Gitops

Understanding the challenges of automating deployments

The Problem with Automated Deployments

DevOps team adopts cloud native approach

Management's confidence is beginning to wane

A GitOps approach promises to improve delivery

Exemplified by:

- Containers
- Kubernetes

Reliability issues:

- Failed deployments
- Environment ambiguity

Improving reliability:

- Adopt GitOps techniques
- Flux operator

Module Outline

Coming up:

- How Kubernetes gets things done
- Cloud native application delivery
- Configuration drift

Container Orchestration

Orchestration is required to manage containers at scale

Kubernetes is the de facto delivery platform for containerized applications

Automation to the Rescue

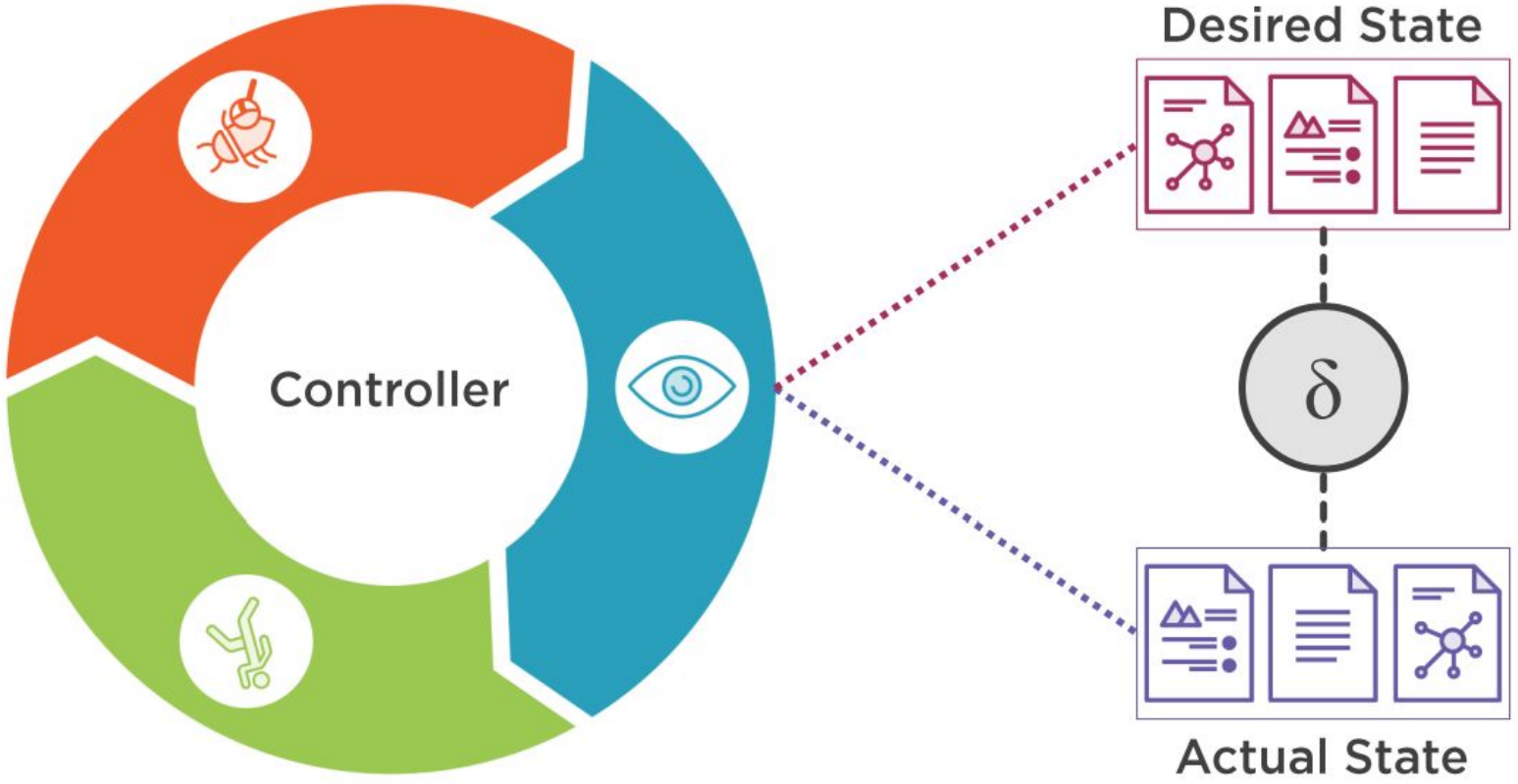
Rolling out and rolling back containerized workloads in the cluster

Enabling co-dependent workloads to discover each other

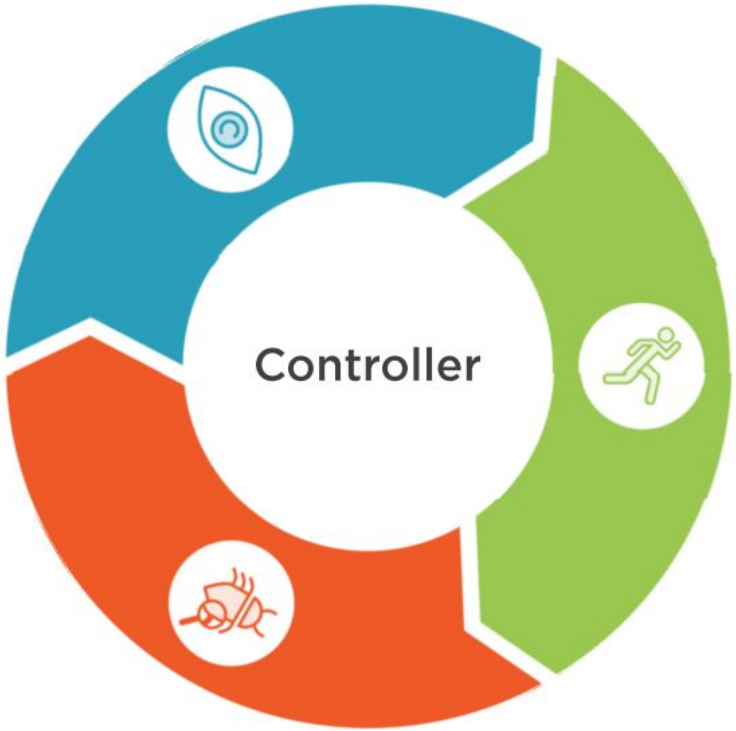
Re-starting deployed workloads when they enter a failed state

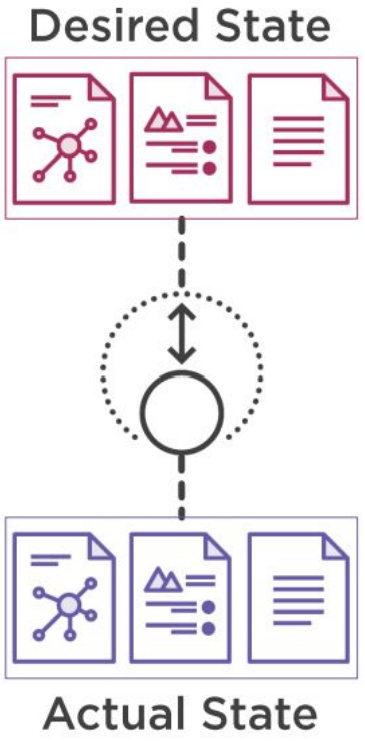
Automatic scaling of workloads in response to fluctuating demand

Desired State Reconciliation

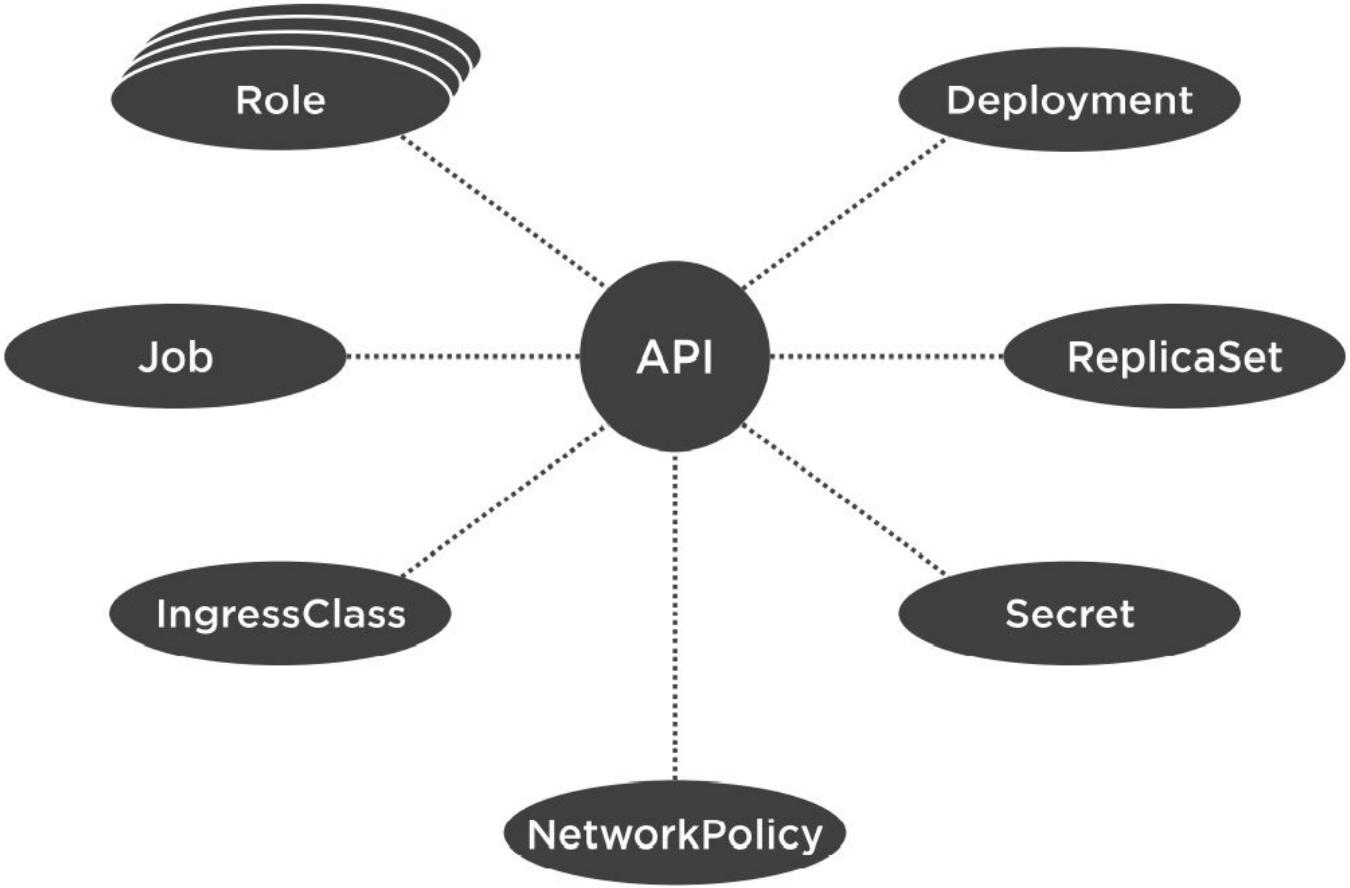


Desired State Reconciliation





Kubernetes API

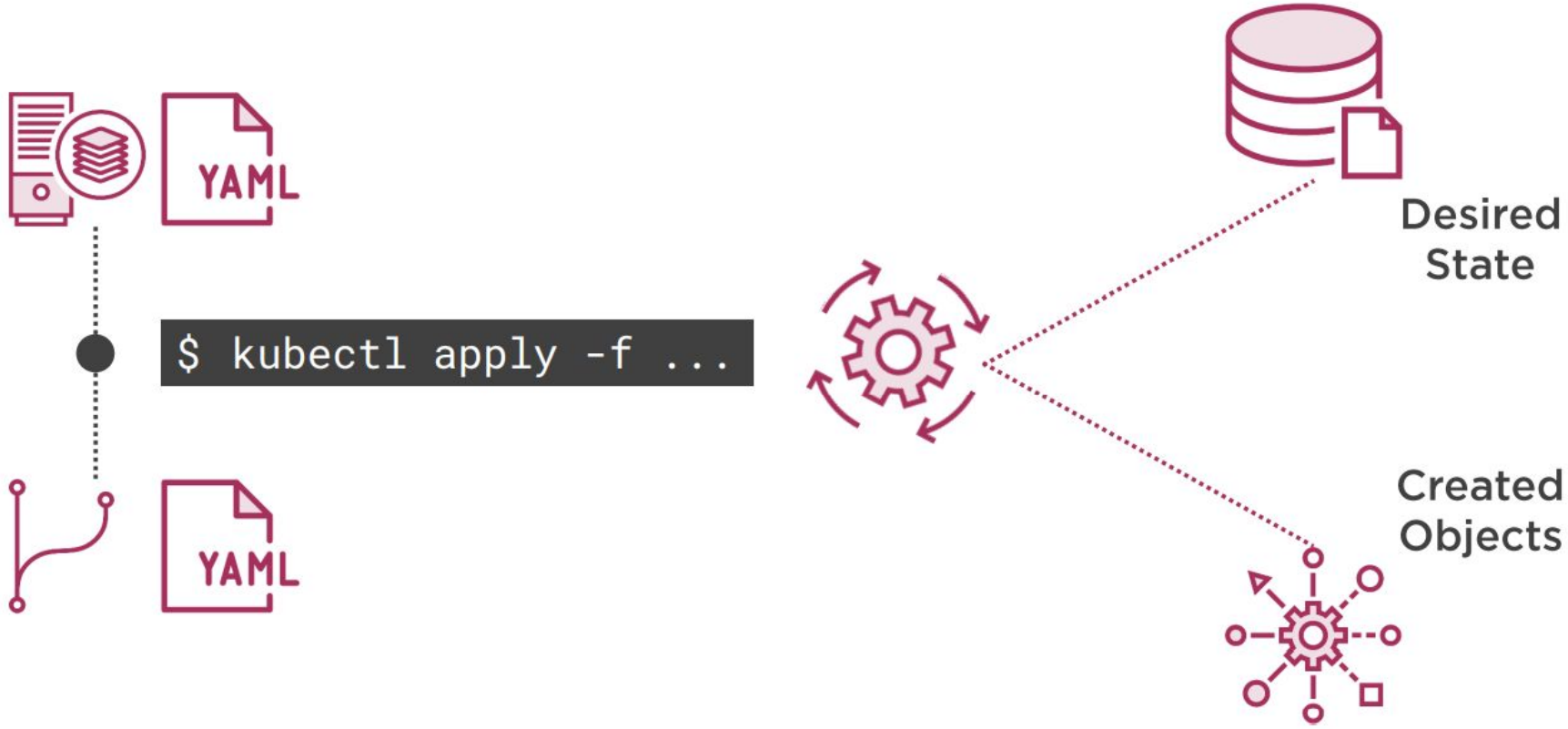


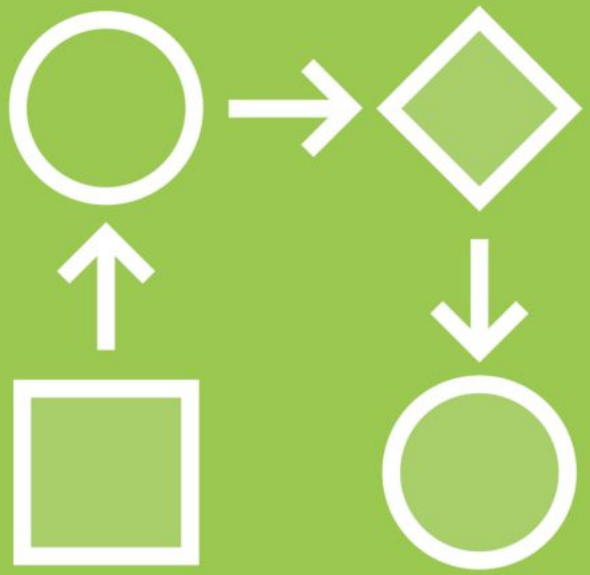
Declarative Configuration

pruner-job.yaml

```yaml
apiVersion: batch/v1
kind: Job
metadata:
  name: pruner
spec:
  template:
    metadata:
      name: pruner
    spec:
      containers:
      - name: pruner
        image: mycorp/pruner:v1.2
        command:
         - “prune"
         - "-c"
         - “/etc/prune.conf”
      restartPolicy: Never
```

Applying Configuration

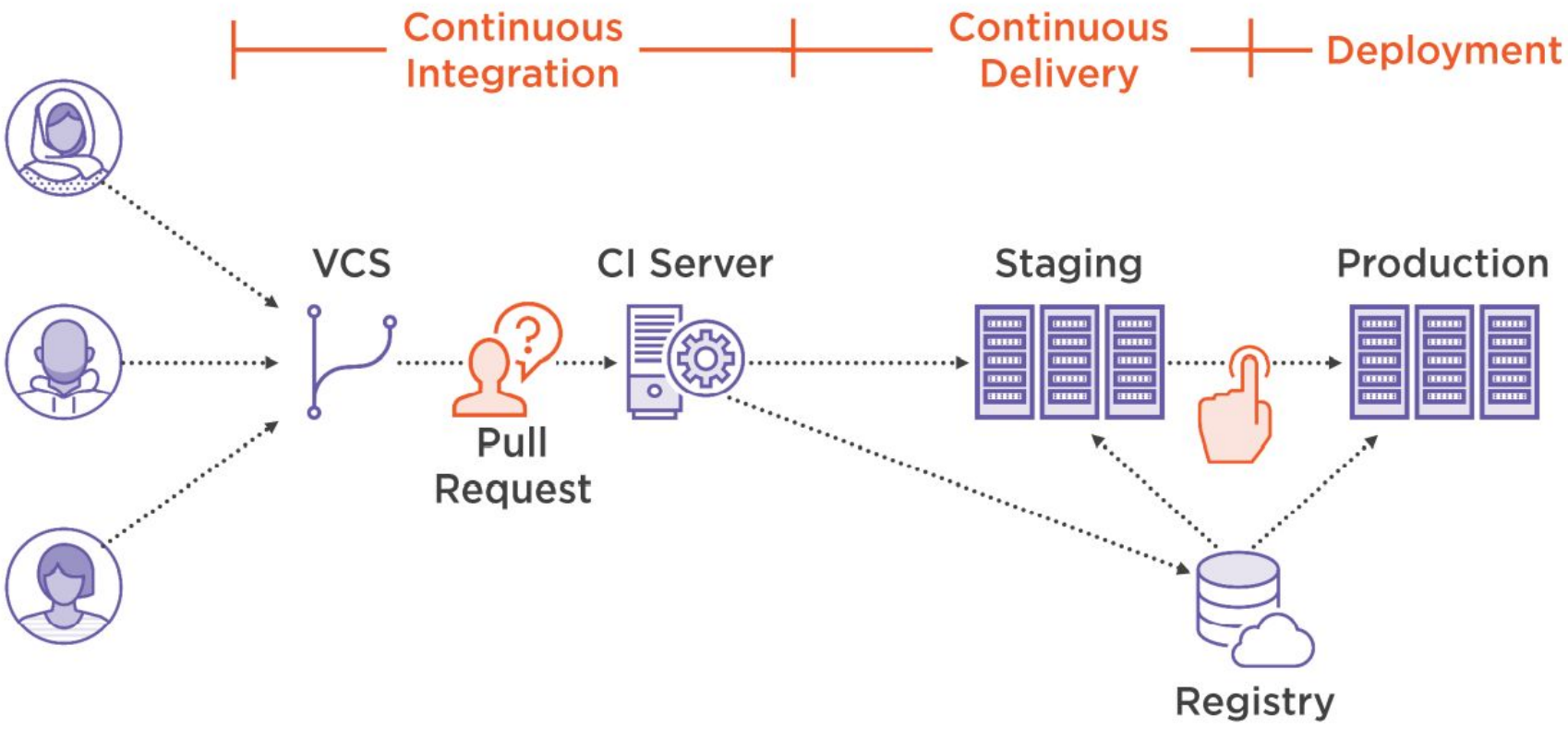




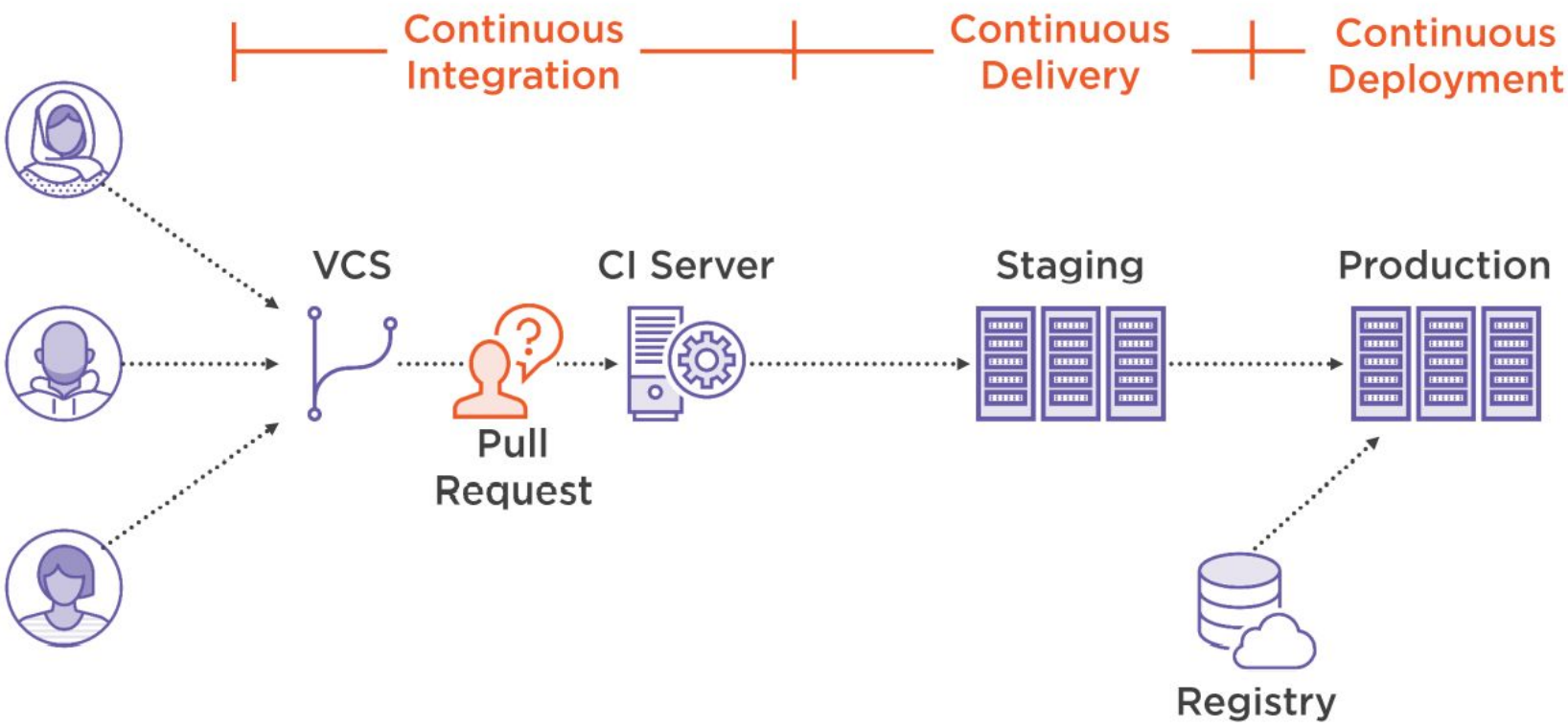
Cloud Native Workflows

Automated workflows or pipelines are used to increase the speed, frequency, and quality of application service deployments

Cloud Native Workflows

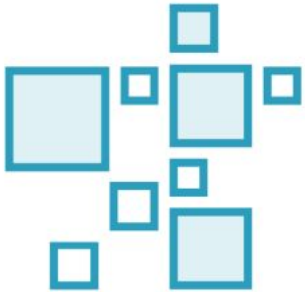


Cloud Native Workflows



Container Images

Container images encapsulate the deployable cloud native application service



Code + Dependencies

Contain application code and the package or library dependencies

Immutable

New versions of applications require a new container image

Container Registry

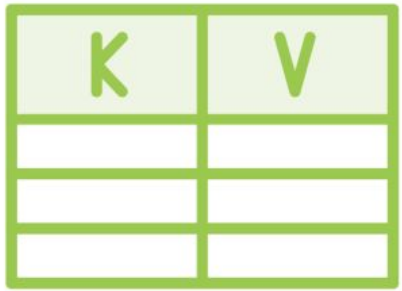
Often stored in remote registries and require pulling before running

Workload Configuration

Kubernetes config manifests are used to define the workload

References Image

Workload manifest references the desired container image



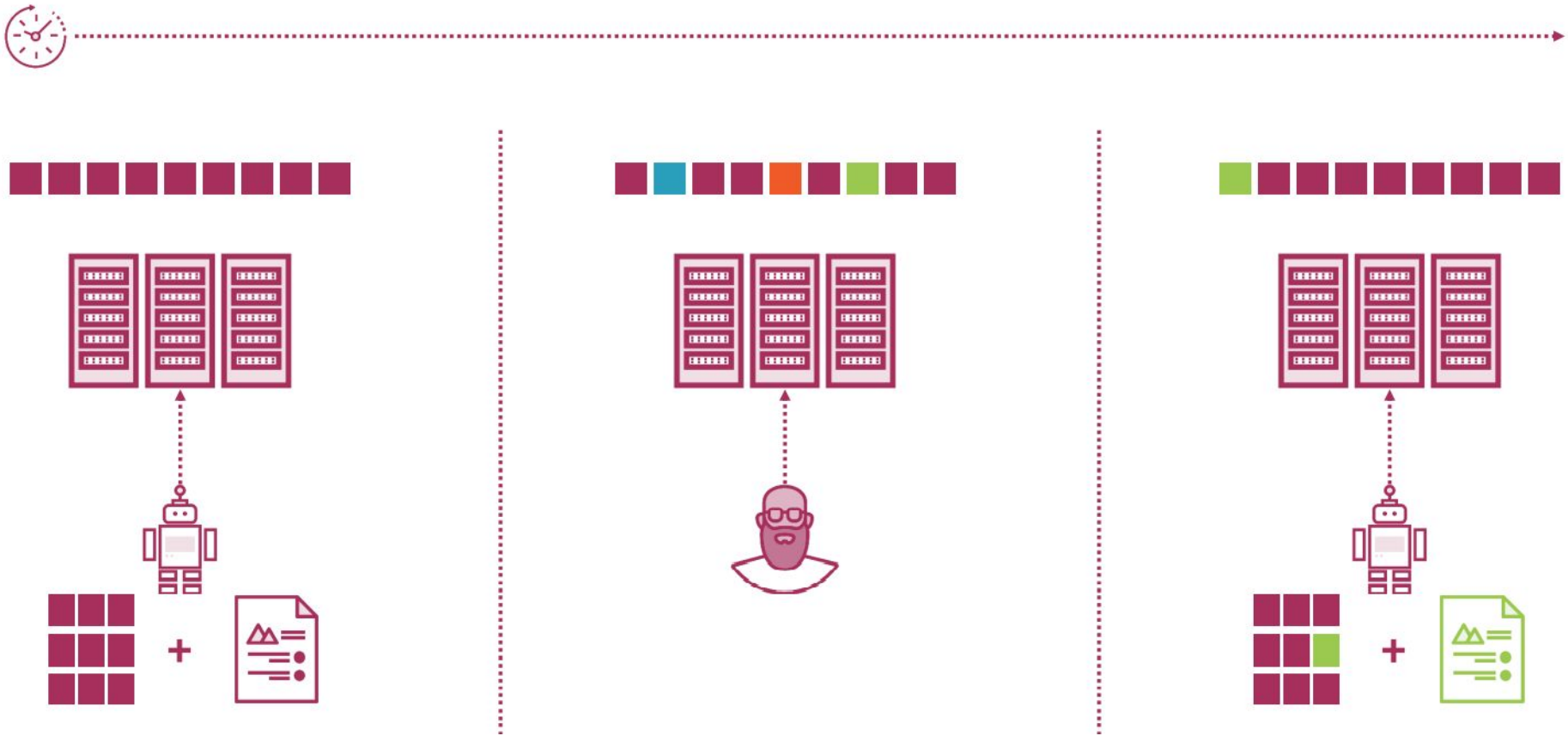
Workload Attributes

Defined in the YAML manifest(s) as configuration items

Mutable

Can be altered after deployment using imperative commands

Configuration Drift



Managing the integrity of configuration in Kubernetes is essential for success

Module Summary

What we covered:

- Application delivery using cloud native workflows
- Desired state, and desired state reconciliation
- Maintaining configuration integrity is a perennial problem

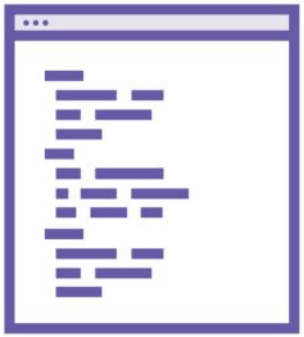
We need to know how to manage the configuration problem effectively!

Using the gitops approach for automating deployments

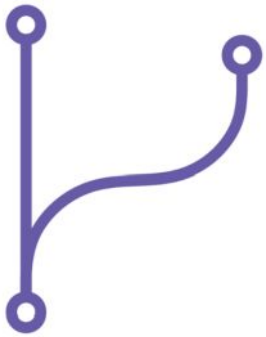
GitOps

GitOps is a workflow that relies on software automation to establish a desired state, which has been expressed declaratively in a version control system

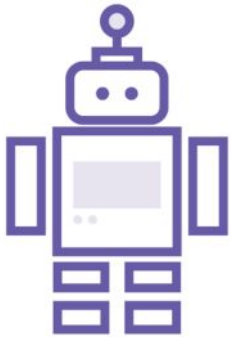
GitOps Principles



Described declaratively



Stored using version control



Automatically applied

Observing the state

Declarative Configuration

Allows for a definitive expression of the desired state of the system

Gives teams a common understanding of the intended system state

Provides a means for recovery in the event of a disaster scenario

GitOps and Version Control

Single Source of Truth

Recording changes to application code and configuration

Familiar Working Practice

Version control systems are already used by DevOps teams

Rollback Mechanism

Inherent properties of a VCS allow state reversion

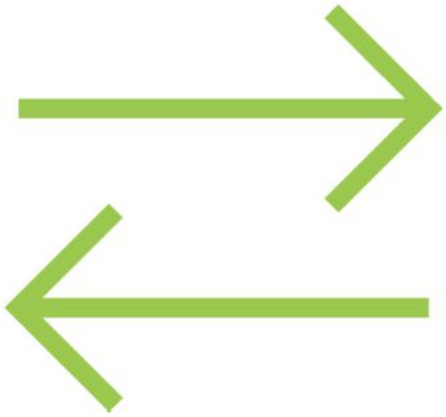
Transaction Log

A VCS records a change history along with the actors involved

Automated State Change

Peer Review

Conducted as part of a pull or merge request in the workflow



State Change Trigger

Results in a pull or push-based change of state

Observing State

Actual vs Desired State

External Command or Agent

Alerts or Synchronizes

State

- Subject to config drift
- Requires observation

Monitoring

- Inside or outside system

Action

- Inform
- Repair

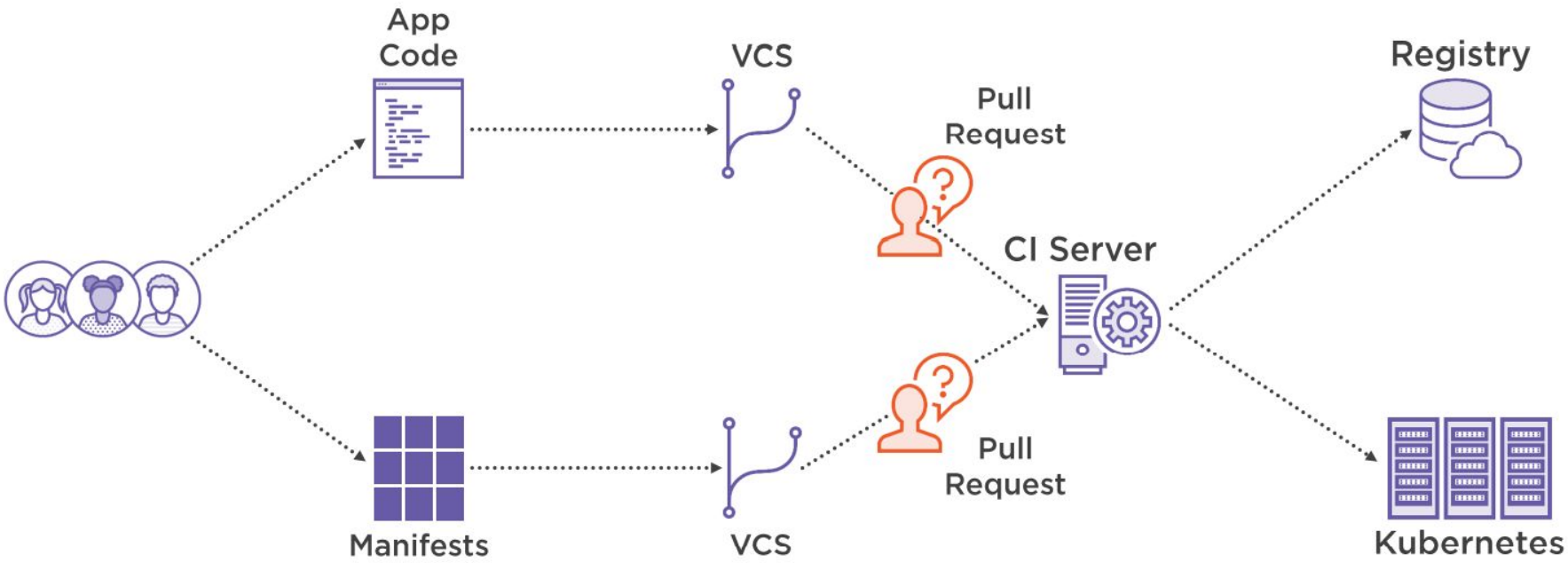
GitOps in the Wild

VCS does not need to be Git-based

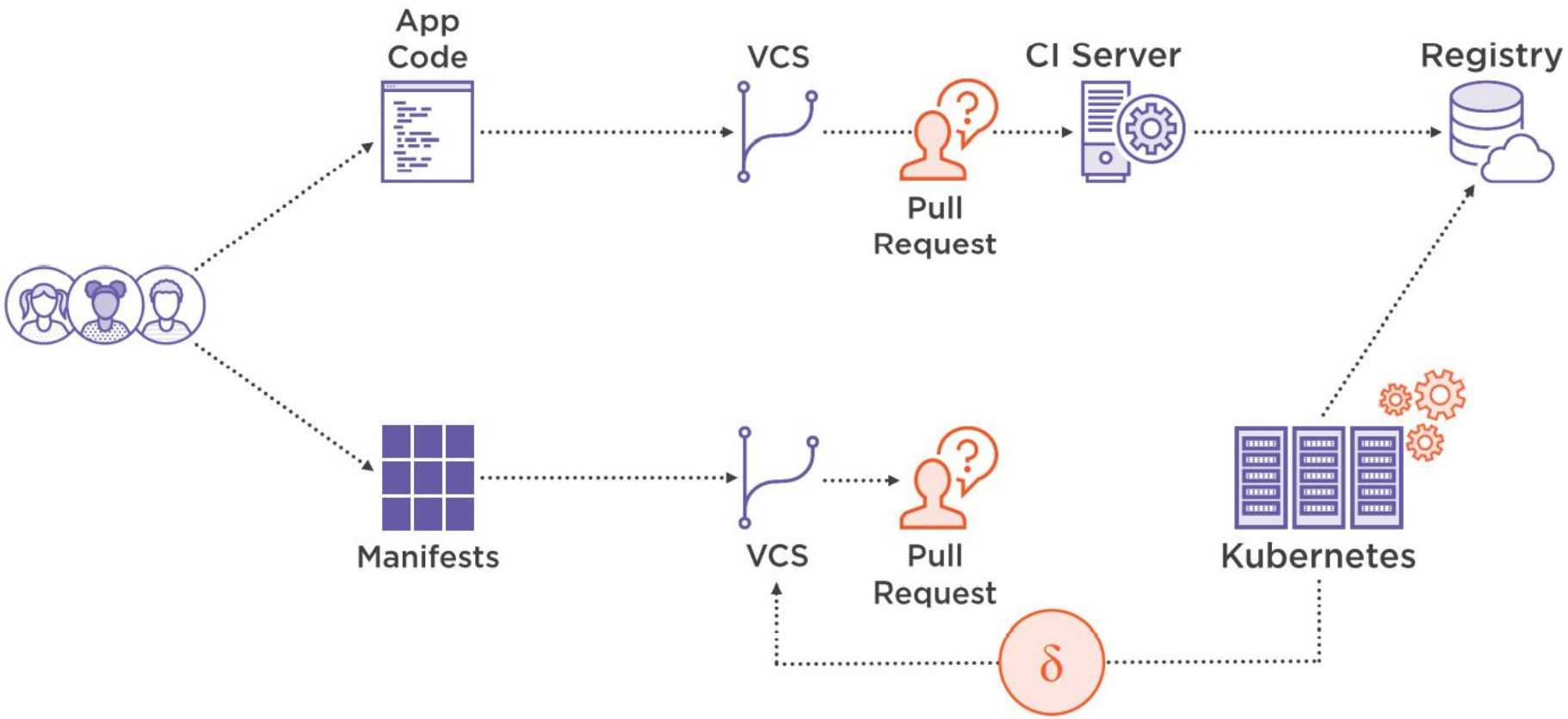
Problem domain not limited to Kubernetes

Appropriate for all layers of the stack

GitOps Push Workflow



GitOps Pull Workflow

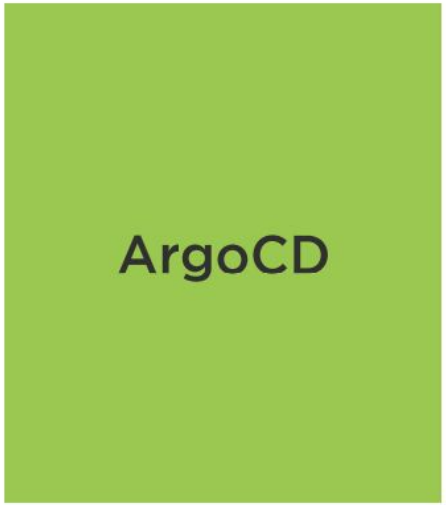


GitOps Deployment Strategy

Push	Pull
No visibility of state inside the cluster	Configured to view relevant API objects
Requires API access from outside cluster boundary	Accesses the API server from within cluster boundary
Variety of tools can fulfil push automation	Limited and bound by smaller set of available tools

GitOps Tools

https://jenkins-x.io/



https://git.io/JJ7yc

https://fluxcd.io/

Introducing Flux

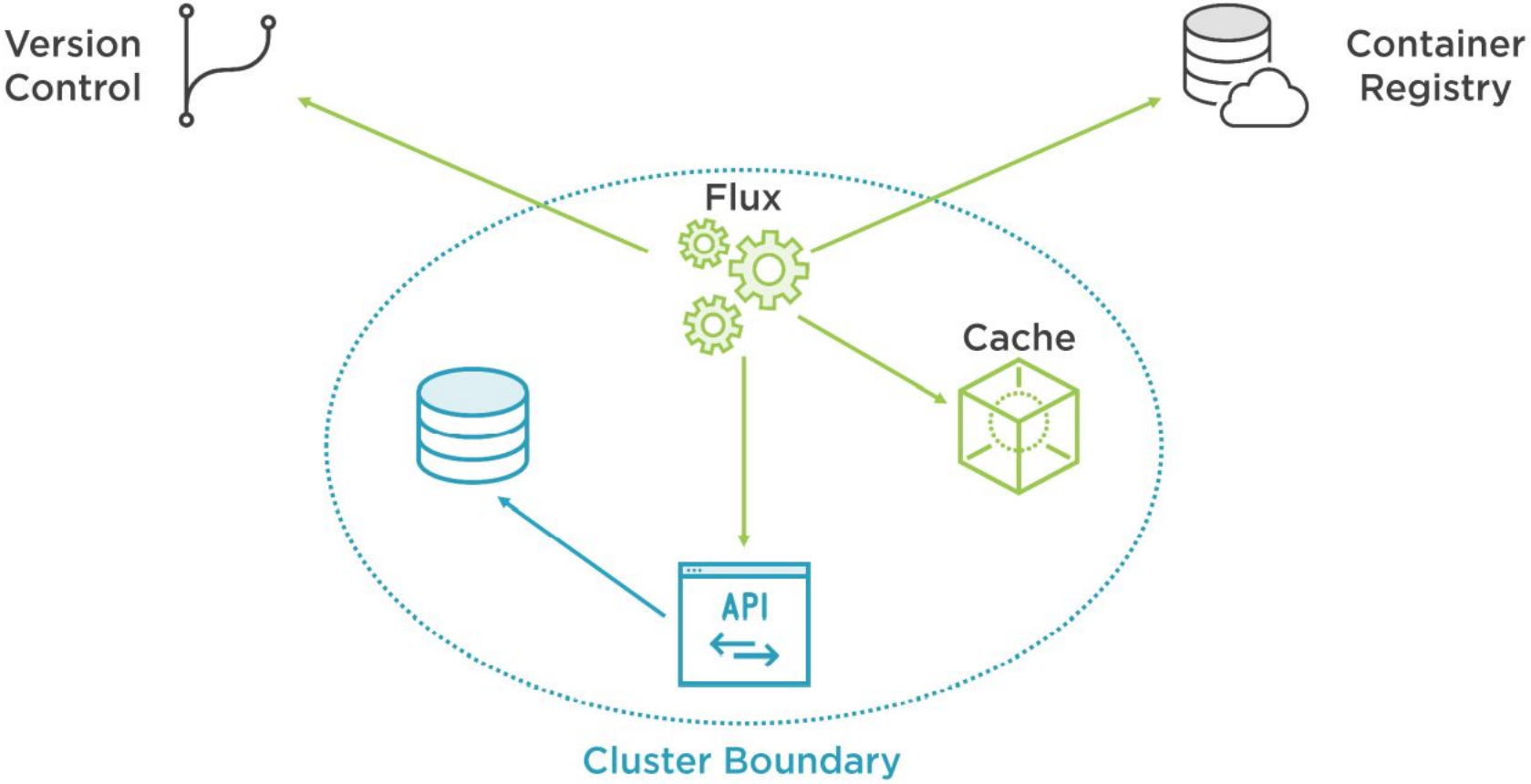
A Kubernetes operator that implements pull-based deployments for cloud-native applications

Scans container images used in deployments, and monitors container registry repositories for changes

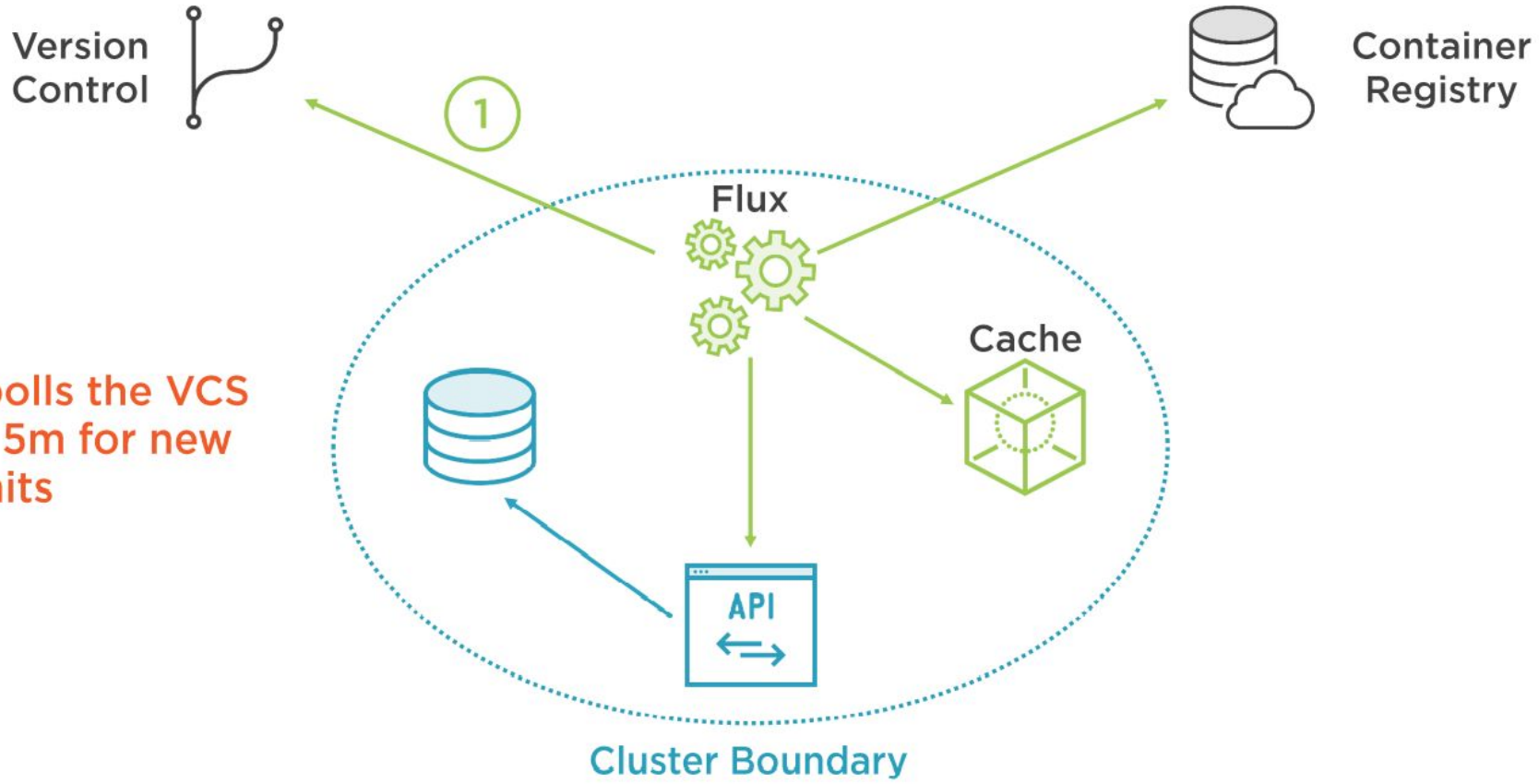
Caters for application deployment config defined in Helm charts, or generated with a command (e.g. Kustomize)

Can be used with other tools to support automated progressive deployments (e.g. canary releases)

GitOps Workflow with Flux

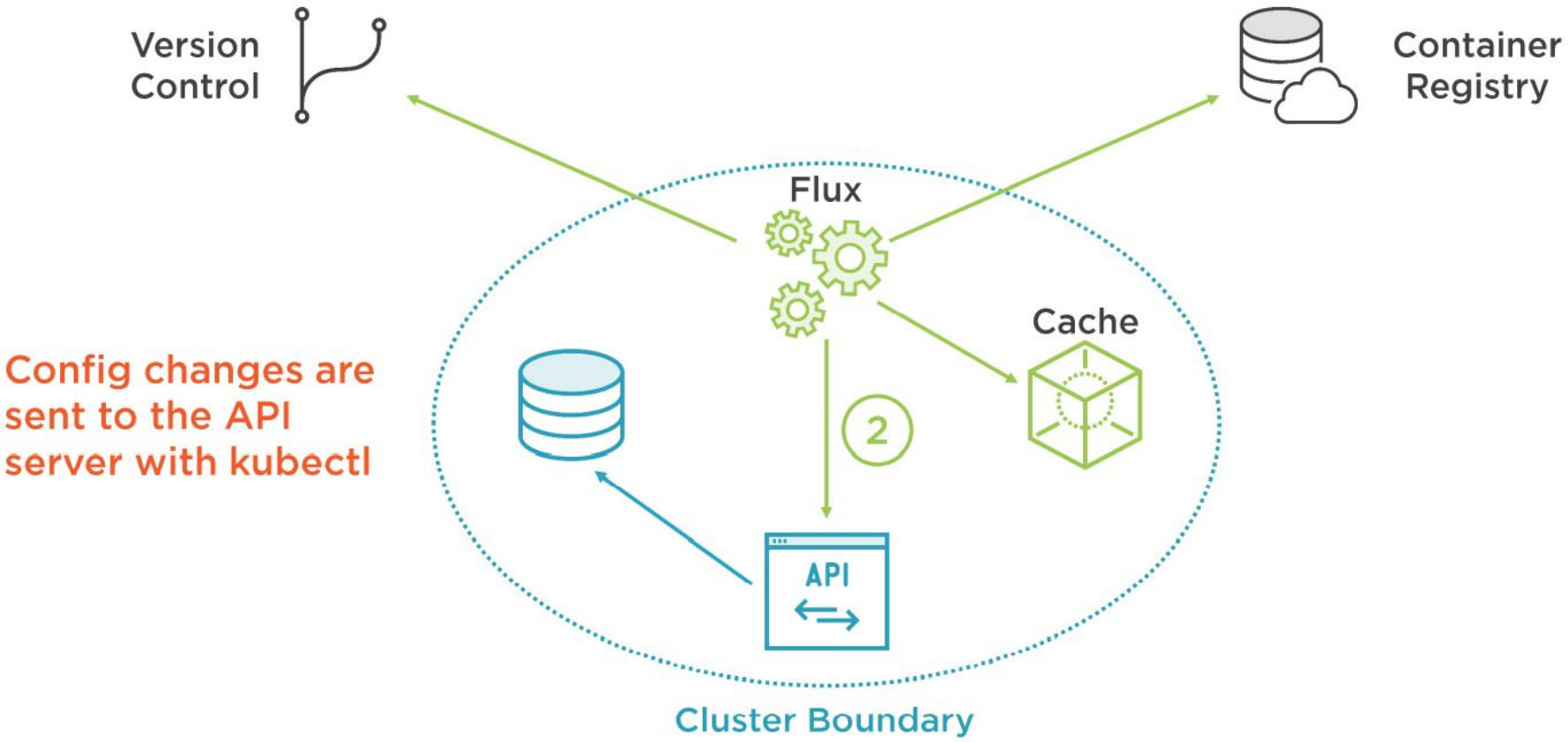


GitOps Workflow with Flux

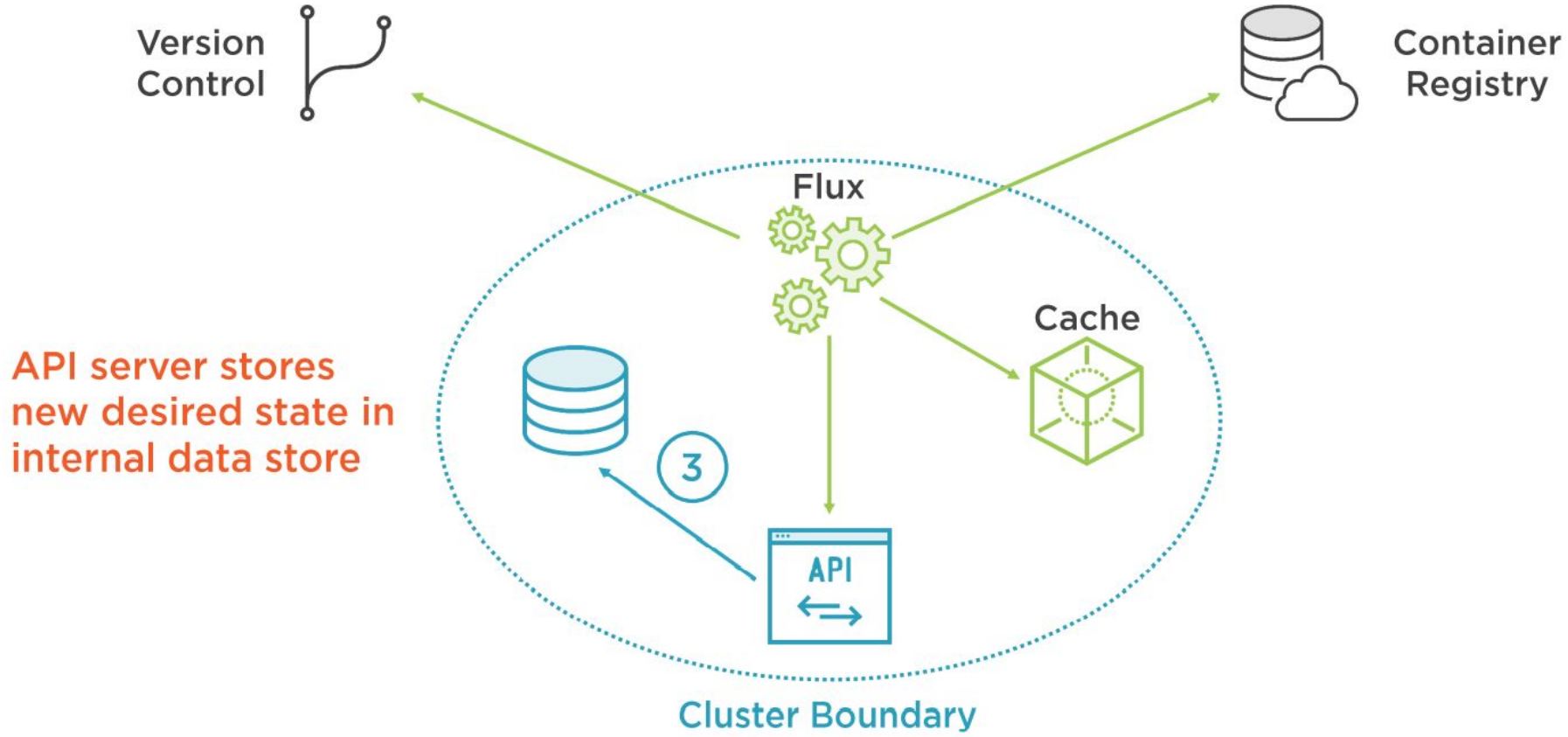


Flux polls the VCS every 5m for new commits

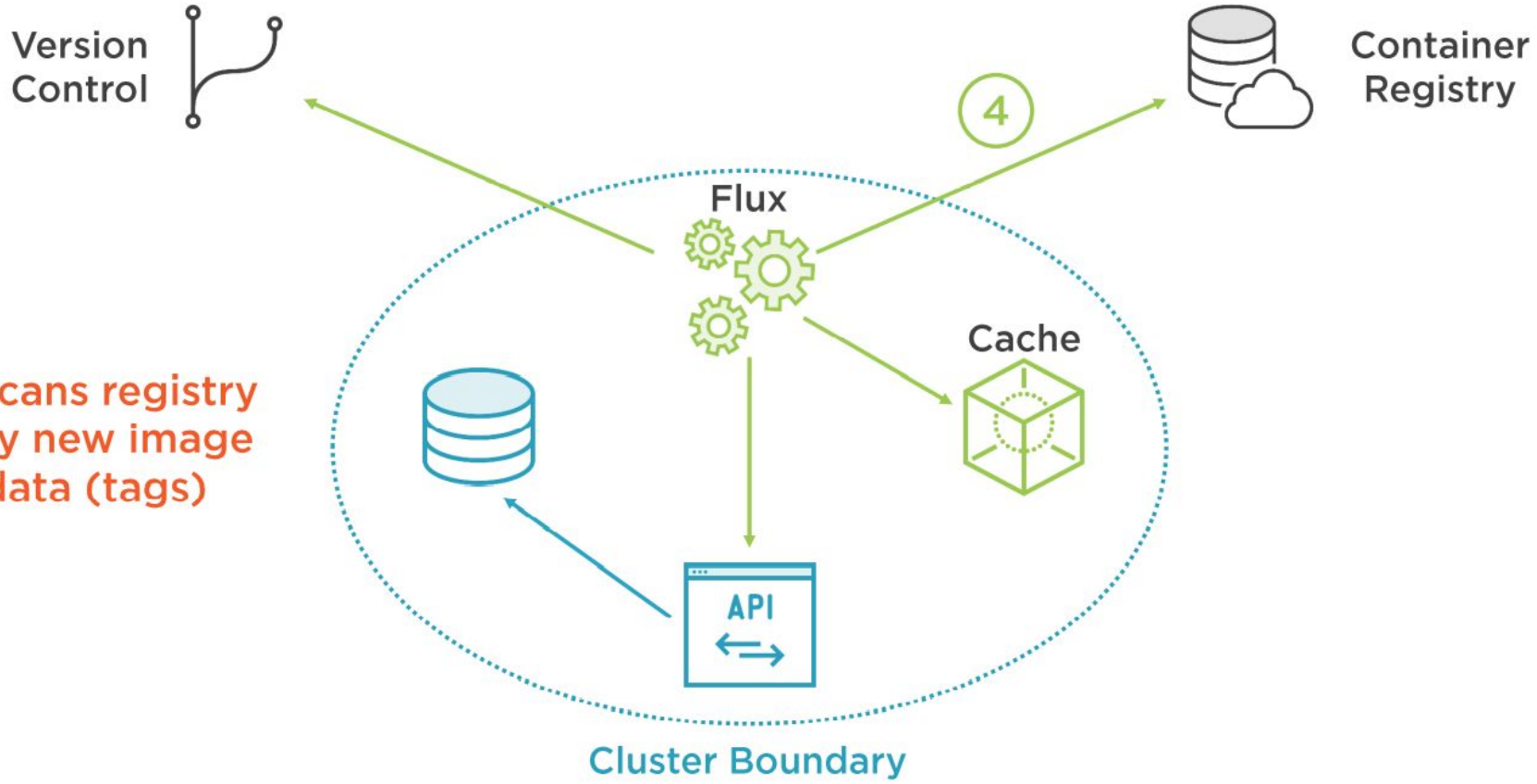
GitOps Workflow with Flux



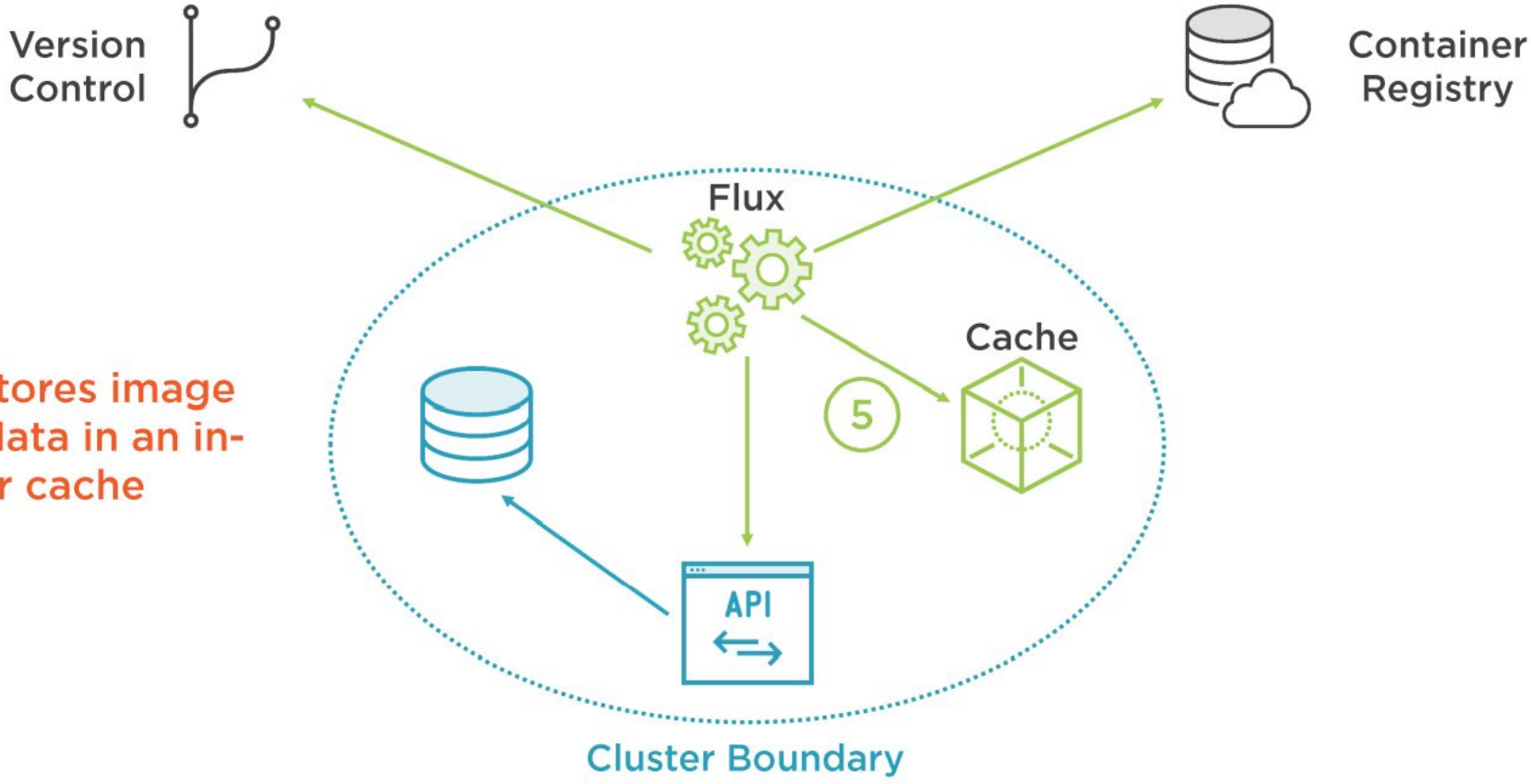
GitOps Workflow with Flux



GitOps Workflow with Flux



GitOps Workflow with Flux



Flux stores image metadata in an in-cluster cache

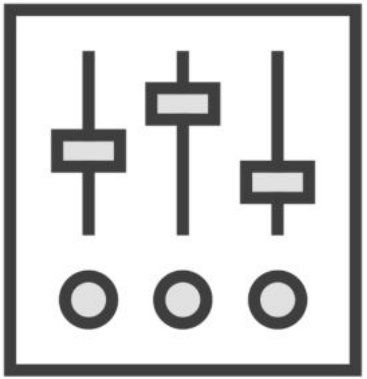
Fluxctl Command Line Interface

Operator Installation

`fluxctl install`

Identity Management

`fluxctl identity`



Flux Parameter Tuning

`fluxctl sync`

Deployment Information

`fluxctl list-*`

Module Summary

What we covered:

- The principles that define a GitOps approach
- Push and pull automation
- Flux operator and the Fluxctl CLI

Let's dig a bit deeper into the Flux operator!

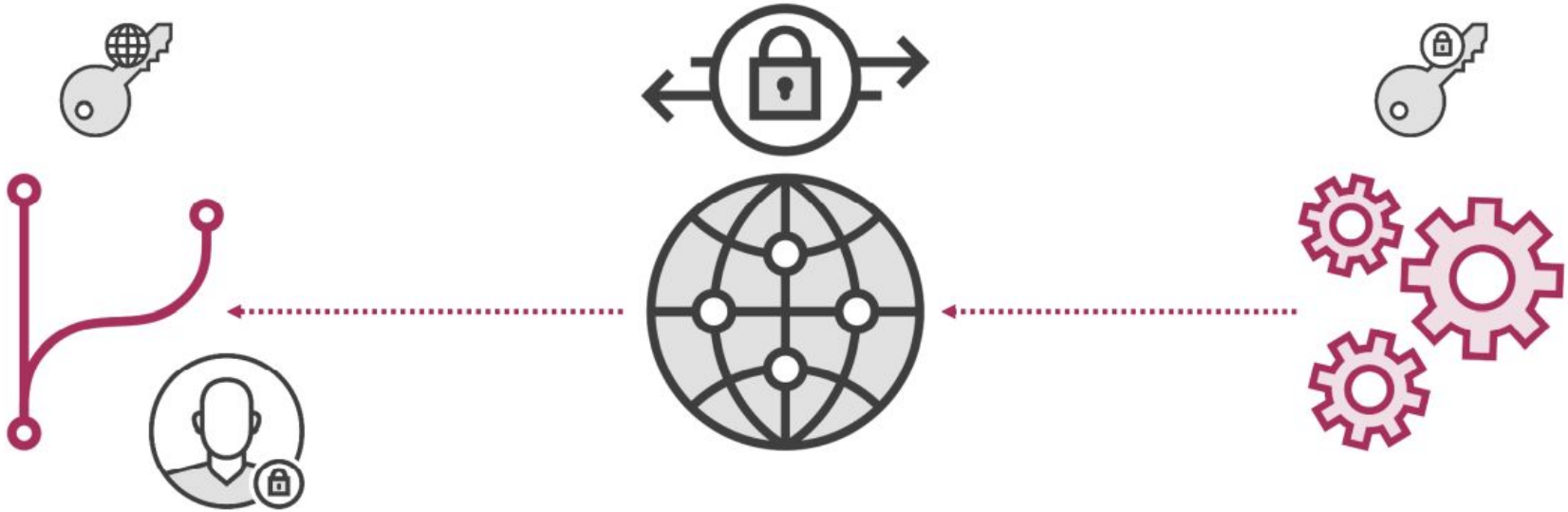
Configuring flux for automated deployments

Trusted Communication

Git VCS

Flux

Trusted Communication



SSH Identity

Flux allows you to ‘bring your own’ identity

Attribute	Flux Configuration Flag	Default Value
Secret Name	`--k8s-secret-name`	`flux-git-deploy`
Data Key	`--k8s-secret-data-key`	`identity`
Mount Path	`--k8s-secret-volume-mount-path`	`/etc/fluxd/ssh`

```
$ fluxctl identity
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABgQCu77eRrTXI0Uou2AmvFN1kIwPkUfs4IVLxxlKk+7Yk2nq
Z2L+LhTnIyycZeXl/Ib9M4aULrFqV0a3O17CE2e4J+2ZW5oLsTi/mHLvD4p7Zoyct8EGaBI0bheMWFIXuUX
CY62F8erxMOPOJSeZabc0uiX7/Qf0vYXgGZxQER+B4yyC56p2wwTO03Pjy4qFgE+v116mPfdIqmGHz5K5tk
wFIUIi6o1MajfXFq9V27pVwpNuL2vSkGenLrGBOvWhlKITxE1J6vzYbp32ITxBSrRPEhrTt1WDq13nuR04r
/6XFibeZvSDi9A/8LnlJViBV1/eg+kOLb05GYiz+Jjra77lRZItrkUarCcniCVly4Fp1tzK9zNeFP+1U985
jJ+5jUf24AKLEADHgBS7spy76iXxEx067seOi2bfQkEslk4AMErTkO0qZku4FaEl3PO0IhgQ8L2EJICNANO
k0NHYMjLy8zoAeK2Mq6QiE6okjqdwqcgKcbyawvmQQ7Unn1zkijY8= root@flux-7ddb575f68-cntck
```

Retrieving the Public Key

The ‘identity’ sub-command for ‘fluxctl’ can be used to retrieve the public key

Public key can also be found in the logs produced by Flux

```
# Use a flag to specify namespace for Flux
$ fluxctl --k8s-fwd-ns=flux identity

# Set environment variable to specify namespace for Flux
$ FLUX_FORWARD_NAMESPACE=flux fluxctl identity
```

Flux API Port-forwarding

The CLI expects to find the API by port-forwarding to port 3030

Alternative to CLI flag, use environment variable

Exposing the Flux API is considered insecure

Adding a Repository Deploy Key

Deploy keys / Add new

Title

flux-deploy-key

Key

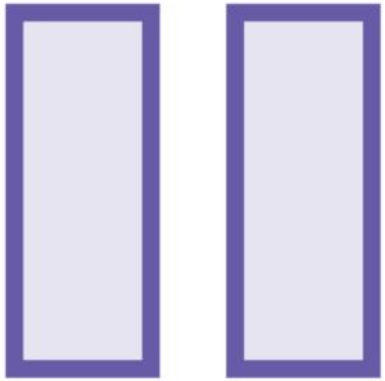
ssh-rsa
AAAAB3NzaC1yc2EAAAADAQABAAABgQCu77eRrTXI0Uou2AmvFN1kIwPkUfs4IVLxxIKk+7Yk2nqZ2L+LhTnIyycZeXI/Ib9M4aULrFqV0a3
O17CE2e4J+2ZW5oLsTi/mHLvD4p7Zoyct8EGaBI0bheMWFIXuUXCY62F8erxMOPOJSeZabc0uiX7
/Qf0vYXgGZxQER+B4yyC56p2wwTO03Pjy4qFgE+v116mPfdIqmGHz5K5tkwFIUIi6o1MajfXFq9V27pVwpNuL2vSkGenLrGBOvWhIKITxE1
J6vzYbp32ITxBSrRPEhrTt1WDq13nuR04r/6XFibeZvSDi9A/8LnIJViBV1
/eg+kOLb05GYiz+Jjra77IRZItrkUarCcniCVIy4Fp1tzK9zNeFP+1U985jJ+5jUf24AKLEADHgBS7spy76iXxEx067seOi2bfQkEsIk4AMErTkO0q
Zku4FaEI3PO0IhgQ8L2EJICNANOk0NHYMjLy8zoAeK2Mq6QiE6okjqdwqcgKcbyawvmQQ7Unn1zkijY8= root@flux-7ddb575f68-cntck

Allow write access

Can this key be used to **push** to this repository? Deploy keys always have pull access.

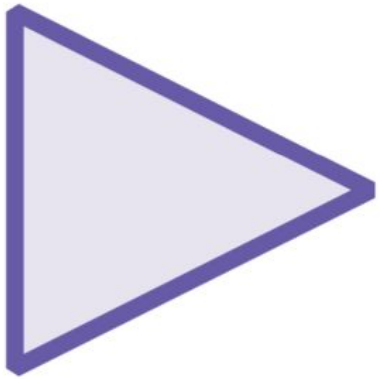
Add key

Flux Syncing Options



Selective syncing

Specific resources excluded from sync



Syncing on demand

Manual sync initiated with fluxctl

Continuous syncing

Sync performed at periodic interval

Ignore Annotation

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: envoy
  name: envoy
  annotations:
    fluxcd.io/ignore: "true"
spec:
  replicas: 1
.
.
.
.
```

```
$ fluxctl --k8s-fwd-ns=flux sync
Synchronizing with ssh://git@github.com/nbrownuk/gitops-nginxhello.git
Revision of master to apply is 112a7b5
Waiting for 112a7b5 to be applied ...
Done.
```

Syncing on Demand

Force an immediate sync using the Fluxctl CLI

Continuous Syncing

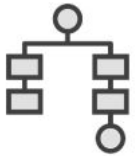
Git poll interval

Every 5m, but can be reduced to retrieve new commits more often

Sync interval

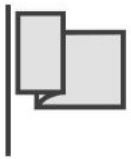
Every 5m, but syncs are applied as soon as new commits are retrieved

Repository Layout



No required structure, Flux recursively descends into sub-directories looking for YAML files

Directories that appear to be templated Helm chart content are ignored by Flux

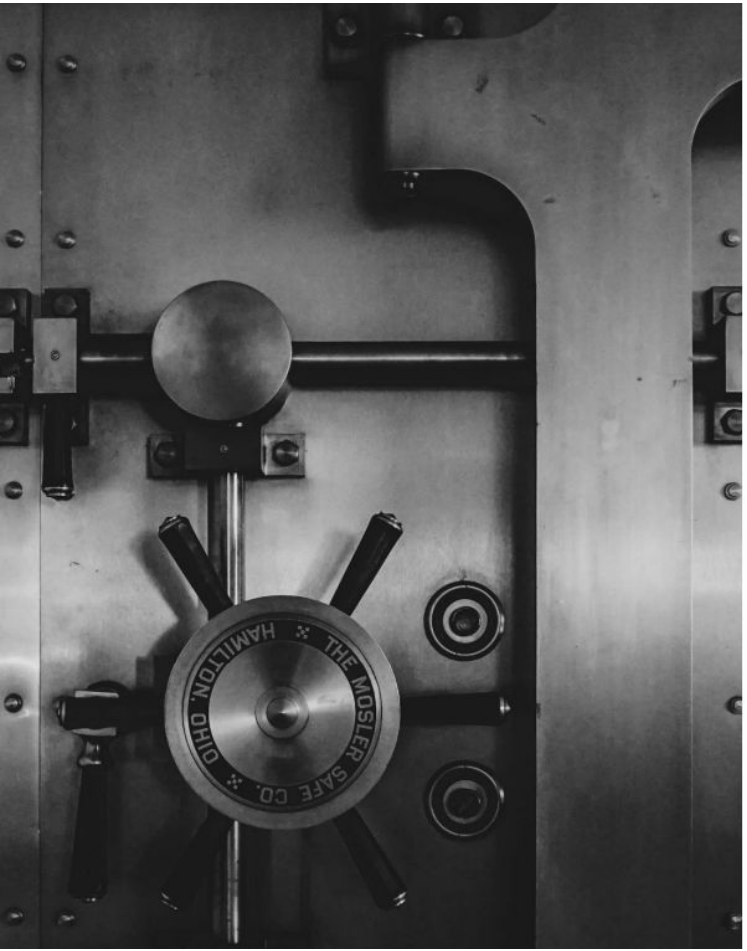


Explicit paths for YAML content can be set using the '--git-path' flag on Flux start up

```
$ fluxctl list-workloads
WORKLOAD                         CONTAINER   IMAGE                         RELEASE  POLICY
default:deployment/nginxhello    nginxhello  nbrown/nginxhello:1.19.0      ready
```

Listing Workloads Managed by Flux

Fluxctl CLI provides details of the workloads managed by the Flux instance



Container registry access may require authorization

Container registry access provided by:

- Image pull secret associated with a service account or workload
- Specific public cloud provider mechanism
- Credentials stored in Docker config file, mounted into Flux

Listing Available Workload Images

```
$ fluxctl list-images
WORKLOAD                          CONTAINER   IMAGE                CREATED
default:deployment/nginxhello     nginxhello  nbrown/nginxhello
                                              |   1.19             18 Aug 20 12:01 UTC
                                              |   1.19.2           18 Aug 20 12:01 UTC
                                              |   latest           18 Aug 20 12:01 UTC
                                              |   mainline         18 Aug 20 12:01 UTC
                                              |   1.19.1           18 Aug 20 11:46 UTC
                                              '-> 1.19.0           01 Jun 20 09:14 UTC
                                                  1.18             04 May 20 13:22 UTC
                                                  1.18.0           04 May 20 13:22 UTC
                                                  stable           04 May 20 13:22 UTC
                                                  1.17             04 May 20 13:17 UTC
```

Flux will apply the most recent image it finds

Working with Image Tag Filters

SemVer

Filter based on semantic versioning

```
semver:~1.0.5
```

Regexp

Filter based on regular expressions

```
regex:^[a-f0-9]{7}$
```

Glob

Filter based on globbing syntax

```
glob:staging-*
```

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: envoy
  namespace: edge-proxy
  annotations:
    fluxcd.io/tag.envoy: semver:~1.14
    fluxcd.io/tag.initconfig: regexp:^[a-f0-9]{7}$
.
.
```

Applying Image Tag Filters

Resource annotation used to specify the image tag filter

Unique annotation required for each container described in pod spec

```
$ fluxctl policy --workload=edge-proxy:daemonset/envoy \
    --tag='initconfig=regexp:^([a-zA-Z]+)$'
```

Defining Tag Filters with Fluxctl

Filters can be applied using the Fluxctl CLI, specifying the target workload

Use single or multiple '--tag' flags to define tag filtering

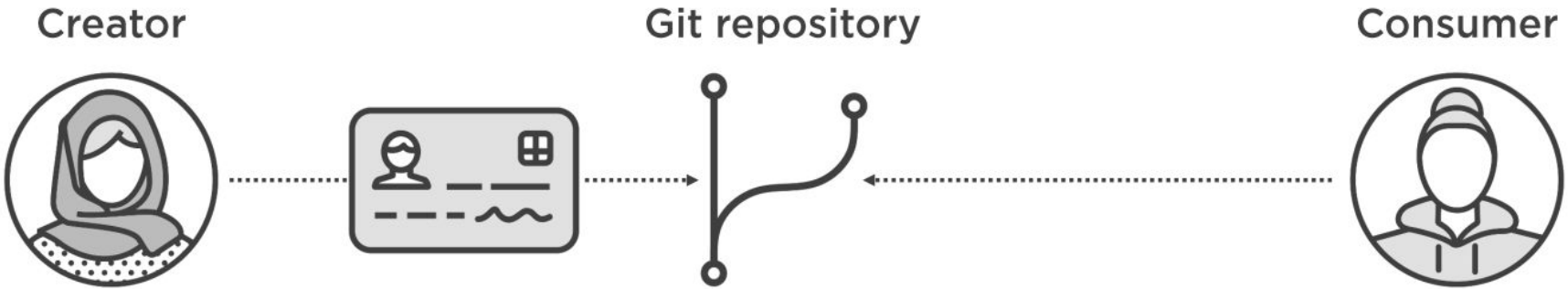
```
# Turning Automation On
$ fluxctl automate --workload=edge-proxy:daemonset/envoy


# Turning Automation Off
$ fluxctl deautomate --workload=edge-proxy:daemonset/envoy
```

Controlling Automated Deployments

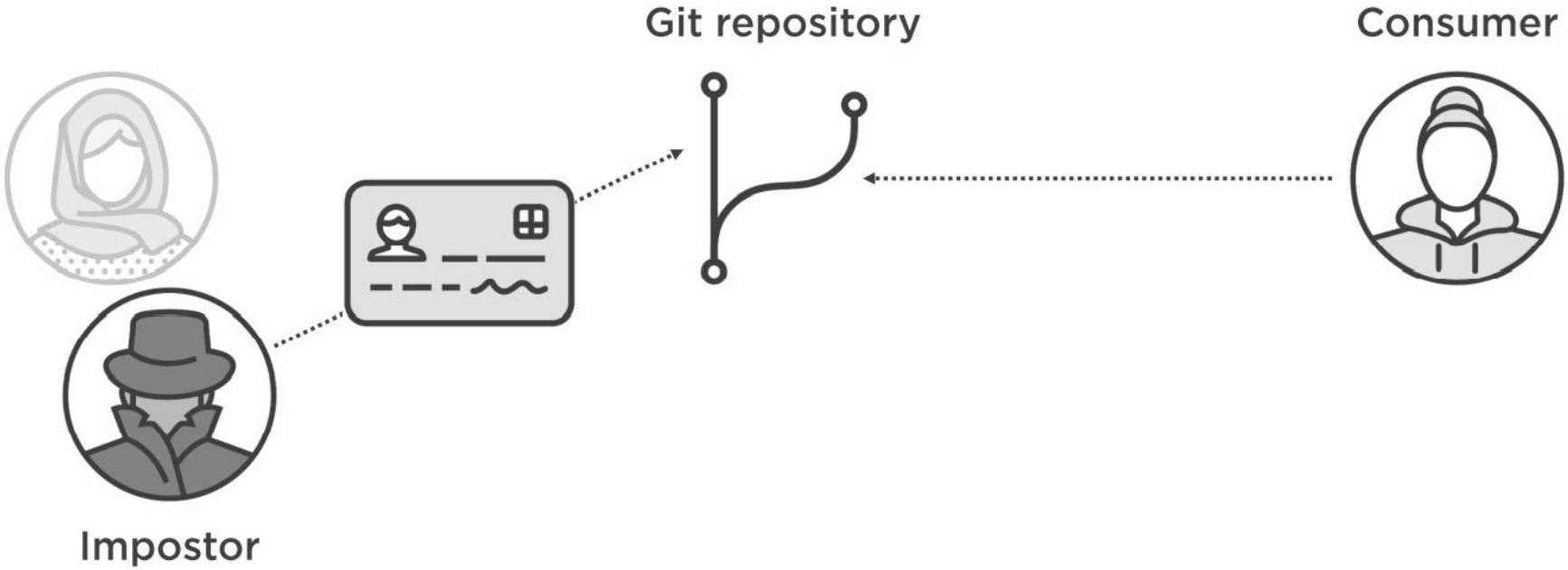
Fluxctl can be used to turn automated deployments on and off

Flux writes fluxcd.io/automated: “true” (or false) annotation to manifest

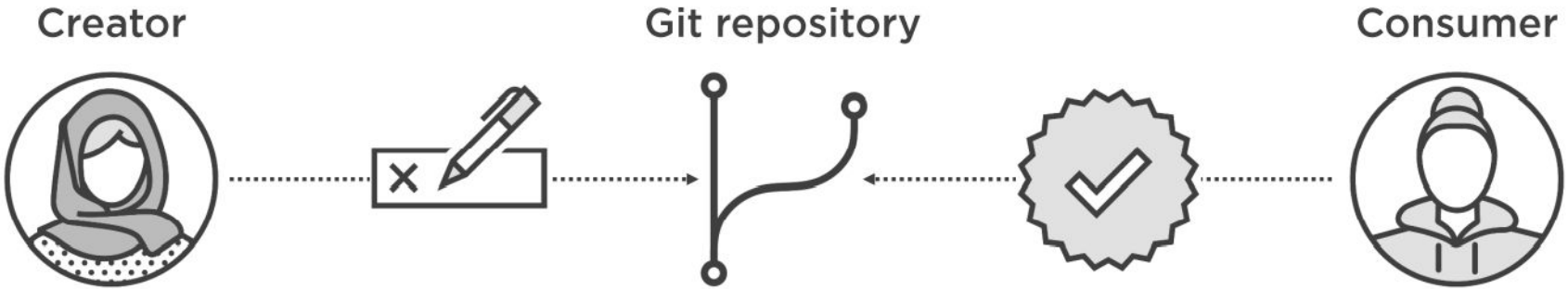
Git Commit Authenticity



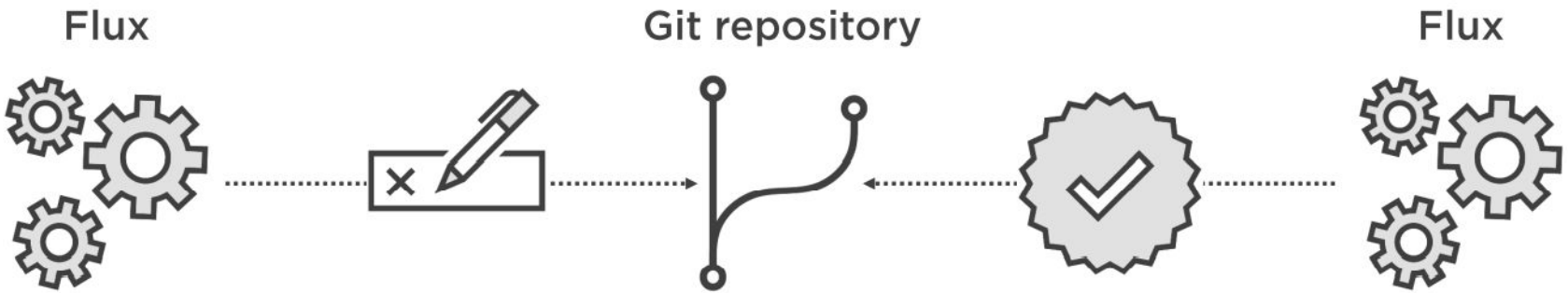
Git Commit Authenticity



Git Commit Authenticity



Git Commit Authenticity



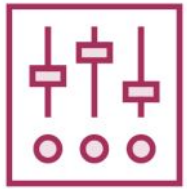
GPG Signing Key for Commits

Flux arguments for GPG:

- Mount point
  `--git-gpg-key-import`
- Key ID
  `--git-signing-key`

Private GPG key embodied in a secret

Secret mounted into the Flux container



Flux arguments used to define location and key ID

Module Summary

What we covered:

- Identities and trust relationship
- Syncing options for desired state
- Automating new application releases
- Commit signing