Automating Packaged Releases with the Helm Controller

Nigel Brown

@n_brownuk www.windsock.io

Module Outline

Coming up:

- Helm and GitOps together
- Using Helm repositories as sources
- Automating Helm actions with Flux’s helm controller
- Planning for failure with remediation
- Wrapping up

Helm Charts

Packaged Kubernetes applications

Application lifecycle management

Flexible deployments using templates

Helm Release

Helm installs charts to Kubernetes clusters, creating a new ‘release’ for each installation of a chart.

A ‘release’ is tracked by Helm, and can be updated multiple times.

GitOps and Helm Releases

Flux treats Helm application releases as first-class citizens.

Helm Controller

Automation that performs Helm tasks.

Chart Sources

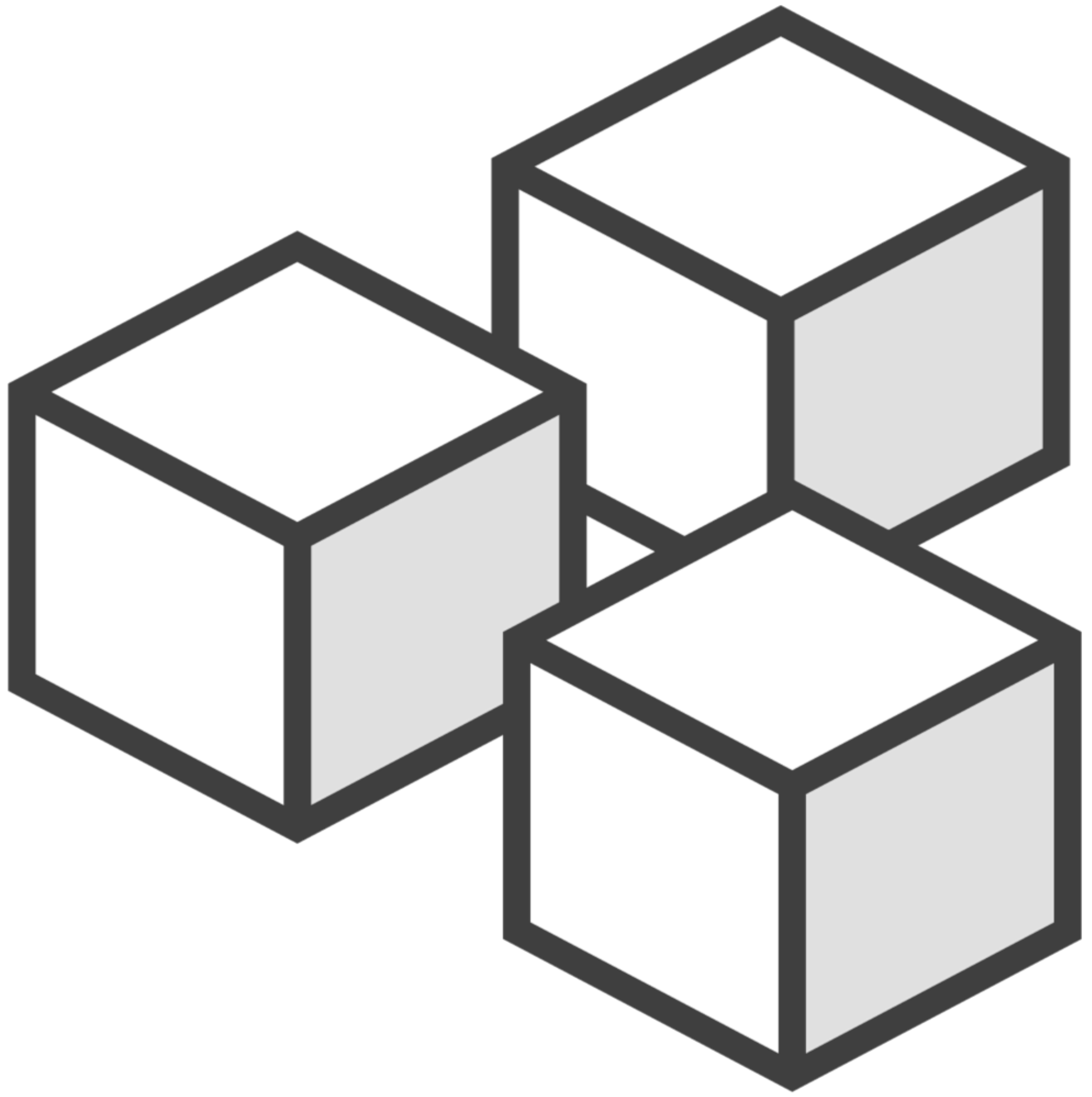
Remote sources that host Helm charts.

Release Definitions

Code that describes Helm releases.

Chart content can be fetched from several different storage medium types.

Helm Chart Sources



Storage buckets

- Accessible via an S3-compatible API

Git repositories

- Remotely-hosted git version control systems

Helm repositories

- Helm repo server or OCI-compliant registry

Choosing a Source Type

Bucket

Entire content of bucket fetched on each sync operation

Git repo

Easier to implement controlled change using native features

Helm repo

Semantic version constraints can be imposed on charts

HelmRepository API

Helm repositories are defined using a HelmRepository CRD

Helm Repository

```yaml
---
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: linkerd
  namespace: default
spec:
  interval: 1m0s
  url: https://helm.linkerd.io/stable
```

OCI Registry

```yaml
---
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: nginxhello
  namespace: default
spec:
  interval: 1m0s
  type: oci
  url: oci://ghcr.io/nbrownuk/charts
```

```yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: HelmRepository
metadata:
  name: nginxhello
  namespace: default
spec:

  <snip>

  secretRef:
    nginxhello-auth

  <snip>
```

Authenticating with Helm Repository Sources

Basic authentication credentials encoded in a referenced secret

OCI registry credentials encoded in a referenced secret of type 'docker-registry'

```
$ flux create source helm nginxhello \
    --url=oci://ghcr.io/nbrownuk/charts \
    --namespace=default \
    --export
```

Creating HelmRepository Resources

The Flux CLI allows for the creation of HelmRepository resources

Demo

Configuring a Helm Repository Source

- Locate the app’s chart repository
- Define a HelmRepository source
- Update the tracked GitHub repo
- Confirm object is created in the cluster

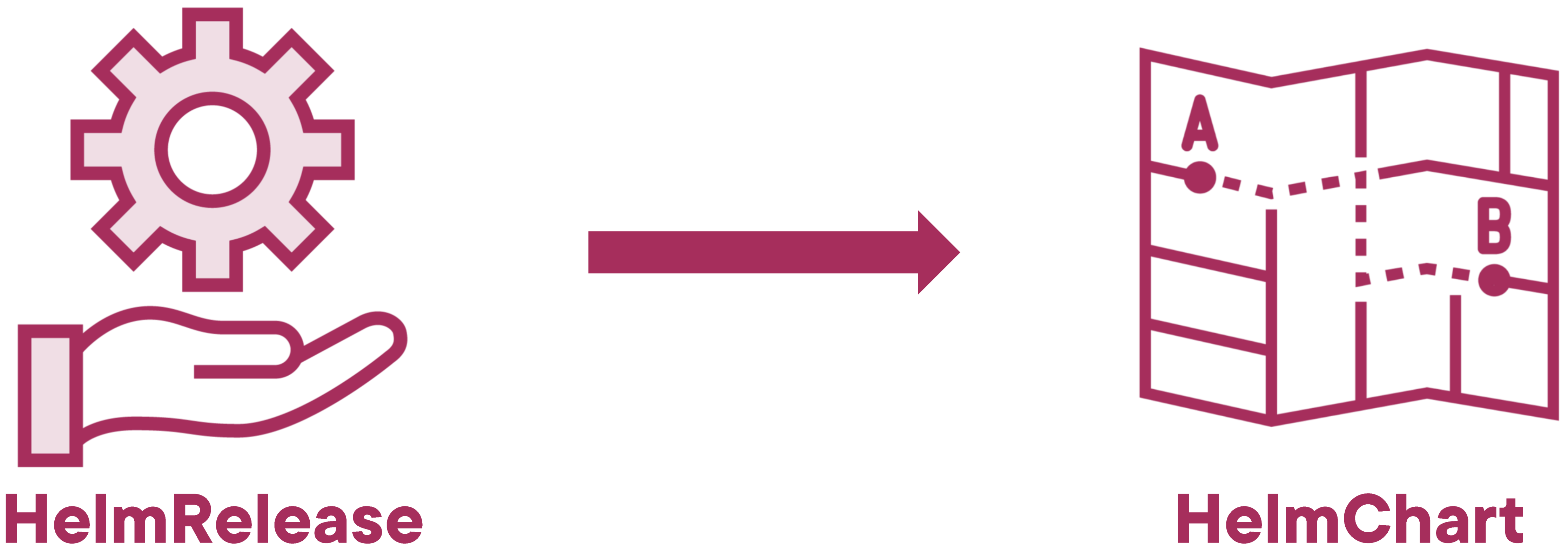
The helm controller uses the Helm SDK to mimic lifecycle actions for applications.

Context for Helm Actions

Command line parameters provide a context for Helm actions performed using its CLI.

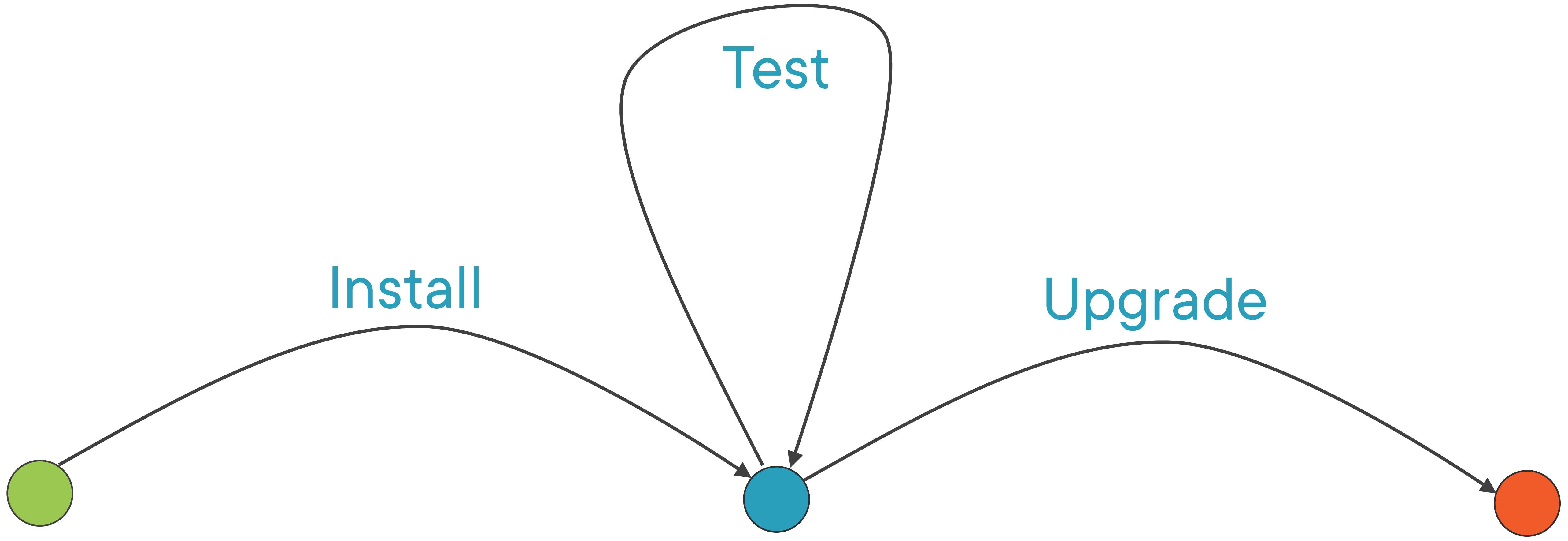
Custom Resources for the Helm Controller

The helm controller creates the HelmChart resource based on the HelmRelease definition



HelmRelease and HelmChart resources define the nature and content of Helm actions (e.g., install, rollback)

Helm Actions



Remediation

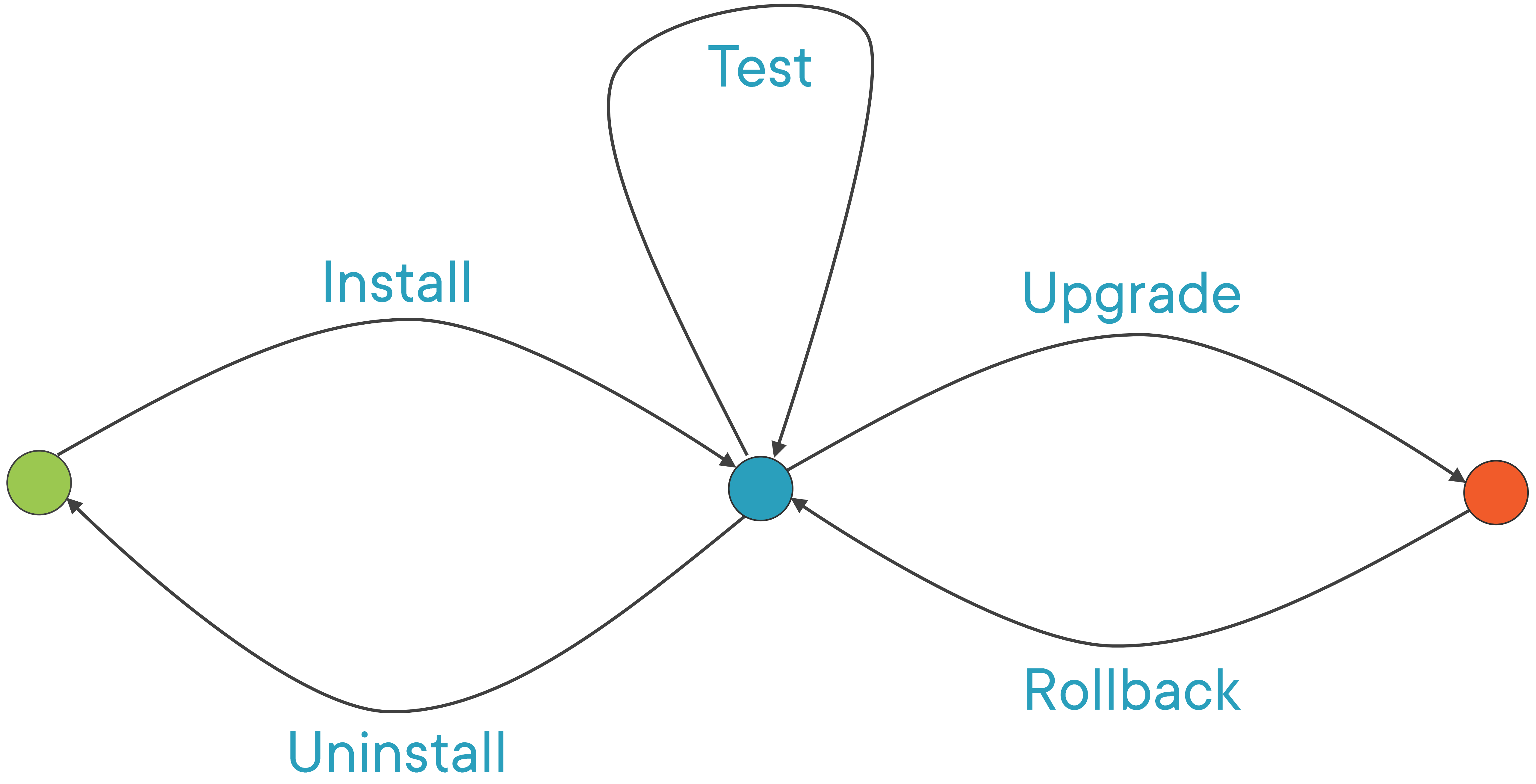


Chart Content

Helm repos

Charts are fetched from the Helm repo referenced in a source

Git repos and Buckets

Charts are built from the artifacts located in a referenced source

HelmRelease API

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  interval: 1m0s
  chart:
    spec:
      chart: nginxhello
      sourceRef:
        kind: HelmRepository
        name: nginxhello
      reconcileStrategy: ChartVersion
```

HelmRelease API

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  interval: 1m0s
  chart:
    spec:
      chart: nginxhello
      sourceRef:
        kind: GitRepository
        name: nginxhello
      reconcileStrategy: Revision
```

Chart Values

Helm charts are built with a set of default values

Values can be overridden to suit the purpose

Overriding Chart Values

Values can be provided inline, or from a Secret or ConfigMap

Inline

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  values:
    replicaCount: 5
    ingress:
      ingressClassName: nginx
```

Reference

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  valuesFrom:
  - kind: ConfigMap
    name: chart-values
```

Ordering and Placement of Values

Values from referenced objects are merged in the order defined, with later values taking precedence

Values defined inline in the HelmRelease resource, override any values taken from referenced objects

A single value can be merged at a ‘target path’ in the YAML, defined using dot syntax (e.g., ingress.ingressClassName)

Access Control for HelmRelease Objects

Cluster admin privileges

HelmRelease resources can manipulate objects across the entire cluster.

Access Control

Role-based access control (RBAC) can be used to limit the scope of operations available.

Access Control

Use role-based access control (RBAC) to limit the type and scope of object modification allowed.

Demo

Automating a Helm Chart Release

- Configure a HelmRelease resource
- Push new config to remote GitHub repo
- Observe creation of custom resources
- Check for successful install of our app

“Anything that can go wrong, will go wrong”

Murphy’s Law

Remediation Actions

Default behavior

Do nothing, unless remediation is configured.

Failed install

Perform an uninstall action for the failed Helm release.

Failed upgrade

Perform a rollback, unless the strategy is set to uninstall.

Test Failures

Helm test failures automatically trigger remediation. Must be explicitly ignored if this is undesired behavior.

Helm Action Retries

Installs or upgrades are performed ad infinitum when the retries field is set to -1

When the retries field is set to default value of 0, no retries are performed on failure

A positive integer value for the retries field, governs how many retries are attempted (e.g., 2)

Configuring Remediation for an Install

```yaml
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  chart:

  <snip>

  install:
    remediation:
      retries: 2
      remediateLastFailure: true
```

Configuring Remediation for an Upgrade

```
---
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: nginxhello
  namespace: default
spec:
  chart:

  <snip>

  upgrade:
    remediation:
      retries: 2
      strategy: ‘uninstall’
      remediateLastFailure: false
```

Demo

Implementing a Rollback for a Failed Helm Upgrade

- Configure HelmRelease with remediation
- Create a new version of the Helm chart
- Push new chart to Helm repository
- Observe the upgrade and remediation
- Confirm final rollback to previous version

Wrapping Up

What we covered:

- Helm compliments a GitOps approach
- HelmRepository resources define Helm chart sources
- Helm controller accommodates the Helm release concept
- HelmRelease API allows for defining remediation

Well done for getting to the end!

Where to Go Next

OpenGitOps, a set of open-source standards, best practices, and community-focused education - https://opengitops.dev/

GitOps Days (https://www.gitopsdays.com/) and GitOpsCon

Flux project documentation - https://fluxcd.io/flux/

Final Words

Feedback

Discussion