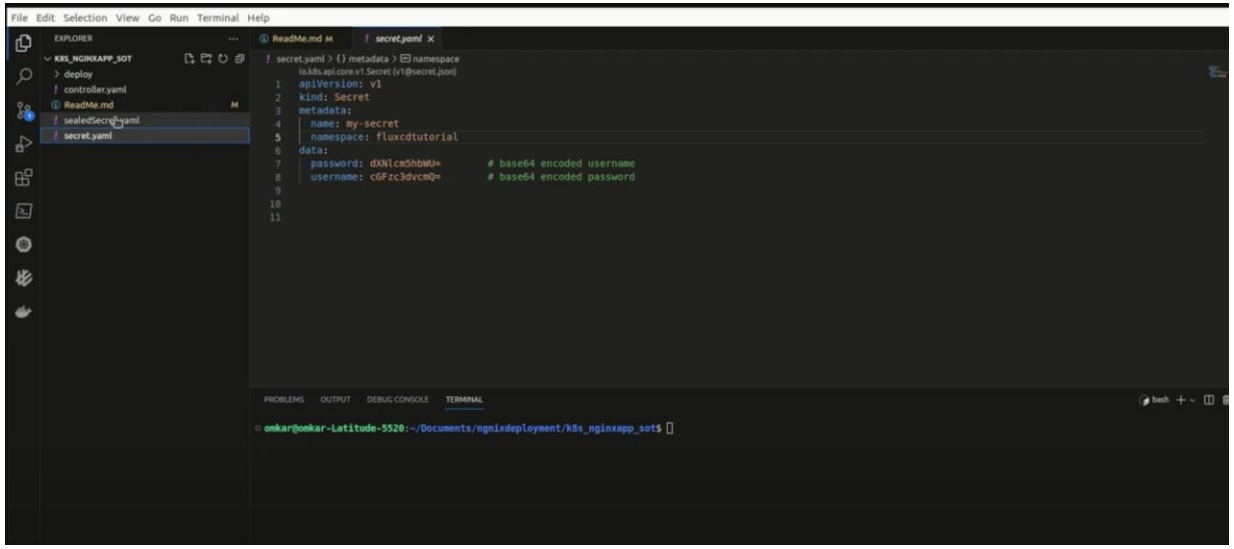
Assuming you already have a Kubernetes cluster synchronized with a repository, to use Sealed Secrets you need to:

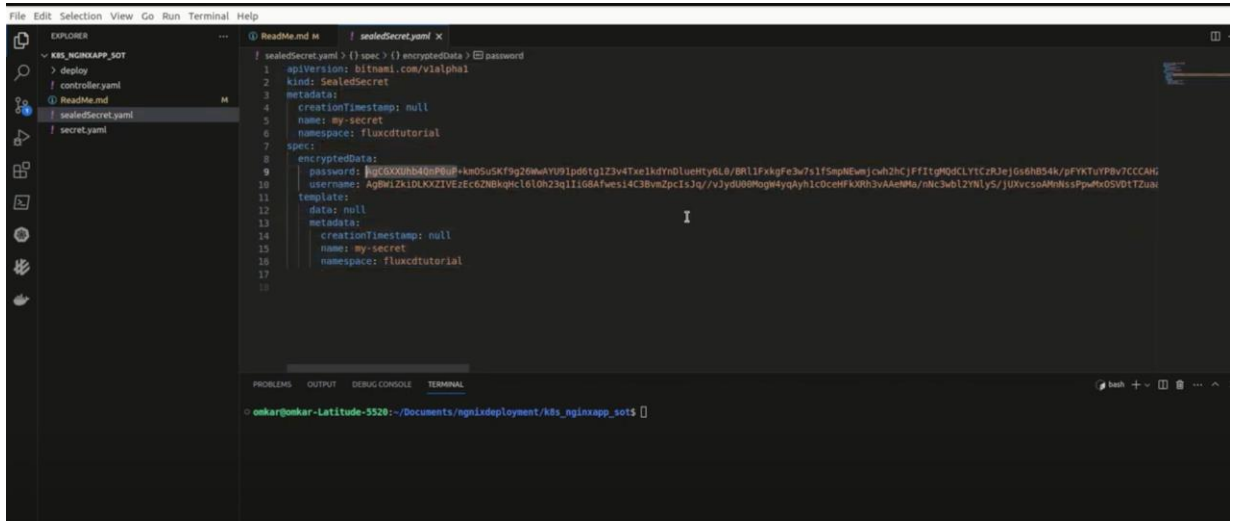
1. Install kubeseal
2. Deploy (via GitOps, of course) the Sealed Secrets controller
3. Retrieve the public key from the Sealed Secrets controller
4. Use the public key to encrypt a regular Kubernetes Secret object into a Sealed Secret.
5. Commit the Sealed Secret to the repository
6. Wait for the Sealed Secret to be deployed and decrypted by the Sealed Secrets controller


Repository
SealedSecret
flux
K8s cluster
SealedSecret
Sealed
Secret
Controller
kubeseal
Secret
Secret


File Edit Selection View Go Run Terminal Help
EXPLORER
K8S_NGINXAPP_SOT
deploy
controller.yaml 1, U
ReadMe.md M
sealedSecret.yaml U
secret.yaml U
ReadMe.md M
controller.yaml 1, U
ReadMe.md > # Setup Sealed Secret Controller
Setup Sealed Secret Controller
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.18.1/controller.yaml
kubectl apply -f controller.yaml
Install kubeseal client
wget https://github.com/bitnami-labs/sealed-secrets/releases/download/v0.19.4/kubeseal-0.19.4-linux-amd64.tar.gz
tar -xvzf kubeseal-0.19.4-linux-amd64.tar.gz kubeseal
install -m 755 kubeseal /usr/local/bin/kubeseal
Create Sealed secret manifest file
kubeseal --controller-name=sealed-secrets-controller --controller-namespace=kube-system -n fluxcdtutorial --format yaml < secret.yaml > sealedSecret.
yaml
Locally decrypt existing sealed secrete using deployed controller in k8s environment
Check logs : kubectl logs -f <SealedSecretPodName> -n kube-system
GetPrivateKeys: kubectl get secret -n kube-system -l sealedsecrets.bitnami.com/sealed-secrets-key -o yaml > sealed-secrets-key.yaml
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL
bash
omkar@omkar-Latitude-5520:~/Documents/ngnixdeployment/k8s_nginxapp_sot$

Secret file


File Edit Selection View Go Run Terminal Help
EXPLORER
K8S_NGINXAPP_SOT
deploy
controller.yaml
ReadMe.md
sealedSecret.yaml
secret.yaml
ReadMe.md M
secret.yaml
secret.yaml > {} metadata > namespace
io.k8s.api.core.v1.Secret (v1@secret.json)
apiVersion: v1
kind: Secret
metadata:
name: my-secret
namespace: fluxcdtutorial
data:
password: dXNlcm5hbWU= # base64 encoded username
username: cGFzc3dvcmQ= # base64 encoded password
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash
omkar@omkar-Latitude-5520:~/Documents/ngnixdeployment/k8s_nginxapp_sot$


File Edit Selection View Go Run Terminal Help
EXPLORER
K8S_NGINXAPP_SOT
deploy
controller.yaml
ReadMe.md
sealedSecret.yaml
secret.yaml
ReadMe.md M
sealedSecret.yaml
sealedSecret.yaml > {} spec > {} encryptedData > password
apiVersion: bitnami.com/v1alpha1
kind: SealedSecret
metadata:
creationTimestamp: null
name: my-secret
namespace: fluxcdtutorial
spec:
encryptedData:
password: AgCGXXUhb4QnP0uP+kmO5u5Kf9g26WwAYU91pd6tg1Z3v4Txe1kdYnDlueHty6L0/BRl1FxkgFe3w7s1fSmpNEwmjcwh2hCjFfItgMQdCLYtCzRJejGs6hB54k/pFYKTuYP0v7CCCAHG
username: AgBWiZkiDLKXZIVEzEc6ZNBkqHcl6l0h23q1Ii68Afwesi4C3BvmZpcIsJq/vJydU00MogW4yqAyh1cOceHFkXRh3vAAeNMa/nNc3wbl2YNlyS/jUXvcsoAMnNssPpwMtxOSVDtTZua
template:
data: null
metadata:
creationTimestamp: null
name: my-secret
namespace: fluxcdtutorial
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
bash
omkar@omkar-Latitude-5520:~/Documents/ngnixdeployment/k8s_nginxapp_sot$

Appendix

- https://amitsharma13318.medium.com/streamlined-application-deployment-in-kubernetes-with-flux-and-gitops-7bf7ebd26224
- https://blog.sighup.io/sealed-secrets-in-gitops/
- https://fluxcd.io/flux/components/notification/
- https://yodamad.hashnode.dev/build-and-publish-an-helm-chart-deploy-it-with-flux – Helm charts
- https://blog.zelarsoft.com/using-gitops-with-flux-5723b7724f2f - Helm charts
- https://github.com/fluxcd/flux2-multi-tenancy/tree/main – Multi-tenancy