Handling Application Updates with Image Automation

Nigel Brown

@n_brownuk www.windsock.io

Module Outline

Coming up:

- Application updates triggered by CI/CD
- Configuring Flux to scan for images tags
- Using image policy to select app version
- Automating updates to desired state

How Are Applications Updated?

Configuration update performed by automation software used in CI/CD pipeline

In-built image automation capabilities of GitOps agent updates the configuration

Demo

Updating an Application Deployment with Flux

- Change app version in cloned git repo
- Push changes to the remote repo
- Observe Flux updating the app in-cluster
- Confirm the update was successful

Can Flux be used to update the desired state configuration for new application images?

Flux's Image Automation

Flux uses the image reflector and image automation controllers to handle automatic application updates to Kubernetes clusters.

ImageRepository

Encodes the location of the container image repo for an application.

ImagePolicy

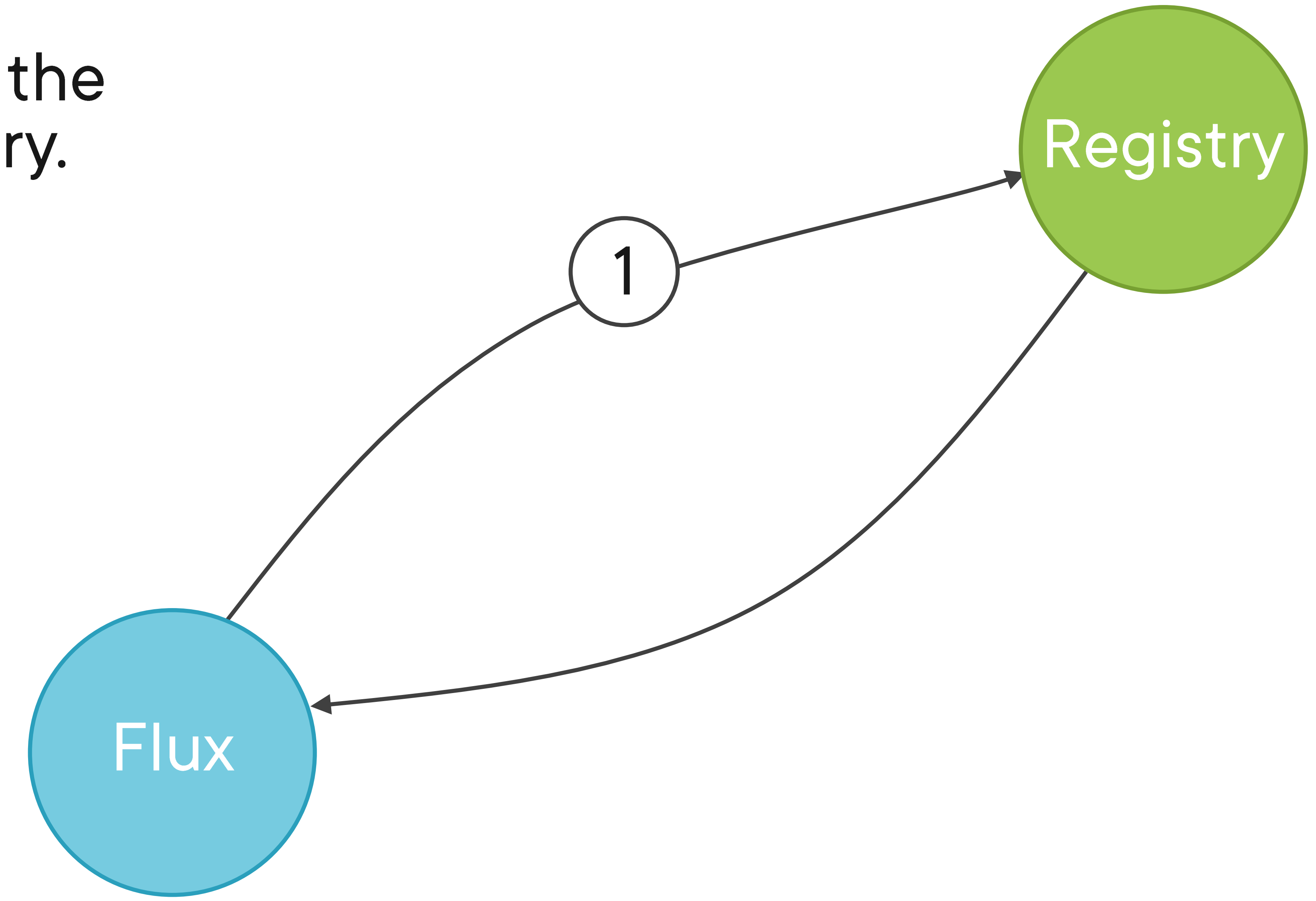
Defines how Flux selects the most recent image to use for an application.

ImageUpdateAutomation

Specifies how Flux updates the config in a source for an application.

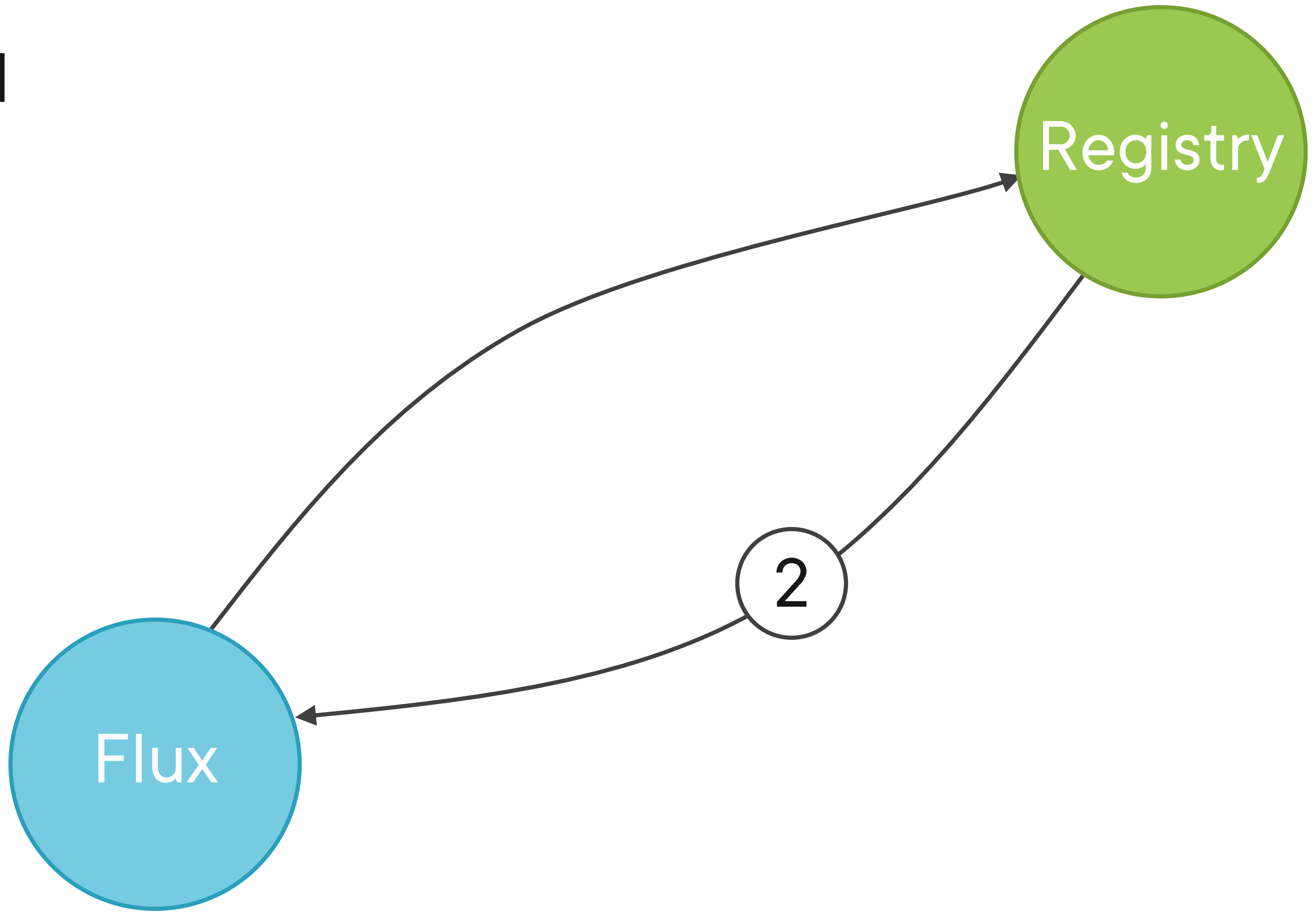
Querying the Image Repository

1 Image reflector controller queries the repo in the container image registry.



Fetching the Image Tags

2 The set of image tags are retrieved and stored in a local database.



ImageRepository API

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageRepository
metadata:
  name: nginxhello
  namespace: default
spec:
  image: docker.io/nbrown/nginxhello
  interval: 5m0s
```

Watch out for container registry rate limits!

Container Registry Authentication

Public repositories don't require authentication.

Self-hosted registries and private repositories do.

Configuring Authentication

Flux provides two different authentication methods.

TLS Certificate

```
---
apiVersion:
image.toolkit.fluxcd.io/v1beta1
kind: ImageRepository
metadata:
  name: nginxhello
  namespace: default
spec:
<snip>

  certSecretRef: nginxhello-auth

<snip>
```

Registry Credentials

```
---
apiVersion:
image.toolkit.fluxcd.io/v1beta1
kind: ImageRepository
metadata:
  name: nginxhello
  namespace: default
spec:
<snip>

  secretRef: nginxhello-auth

<snip>
```

```
$ flux create image repository nginxhello \
    --image=docker.io/nbrown/nginxhello \
    --interval=5m \
    --namespace=default \
    --export
```

Creating ImageRepository Resources

Flux CLI enables the creation of ImageRepository resources on our behalf

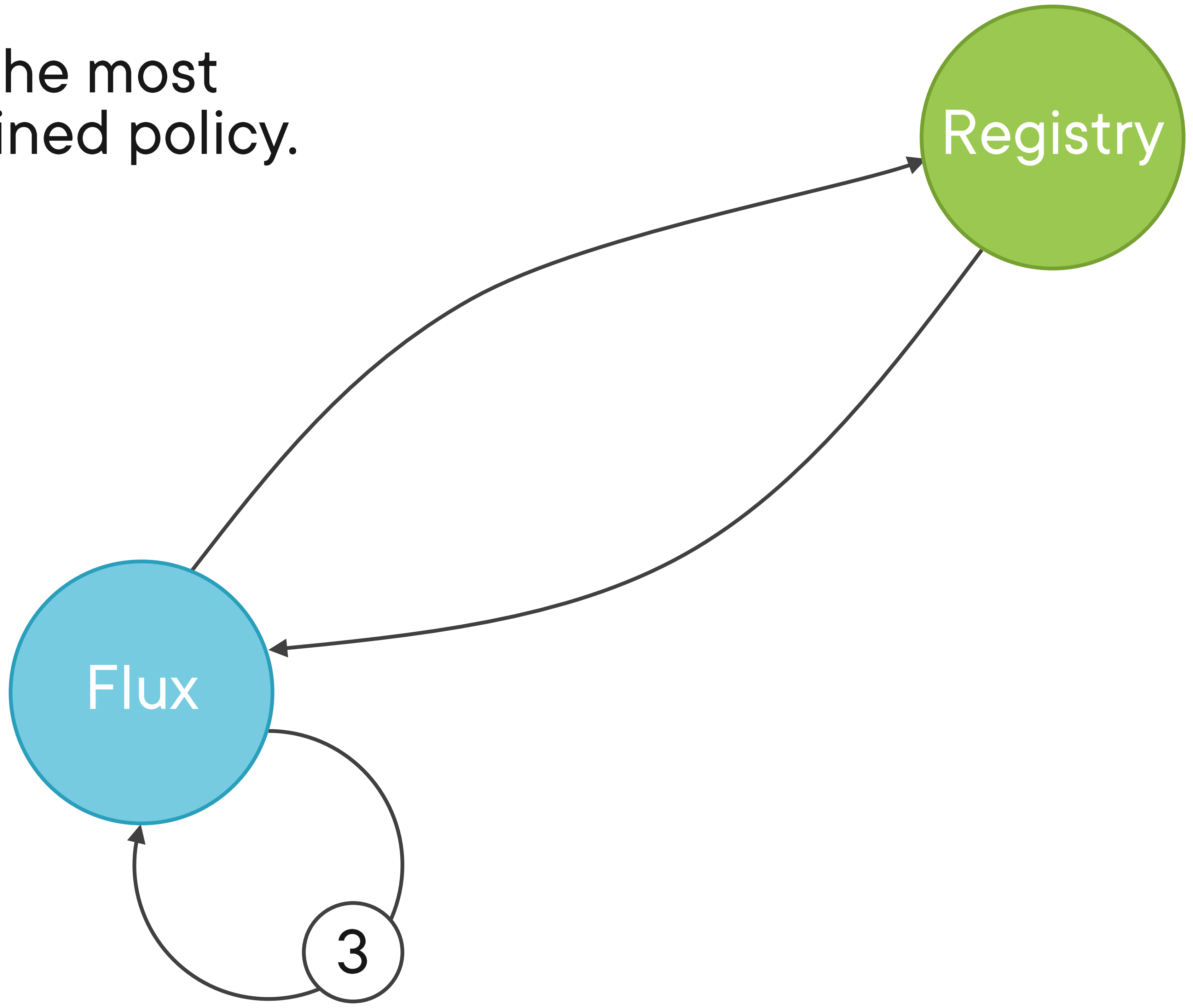
Demo

Configuring Flux to Scan a Container Image Repository

- Generate an ImageRepository resource
- Stage, commit and push the changes to GitHub repo
- Confirm that revised configuration has been applied to cluster

Selecting the Latest Image Tag

3 Image reflector controller selects the most recent image tag according to defined policy.



ImagePolicy API

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy
metadata:
  name: nginxhello
  namespace: default
spec:
  imageRepositoryRef:
    name: nginxhello
  policy:

    <snip>
```

Image Tag Selection Policy

The policy for selecting an image tag from a repository can be defined using one of three different techniques.

[a, b, c] [1, 2, 3] [1.20.2]

Alphabetical

Tags are sorted in alphabetical order before last tag is selected.

Numerical

Tags are sorted in numerical order before last tag is selected.

SemVer

Tags sorted according to SemVer constraints before highest selected.

Alphabetical Tag Selection

```
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy

<snip>

spec:
  policy:
    filterTags:
      pattern: ‘^REL\.(?P<ts>.*)Z.*$’
      extract: ‘$ts’
    alphabetical:
      order: asc
```

REL.2022-05-23T18-45-11Z.fix

2022-05-23T18-45-11

Numerical Tag Selection

```
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy

<snip>

spec:
  policy:
    filterTags:
      pattern: '^circle-ci-[0-9]+-(?P<ts>[1-9][0-9]*)'
      extract: '$ts'
    numerical:
      order: asc
```

```
circle-ci-782-1654004226
```

```
1654004226
```

Useful Resources for Image Policy Patterns

(.*)

Regex101 - https://www.regex101.com/

2.1

Semantic Versioning 2.0.0 - https://semver.org/

SemVer Tag Selection

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImagePolicy

<snip>

spec:
  policy:
    semver:
      range: ‘>=1.2.0 <1.3.0’   # Range containing patch versions to minor version 1.2
```

```
$ flux create image policy nginxhello \
    --image-ref=nginxhello \
    --select-semver=‘>=1.20.x’ \
    --namespace=default \
    --export
```

Creating ImagePolicy Resources

Flux CLI enables the creation of ImagePolicy resources on our behalf

Demo

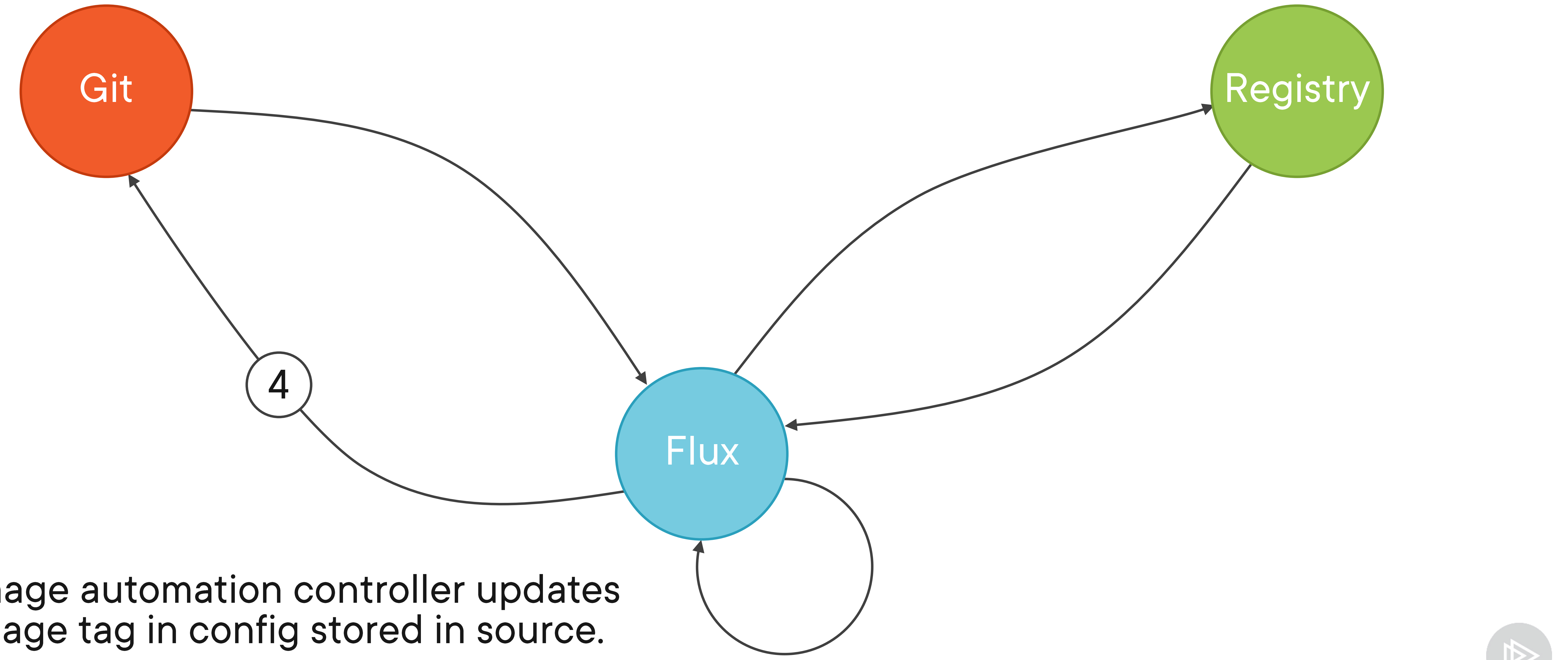
Applying Image Policy for a SemVer Range

- Add a new event source to Discord alert
- Create a new ImagePolicy resource
- Update the desired state in GitHub repo
- Check the app version selected by policy

Image Update Automation

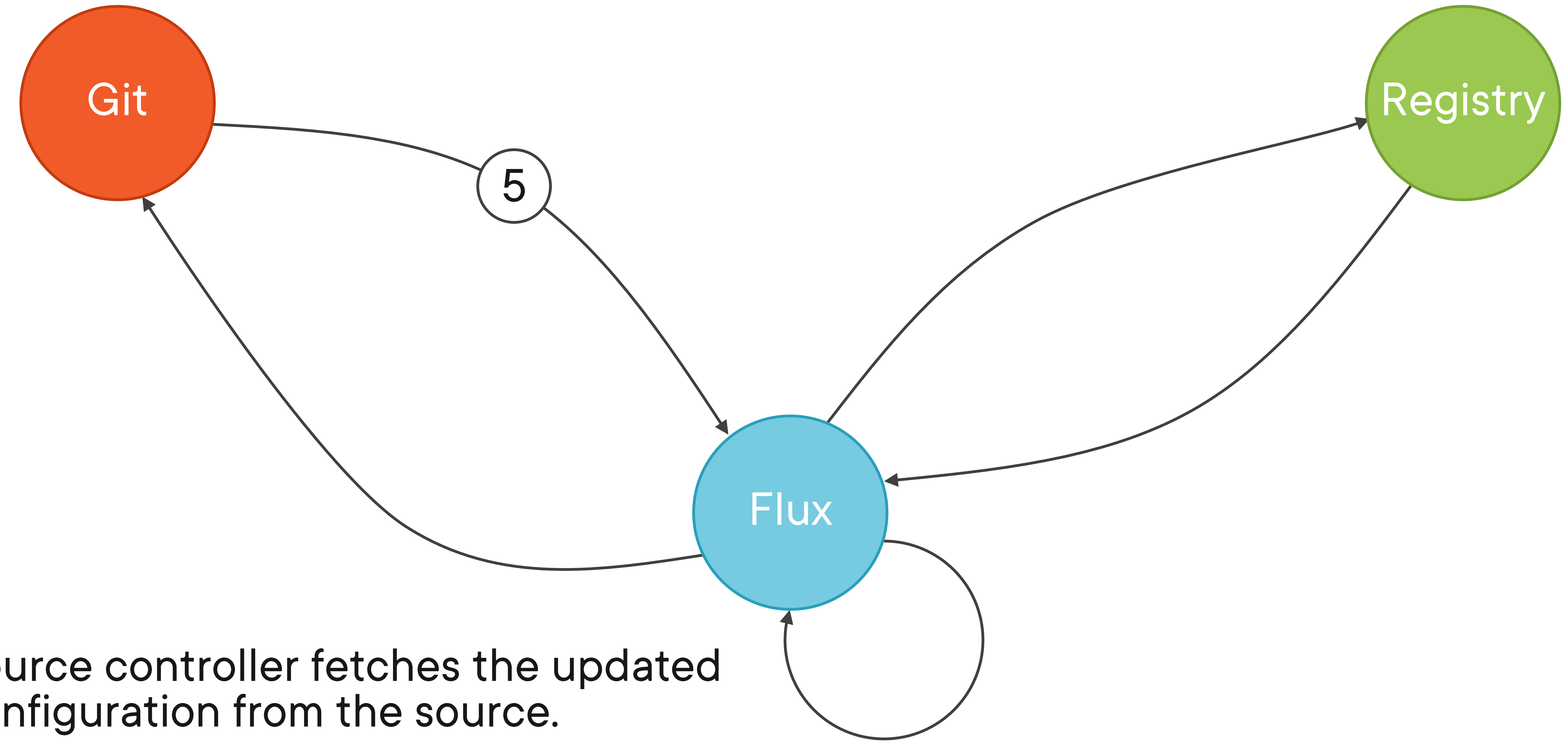
The means by which Flux updates the desired state in the source, with the most recent image tag acquired using image policy.

Updating Configuration in the Source



4 Image automation controller updates image tag in config stored in source.

Fetching the Updated Configuration



5 Source controller fetches the updated configuration from the source.

ImageUpdateAutomation API

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: nginxhello
  namespace: default
spec:
  sourceRef:
    kind: GitRepository
    name: nginxhello
    namespace: default
  update:
    path: ./deploy
    strategy: Setters

  <snip>
```

The source git repository is cloned, and the git reference defined in the source is checked out.

Cloning Details

```
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: nginxhello
  namespace: default
spec:

  <snip>

  git:
    checkout:
      ref:
        branch: staging

  <snip>
```

Push Branch

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: nginxhello
  namespace: default
spec:

  <snip>

  git:
    push:
      branch: main

  <snip>
```

Push Branch

```yaml
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: nginxhello
  namespace: default
spec:

  <snip>

  git:
    push:
      branch: flux-auto-updates

  <snip>
```

Commit Details

```
---
apiVersion: image.toolkit.fluxcd.io/v1beta1
kind: ImageUpdateAutomation
metadata:
  name: nginxhello
  namespace: default
spec:

  <snip>

  git:
    commit:
      author:
        email: flux@users.noreply.github.com
        name: flux
      messageTemplate: '{{range .Updated.Images}}{{println .}}{{end}}'

  <snip>
```

Commit message template data: https://bit.ly/3HH1FoP

```
$ flux create image update nginxhello \
    --git-repo-ref=nginxhello \
    --git-repo-path=./deploy \
    --checkout-branch=main \
    --push-branch=main \
    --author-name=flux \
    --author-email=flux@users.noreply.github.com \
    --commit-template="{{range .Updated.Images}}{{println .}}{{end}}" \
    --namespace=default \
    --export
```

Creating ImageUpdateAutomation Resources

The Flux CLI enables the creation of ImageUpdateAutomation resources

Which Resource to Update?

Deployment, StatefulSet, DaemonSet, CronJob?

A ‘marker’ provides a link with an ‘ImagePolicy’

The marker is a comment known as a ‘setter’

Image Automation Marker

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginxhello
spec:

  <snip>

  template:

    <snip>

    spec:
      containers:
      - image: nbrown/nginxhello:1.21.6
```

Image Automation Marker

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginxhello
spec:

  <snip>

  template:

    <snip>

    spec:
      containers:
      - image: nbrown/nginxhello:1.21.6 # {"$imagepolicy": "default:nginxhello"}
```

Demo

Updating an Application Version with Image Automation

- Add a marker to the workload manifest
- Create an ImageUpdateAutomation resource for app
- Witness a consequential app update
- Check the repo update made by Flux

Automating Packaged Releases with the Helm Controller

Module Summary

What we covered:

- Different approaches for updating app versions in desired state
- How Flux scans container image repos for image tags
- Tag selection with image policy definition
- Parameters required by Flux for updating desired state in source