Supercharge your AKS Deployments

with GitOps and Flux V2

Who Am I?

- Geert Baeke
- Twitter: @geertbaeke
- Blog: https://blog.baeke.info
- GitHub: https://github.com/gbaeke
- YouTube: https://youtube.com/geertbaeke

What is GitOps? ✓

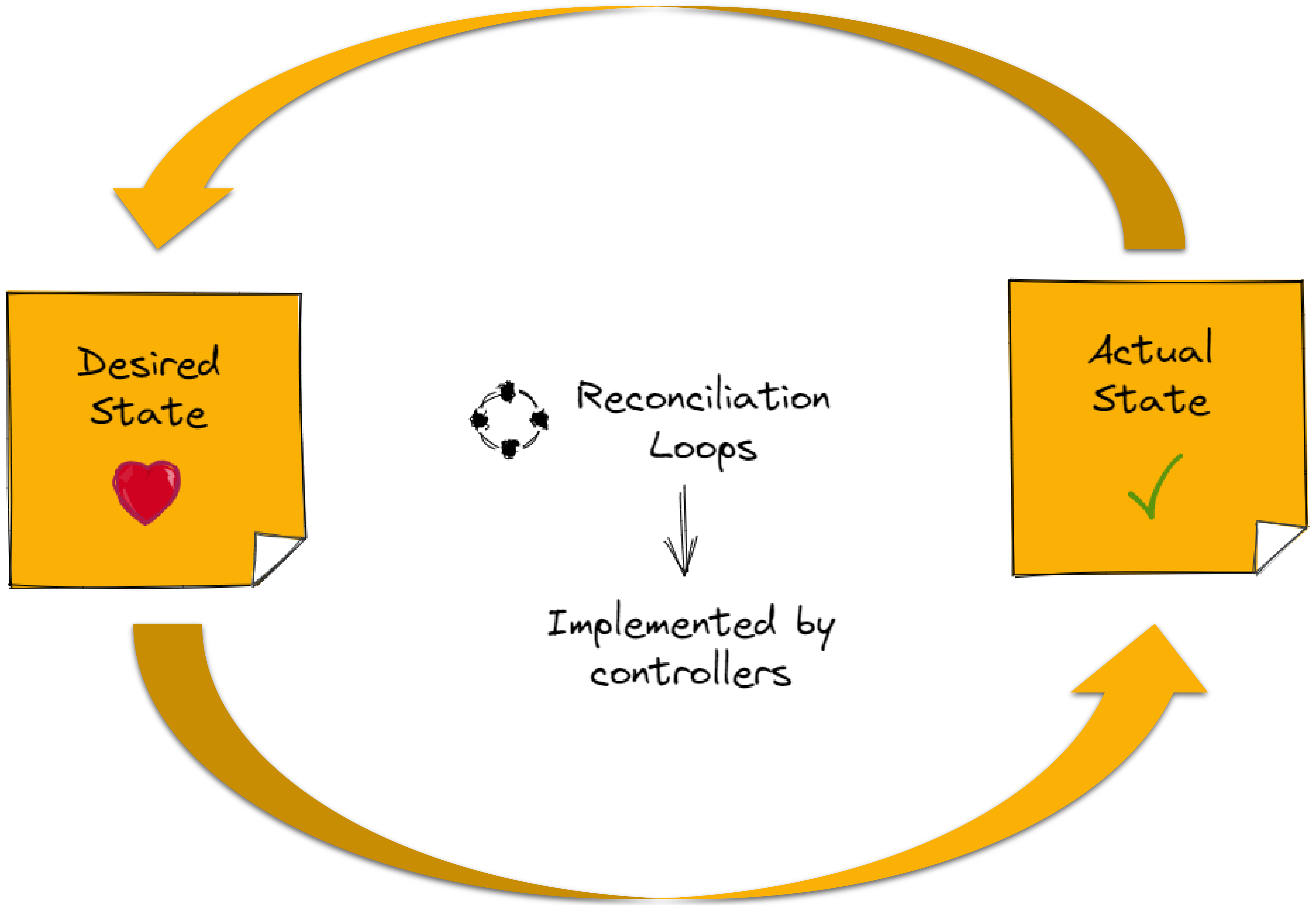
GitOps vs CIOps

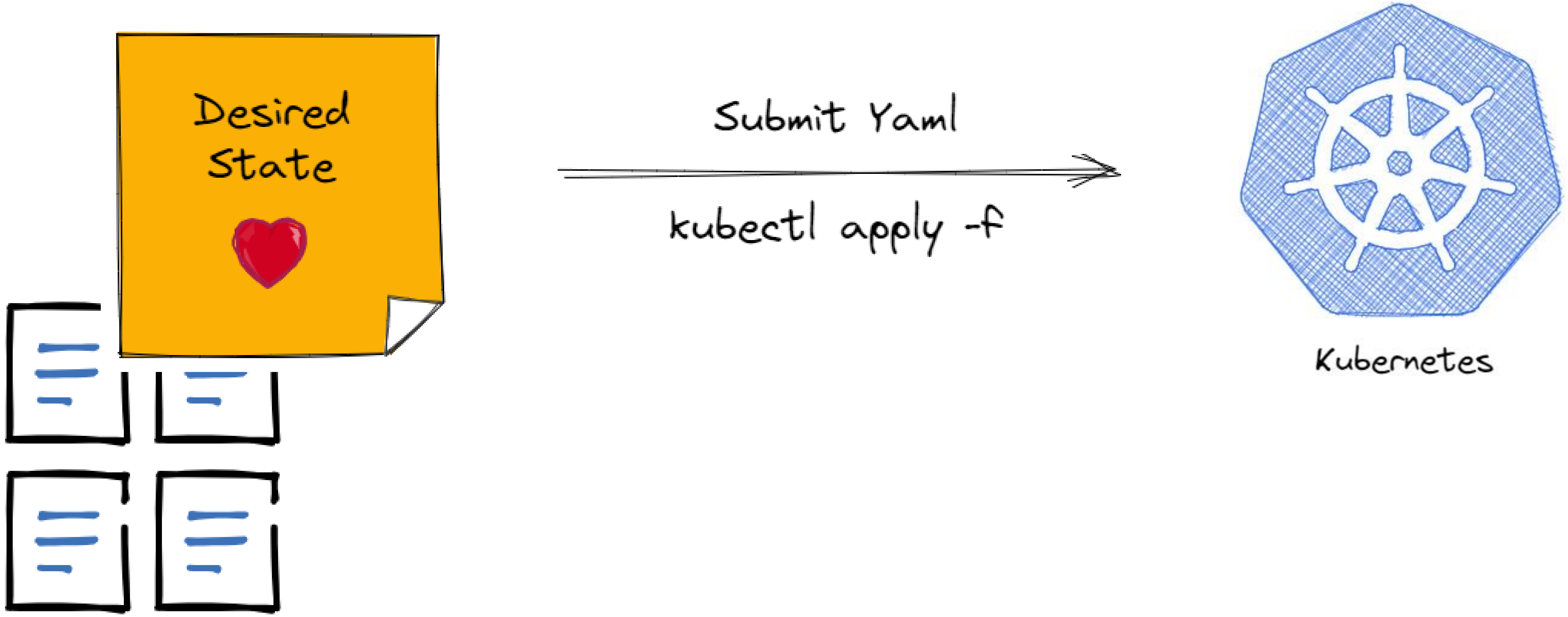
Flux v2

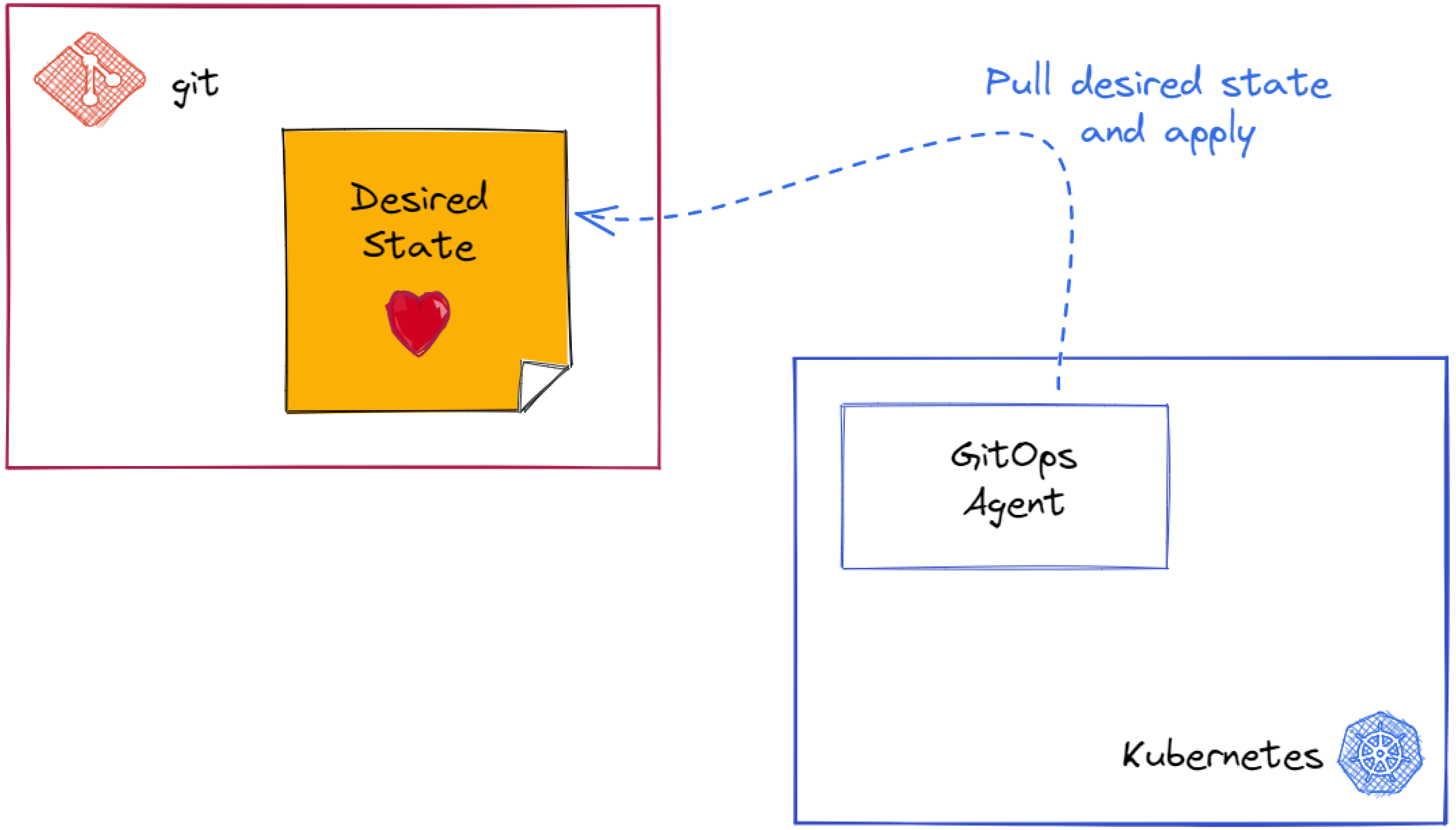
GitOps on AKS

Kubernetes

Git

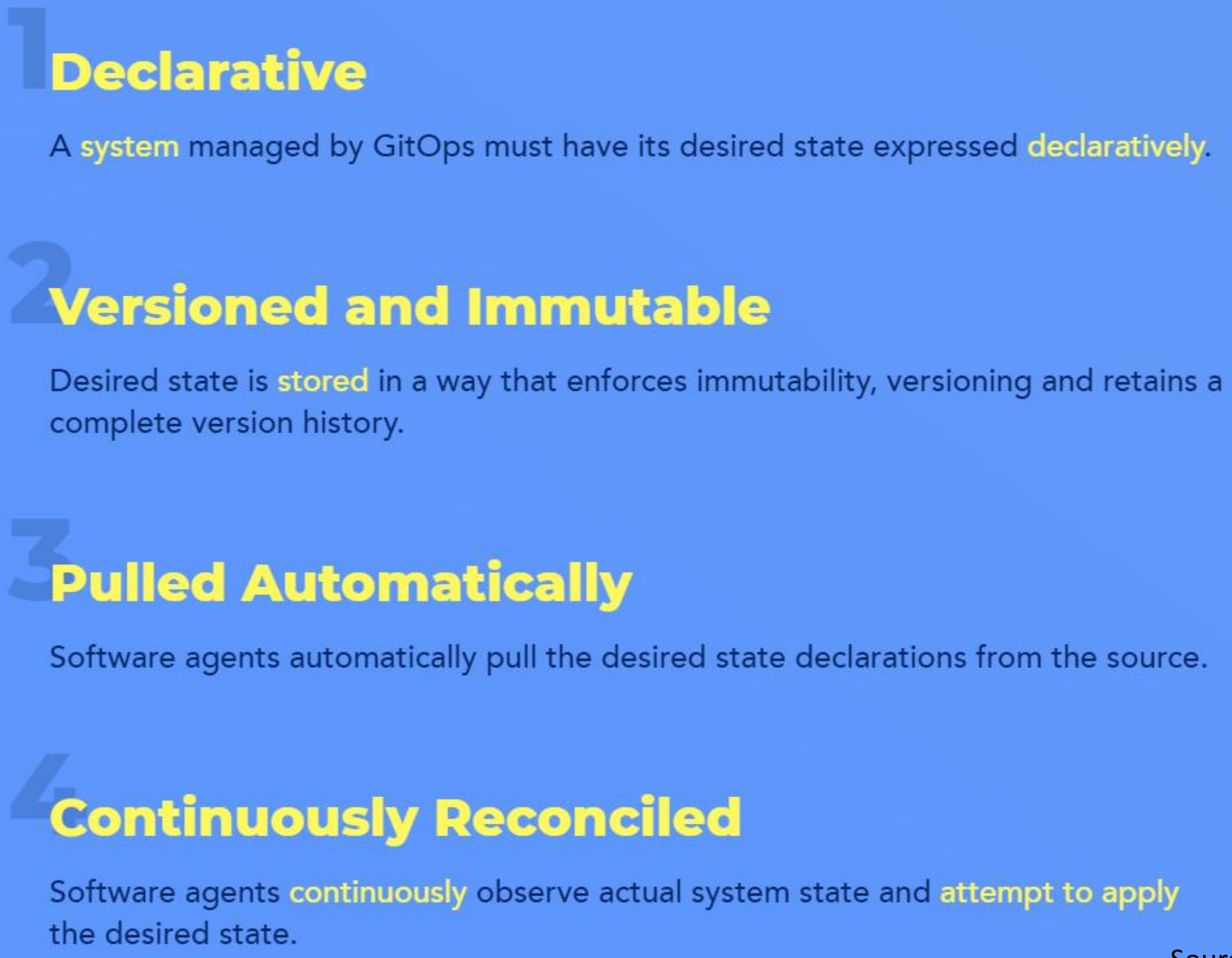

Desired State
Reconciliation Loops
Implemented by controllers
Actual State


Desired State
Submit Yaml
kubectl apply -f
Kubernetes


git
Desired State
Pull desired state and apply
GitOps Agent
Kubernetes

KEEP
CALM

CONTINUE
DOING CI



1 Declarative

A system managed by GitOps must have its desired state expressed declaratively.

2 Versioned and Immutable

Desired state is stored in a way that enforces immutability, versioning and retains a complete version history.

3 Pulled Automatically

Software agents automatically pull the desired state declarations from the source.

4 Continuously Reconciled

Software agents continuously observe actual system state and attempt to apply the desired state.

Source: https://opengitops.dev

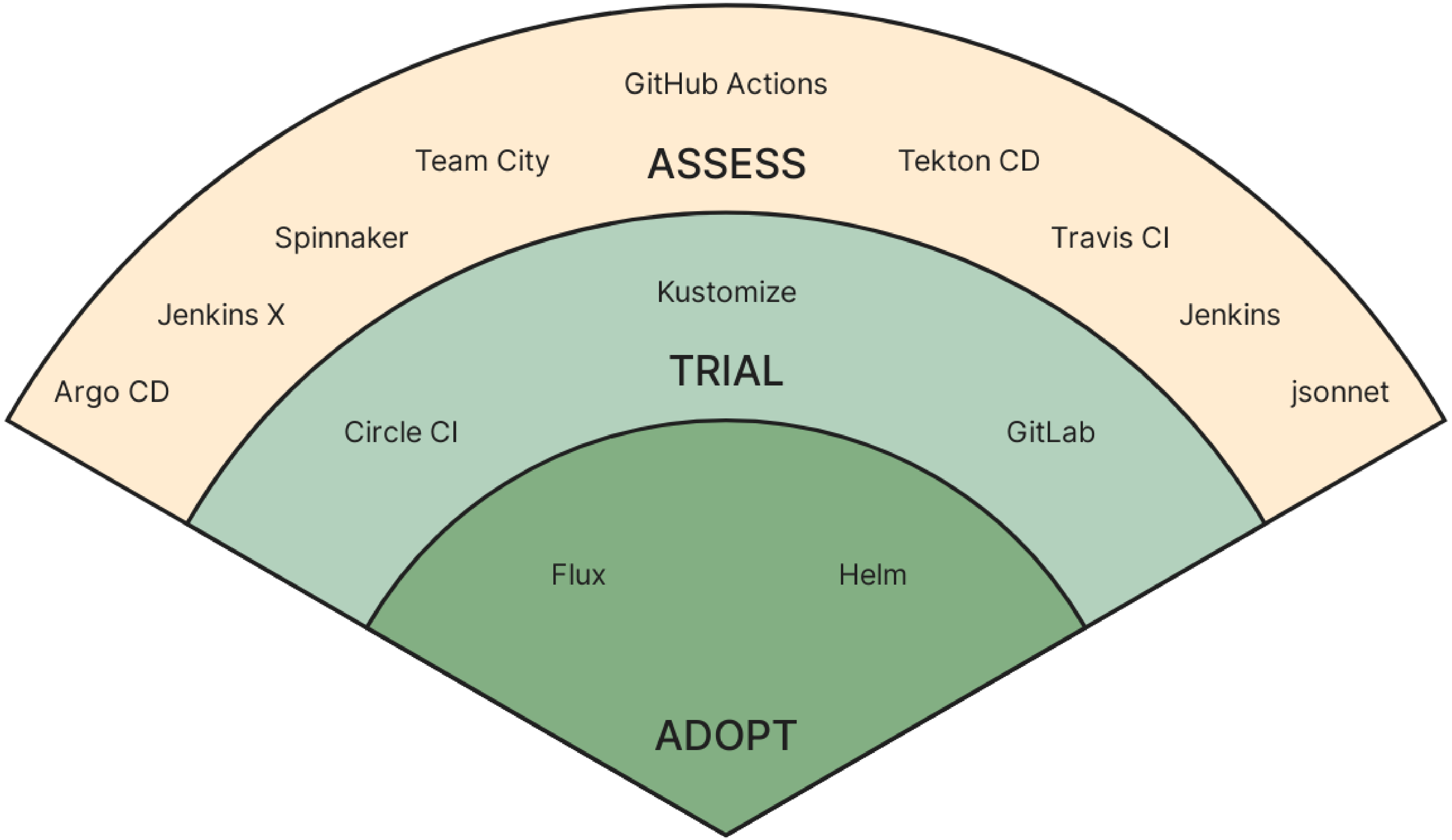
Pipelines	Steps/Tasks	Credentials	Push-based
Driven by CI/CD systems	Imperative	Store K8S Credentials	Connectivity & Local Agents

Flux v2

Open-source GitOps solution

🎓 Flux is a CNCF Graduated project!

CNCF Technology Radar Continuous Delivery, June 2020



Source: [Continuous Delivery | CNCF Radars]

Microsoft

D2IQ

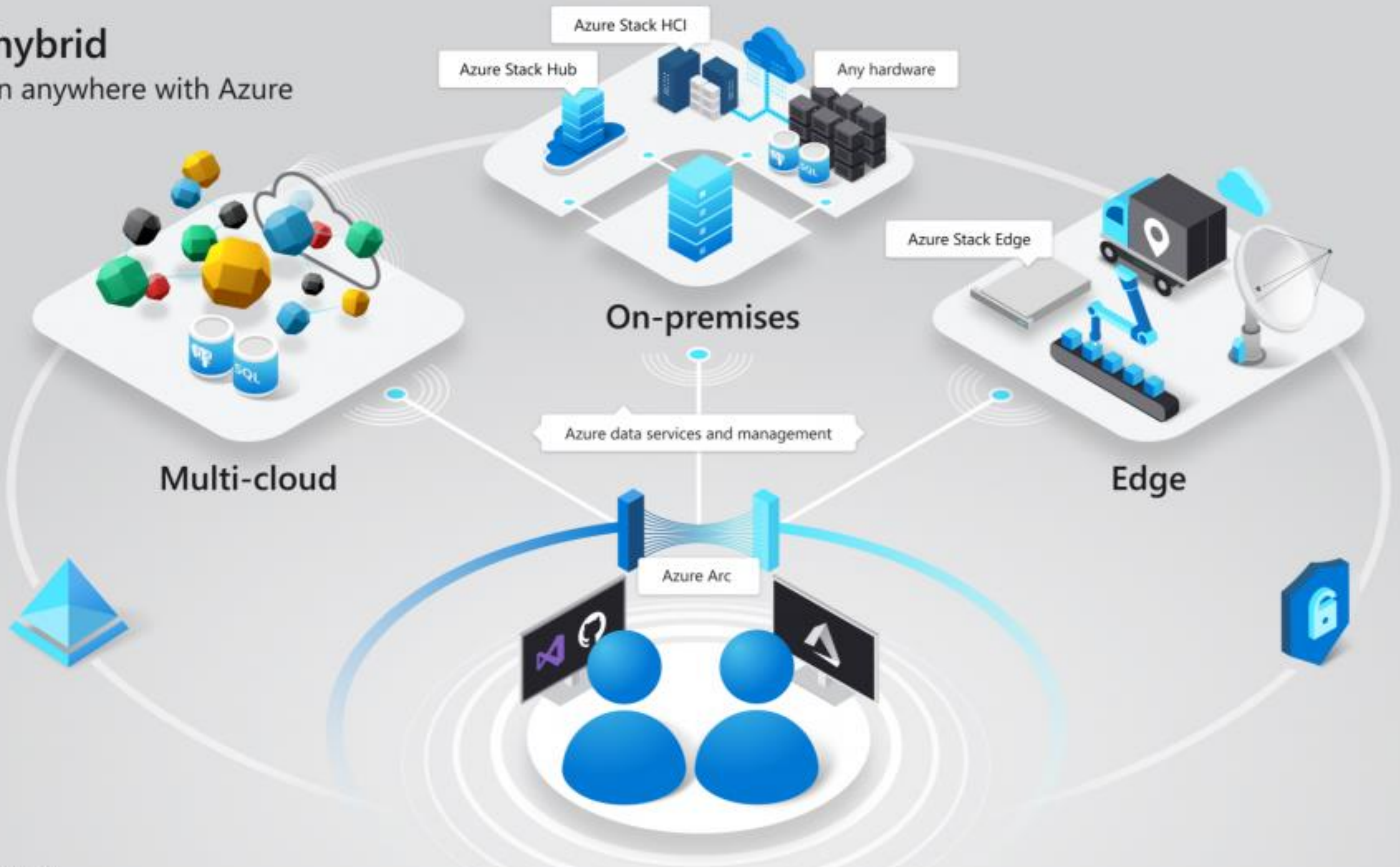
OPENSHIFT

vmware®

amazon

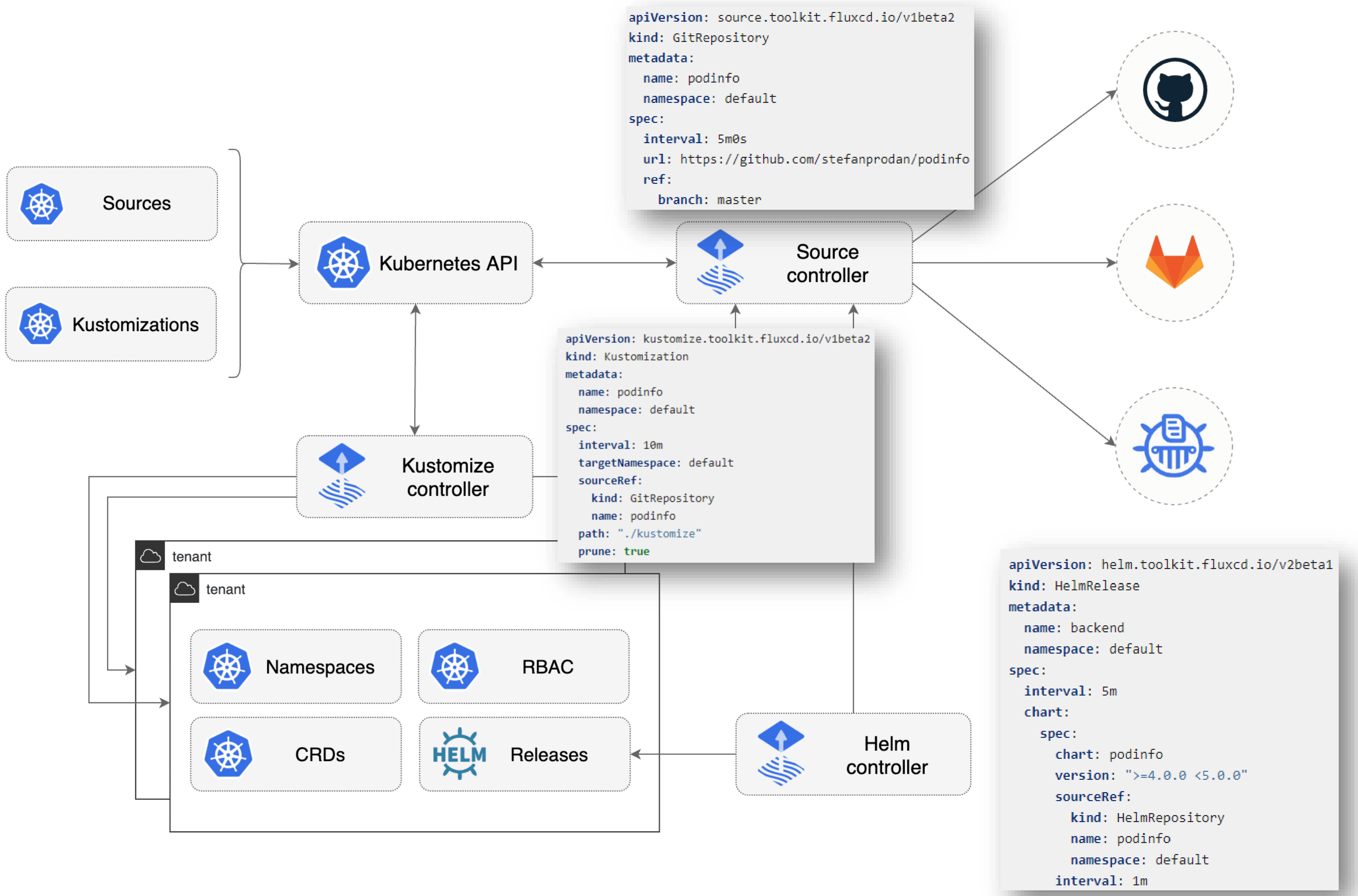
Azure hybrid

Innovation anywhere with Azure



Microsoft Azure

azure.com/hybrid

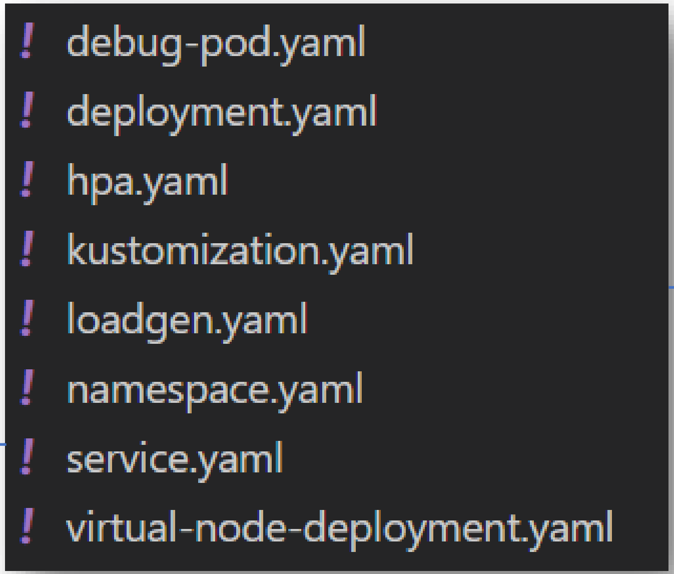


```yaml
apiVersion: source.toolkit.fluxcd.io/v1beta2
kind: GitRepository
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 5m0s
  url: https://github.com/stefanprodan/podinfo
  ref:
    branch: master
```

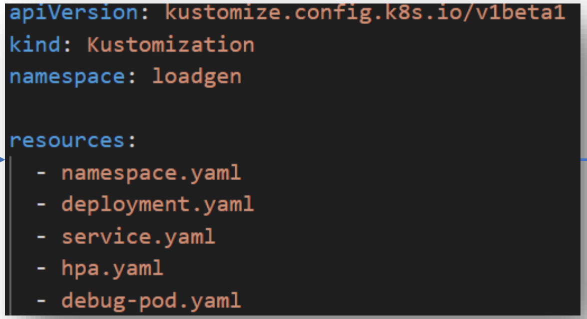
```yaml
apiVersion: kustomize.toolkit.fluxcd.io/v1beta2
kind: Kustomization
metadata:
  name: podinfo
  namespace: default
spec:
  interval: 10m
  targetNamespace: default
  sourceRef:
    kind: GitRepository
    name: podinfo
  path: "./kustomize"
  prune: true
```

```yaml
apiVersion: helm.toolkit.fluxcd.io/v2beta1
kind: HelmRelease
metadata:
  name: backend
  namespace: default
spec:
  interval: 5m
  chart:
    spec:
      chart: podinfo
      version: ">=4.0.0 <5.0.0"
      sourceRef:
        kind: HelmRepository
        name: podinfo
        namespace: default
      interval: 1m
```

Source: Flux Documentation | Flux (fluxcd.io)



```
! debug-pod.yaml
! deployment.yaml
! hpa.yaml
! kustomization.yaml
! loadgen.yaml
! namespace.yaml
! service.yaml
! virtual-node-deployment.yaml
```

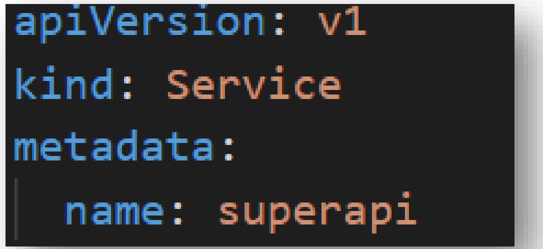


```
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization
namespace: loadgen

resources:
  - namespace.yaml
  - deployment.yaml
  - service.yaml
  - hpa.yaml
  - debug-pod.yaml
```

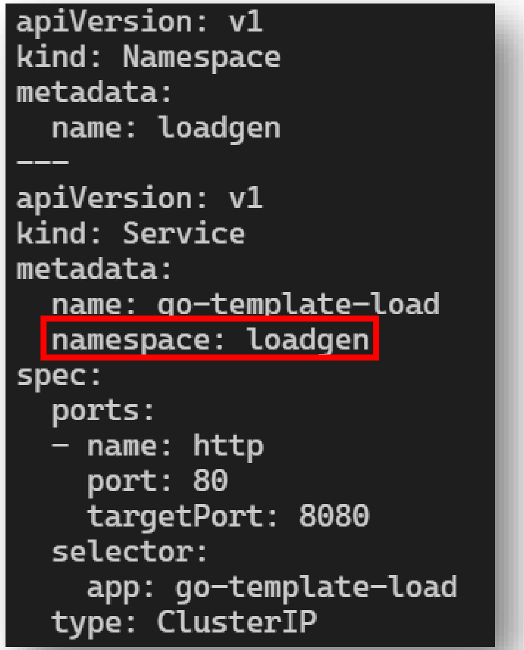
kustomization.yaml

Run “**kubectl kustomize .**” in folder containing these files

original service.yaml without namespace

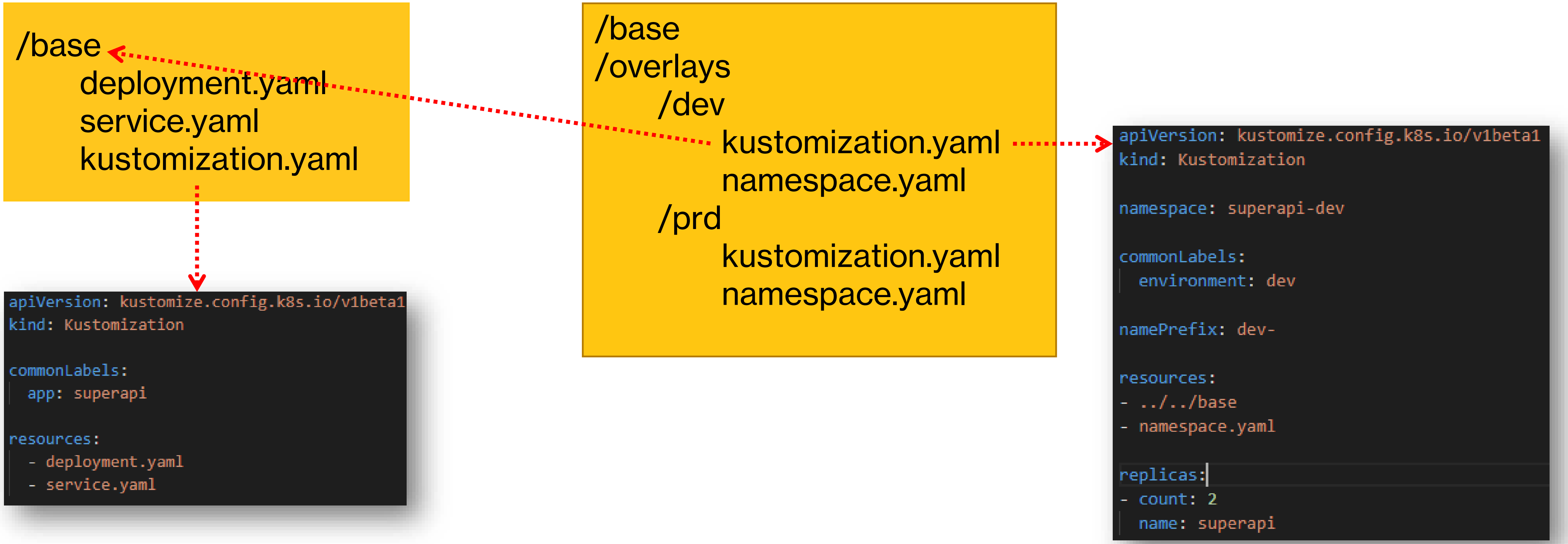


```
apiVersion: v1
kind: Service
metadata:
  name: superapi
```



```
apiVersion: v1
kind: Namespace
metadata:
  name: loadgen
---
apiVersion: v1
kind: Service
metadata:
  name: go-template-load
  namespace: loadgen
spec:
  ports:
  - name: http
    port: 80
    targetPort: 8080
  selector:
    app: go-template-load
  type: ClusterIP
```

result is a composition of resources in one YAML manifest



```
/base
      deployment.yaml
      service.yaml
      kustomization.yaml
```

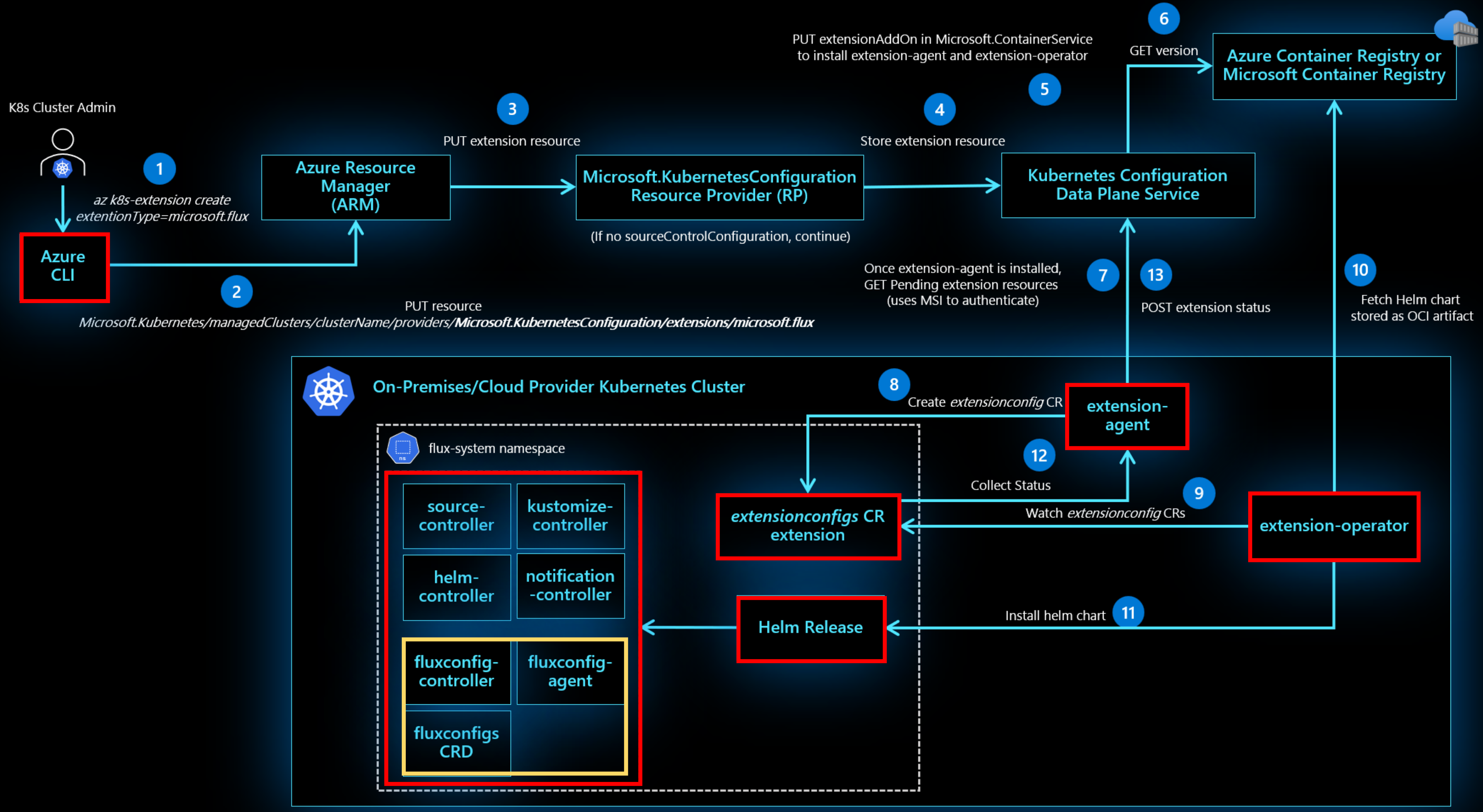
```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

commonLabels:
  app: superapi

resources:
  - deployment.yaml
  - service.yaml
```

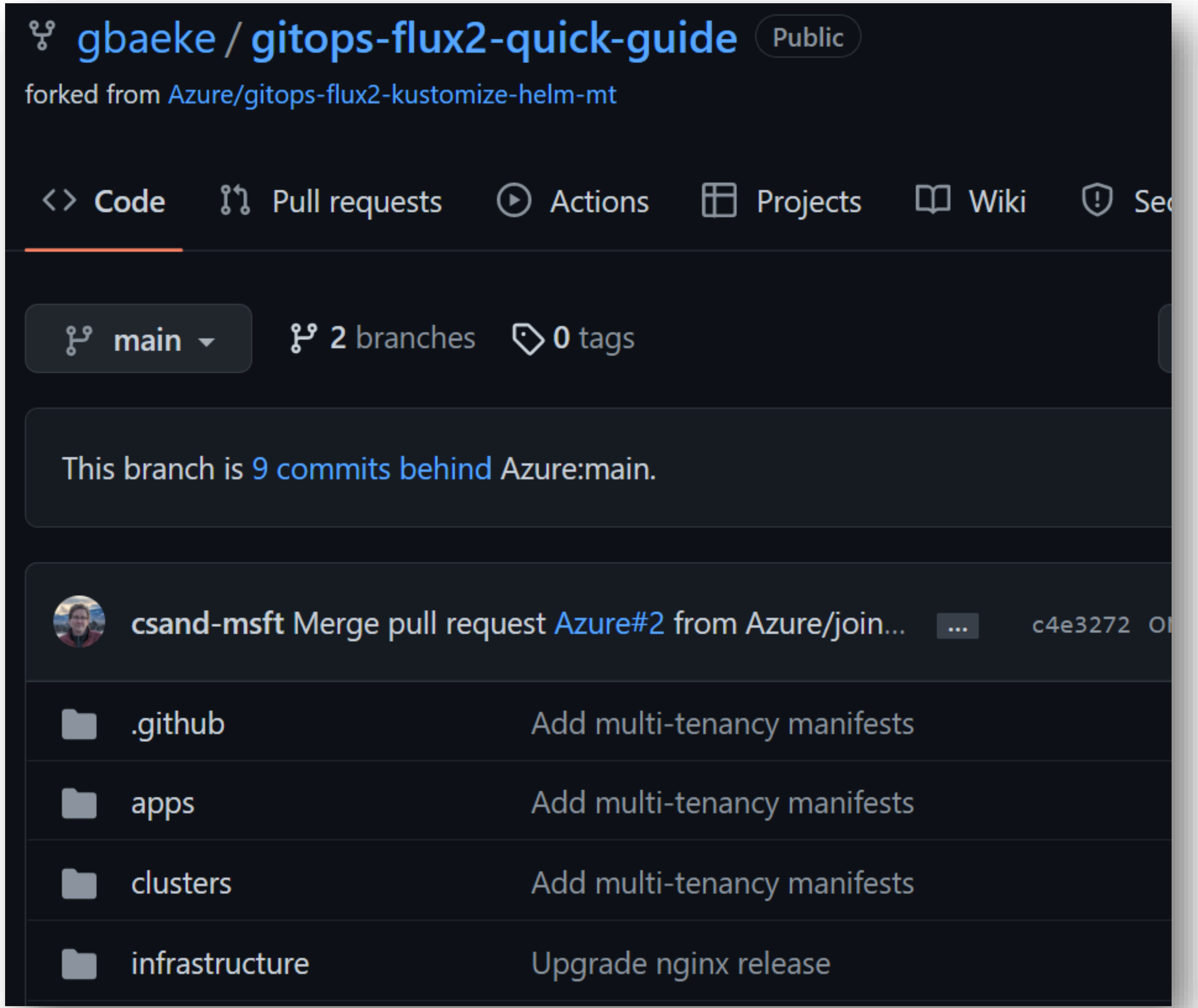
```
/base
/overlays
      /dev
            kustomization.yaml
            namespace.yaml
      /prd
            kustomization.yaml
            namespace.yaml
```

```yaml
apiVersion: kustomize.config.k8s.io/v1beta1
kind: Kustomization

namespace: superapi-dev

commonLabels:
  environment: dev

namePrefix: dev-

resources:
- ../../base
- namespace.yaml

replicas:
- count: 2
  name: superapi
```

GitOps on AKS

with the microsoft.flux extension


K8s Cluster Admin
1
az k8s-extension create extentionType=microsoft.flux
Azure CLI
2
PUT resource
Microsoft.Kubernetes/managedClusters/clusterName/providers/Microsoft.KubernetesConfiguration/extensions/microsoft.flux
Azure Resource Manager (ARM)
3
PUT extension resource
Microsoft.KubernetesConfiguration Resource Provider (RP)
(If no sourceControlConfiguration, continue)
4
Store extension resource
Kubernetes Configuration Data Plane Service
PUT extensionAddOn in Microsoft.ContainerService to install extension-agent and extension-operator
5
6
GET version
Azure Container Registry or Microsoft Container Registry
Once extension-agent is installed, GET Pending extension resources (uses MSI to authenticate)
7
13
POST extension status
10
Fetch Helm chart stored as OCI artifact
On-Premises/Cloud Provider Kubernetes Cluster
8
Create extensionconfig CR
extension-agent
flux-system namespace
source-controller
kustomize-controller
helm-controller
notification-controller
fluxconfig-controller
fluxconfig-agent
fluxconfigs CRD
extensionconfigs CR extension
12
Collect Status
9
Watch extensionconfig CRs
extension-operator
Helm Release
11
Install helm chart



gbaeke / gitops-flux2-quick-guide Public

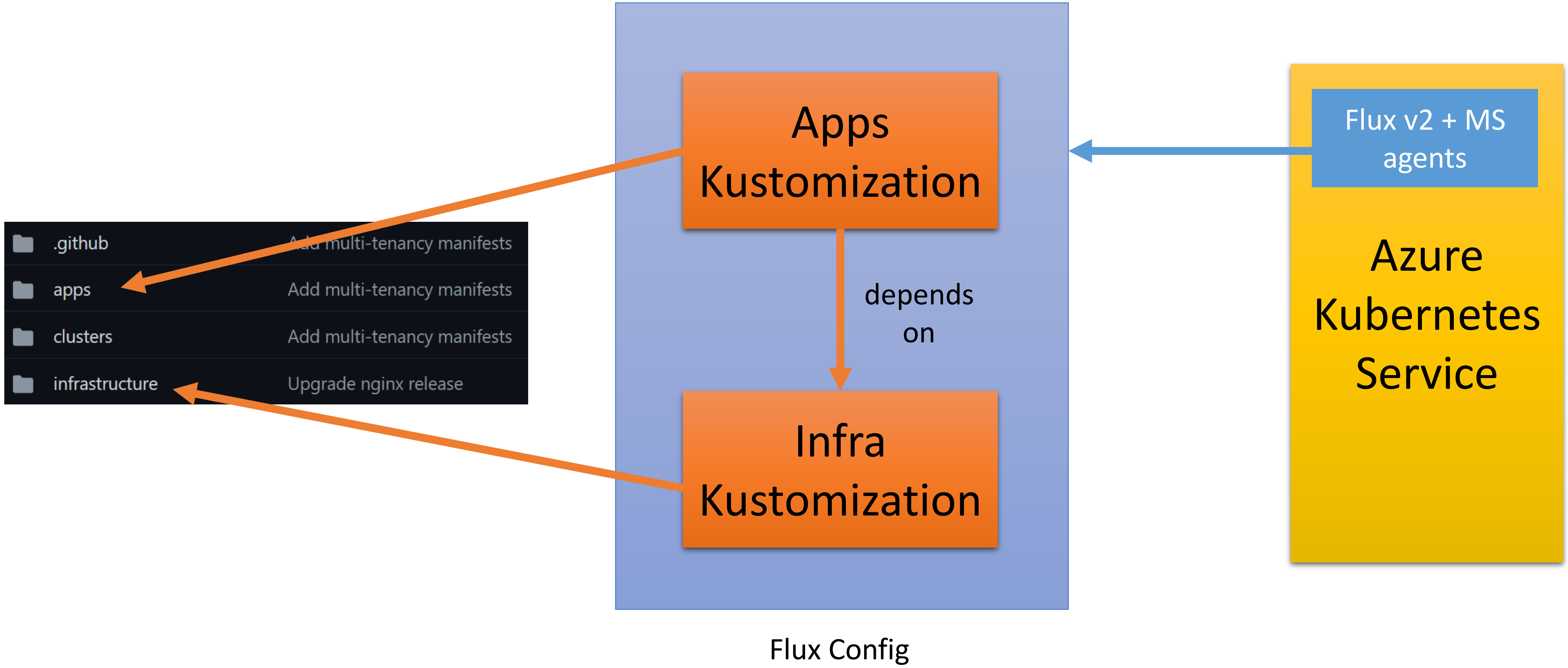
forked from Azure/gitops-flux2-kustomize-helm-mt

This branch is 9 commits behind Azure:main.

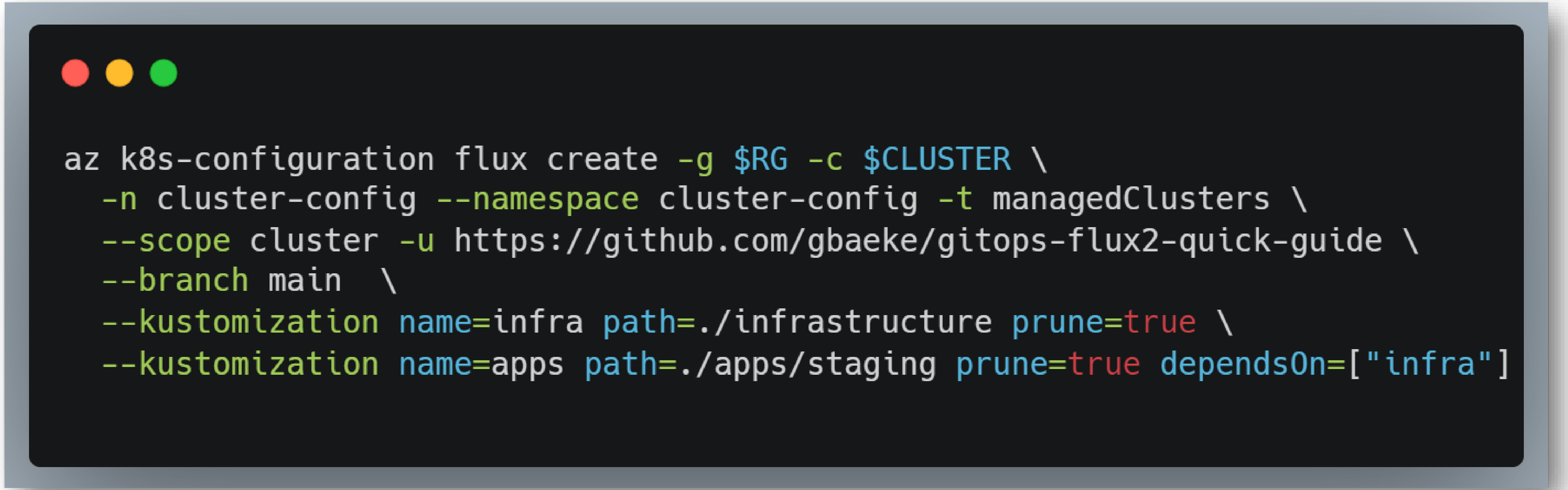
csand-msft Merge pull request Azure#2 from Azure/join... c4e3272

.github	Add multi-tenancy manifests
apps	Add multi-tenancy manifests
clusters	Add multi-tenancy manifests
infrastructure	Upgrade nginx release

https://bit. y/3RItPUP l

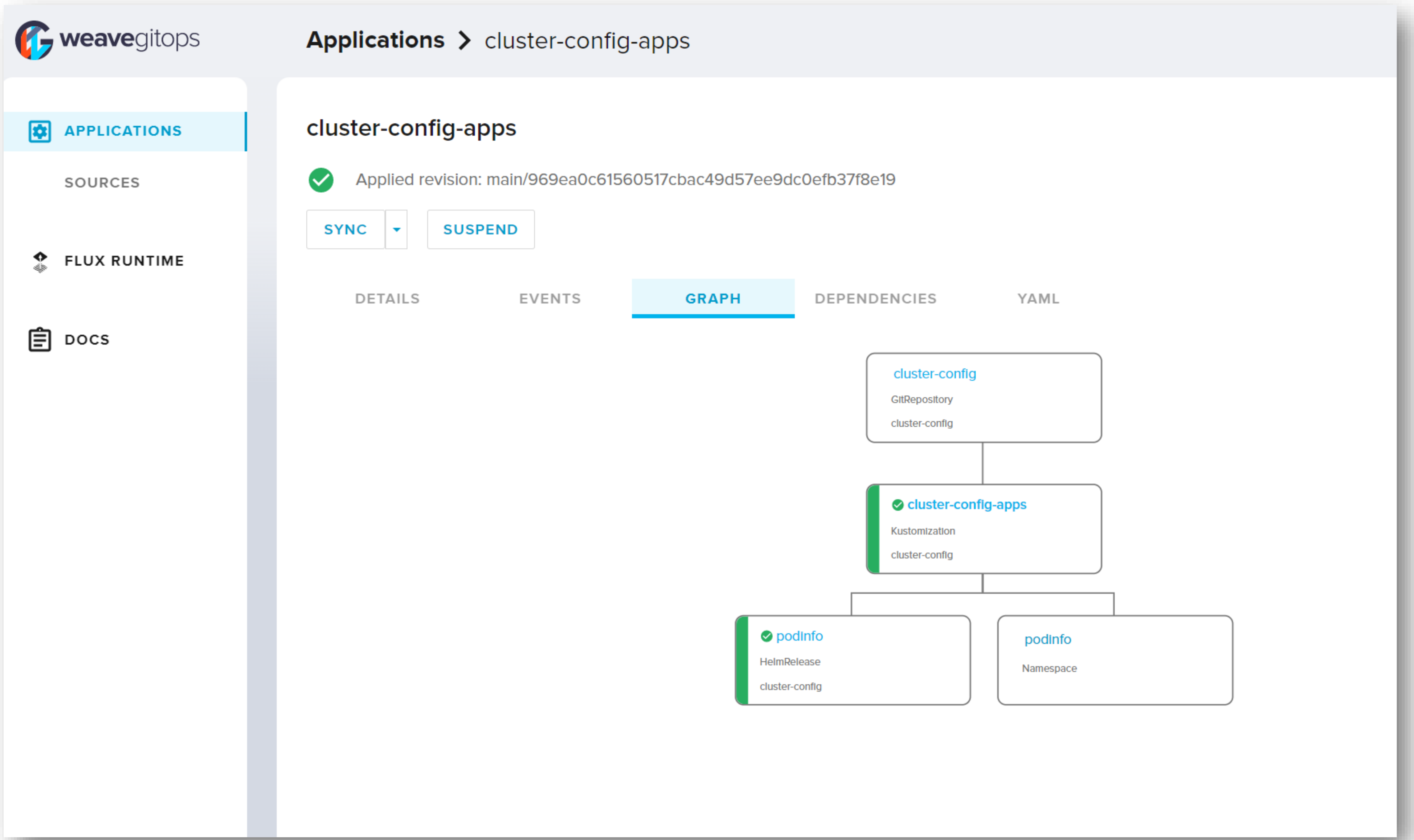


.github	Add multi-tenancy manifests
apps	Add multi-tenancy manifests
clusters	Add multi-tenancy manifests
infrastructure	Upgrade nginx release



```
az k8s-configuration flux create -g $RG -c $CLUSTER \
  -n cluster-config --namespace cluster-config -t managedClusters \
  --scope cluster -u https://github.com/gbaeke/gitops-flux2-quick-guide \
  --branch main  \
  --kustomization name=infra path=./infrastructure prune=true \
  --kustomization name=apps path=./apps/staging prune=true dependsOn=["infra"]
```

⚠ The above commands results in one source (git repository) and two "kustomizations"
The Source Controller and Kustomization controllers take action when they see these resources in the "cluster-config" namespace.



cluster-config-apps

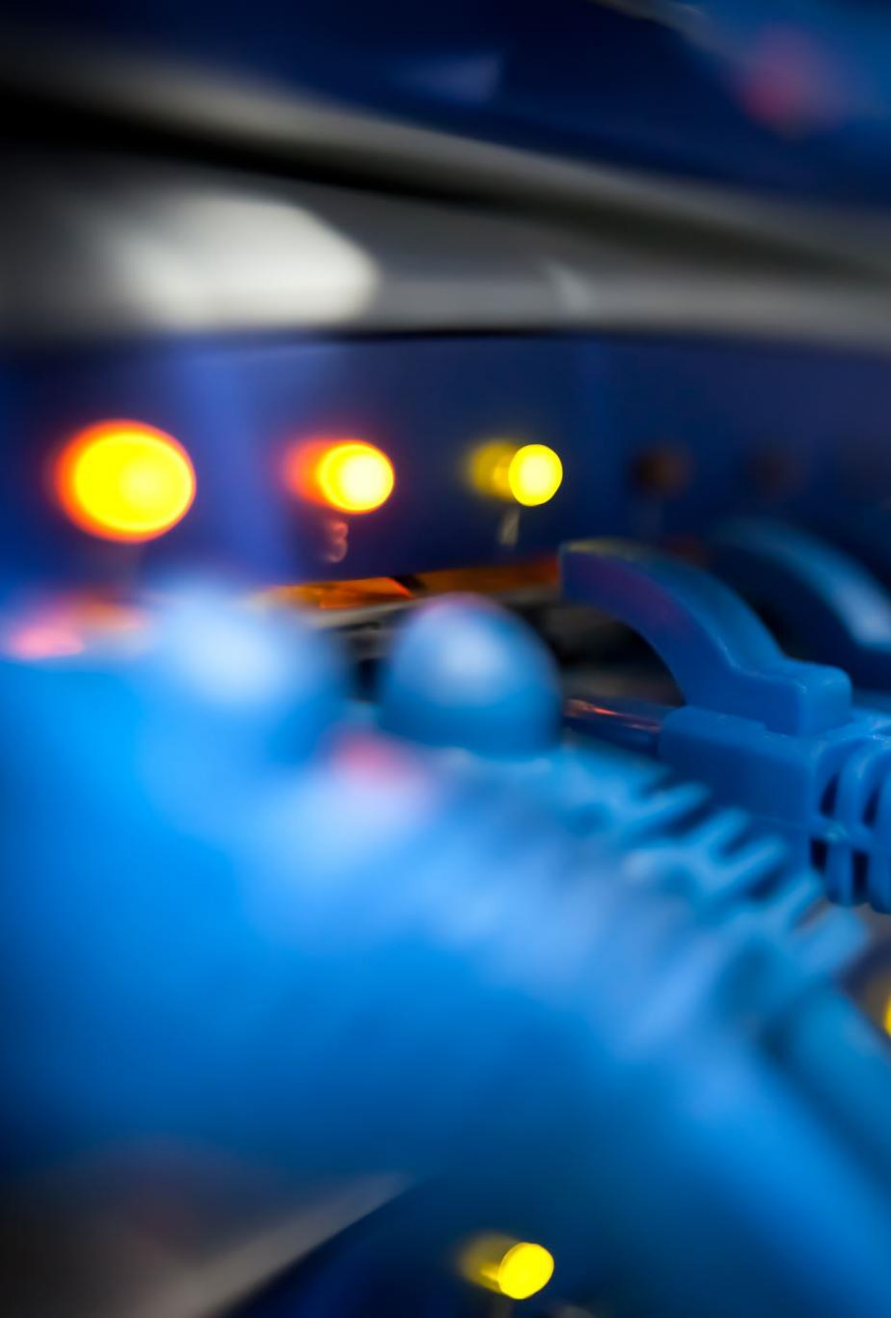
Applied revision: main/969ea0c61560517cbac49d57ee9dc0efb37f8e19

cluster-config
GitRepository
cluster-config

cluster-config-apps
Kustomization
cluster-config

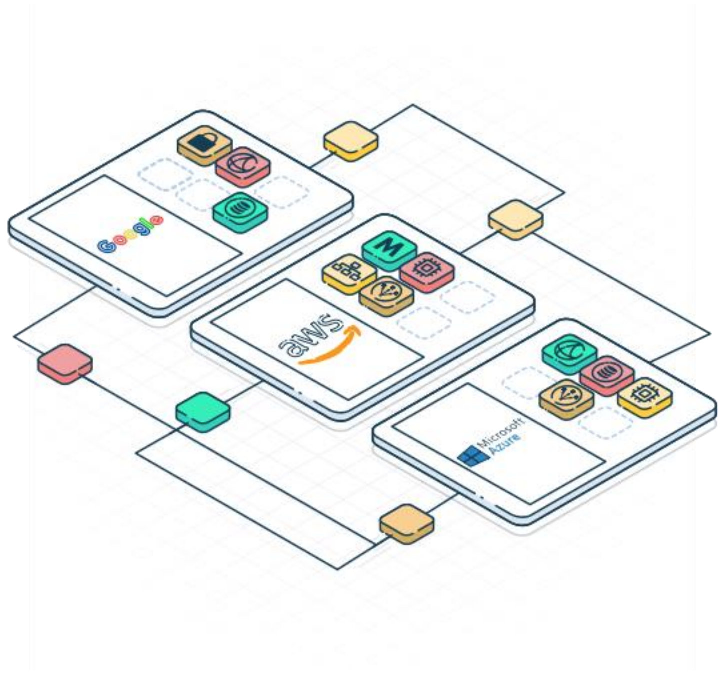
podinfo
HelmRelease
cluster-config

podinfo
Namespace



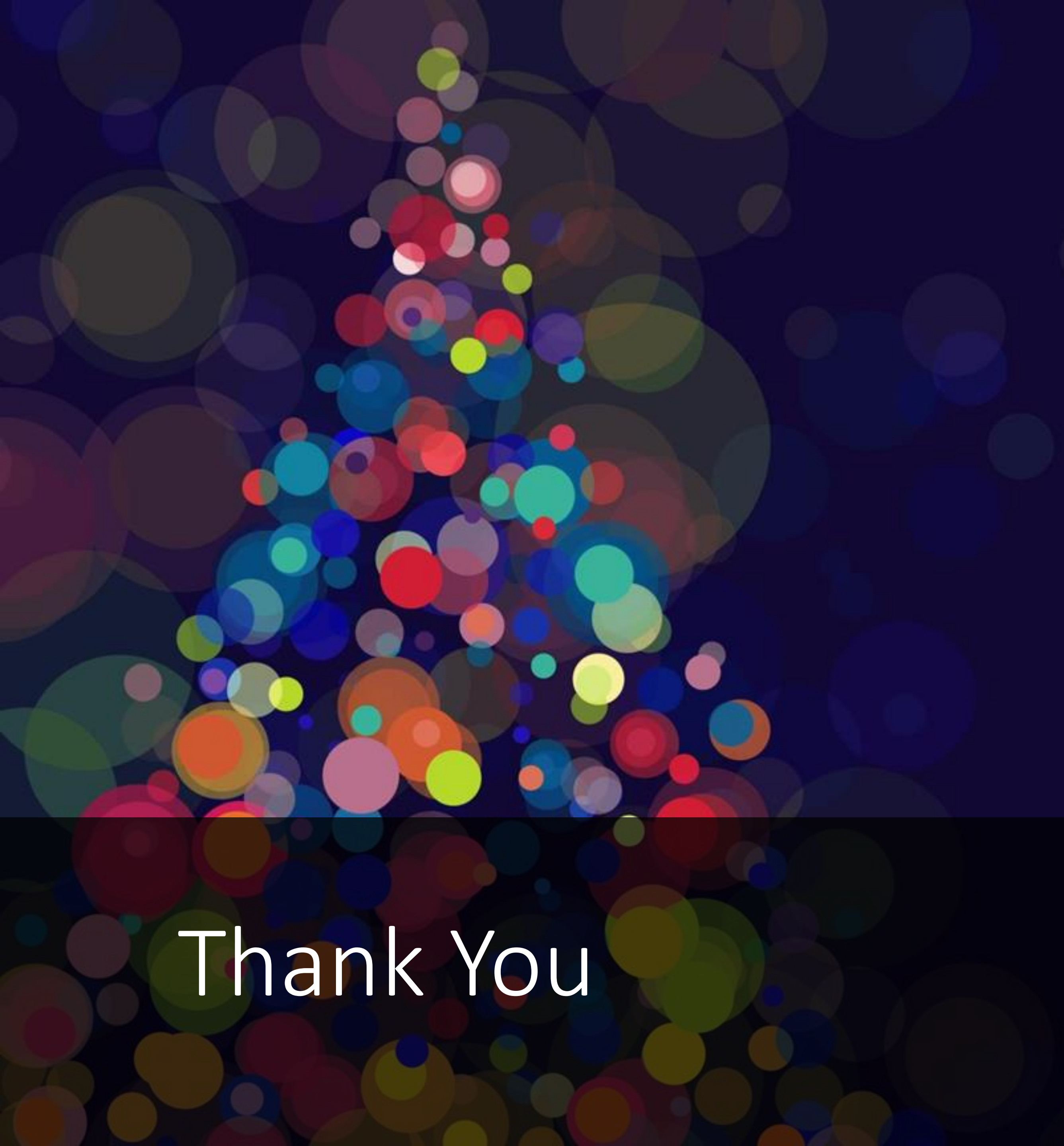
What if you need to deploy Azure resources?

- Azure Service Operator v2
- Crossplane



Lots more to learn

- Automated image updates
- Enabling Flux with Azure Policy
- Working with secrets
- Notifications
- Monitoring
- Using OCI artifacts
- ...



Thank You

and Happy Holidays!

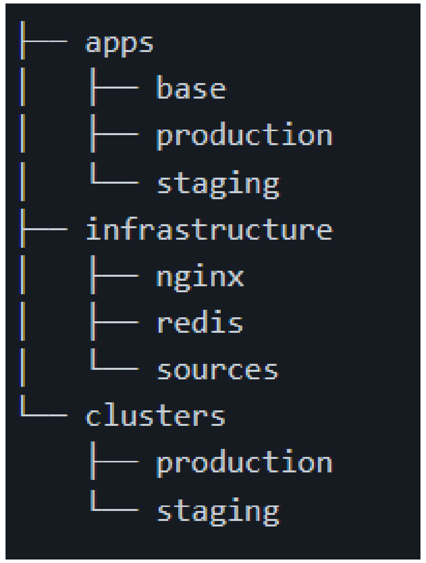
Additional Slides

OCI

- Package k8s configuration as OCI artifacts
  - those artifacts are stored in a container registry such as ACR
- When is this useful?
  - when git does not contain the final k8s manifests and you are using cuelang or jsonnet or other generators; run generator in CI and push the artifacts
- Flux CLI supports this with:
  - flux push artifact, flux pull artifact etc...
- You need:
  - Resource of **kind** OCIRepository
  - **Kustomization** that has sourceRef to the OCIReposiroty and specifies the path
- **Note:** also supported for Helm → create HelmRepository with url of oci://... and then have a HelmRelease with that HelmRepository in sourceRef

Multi-cluster with Flux only

- E.g. two clusters (staging and production) and use Flux & Kustomize to manage both **while minimizing duplications**
- Infrastructure & apps similar to our demo
  - apps: base + 2 overlays
  - infrastructure: same for both clusters here
- Both clusters bootstrapped to different folders in the same repo
  - each folder has its own Kustomizations
  - installation of the app by pointing to the correct overlay
- Note that the redis password is set via a SOPS encrypted secret stored in the repo; Kustomizations can have a decryption field to indicate the secret to decrypt with a key previously saved to flux-system (see https://github.com/fluxcd/flux2-kustomize-helm-example)



```
├── apps
│   ├── base
│   ├── production
│   └── staging
├── infrastructure
│   ├── nginx
│   ├── redis
│   └── sources
└── clusters
    ├── production
    └── staging
```

Multi-cluster with Microsoft FluxConfig

- One repo for YAMLs with overlays
- There are no **bootstrap** folders per cluster
- Create a GitOps configuration for each cluster that has the Kustomizations you require:
    - staging: infra + apps with staging overlay
    - production: infra + apps wth production overlay
- Basically, the GitOps Configurations are a replacement for bootstrapping
    - You don't do Flux bootstrapping here

Validating manifests & Kustomize overlays

- Use kubeconform (https://github.com/yannh/kubeconform)
  - Workflow https://github.com/fluxcd/flux2-kustomize-helm-example/blob/main/.github/workflows/test.yaml contains an example
- E2E test with a kind cluster
  - See https://github.com/fluxcd/flux2-kustomize-helm-example/blob/main/.github/workflows/e2e.yaml
  - Sets up kind with engineerd/setup-kind@v0.5.0
  - Installs Flux in kind
  - Creates Flux source and kustomization for staging cluster folder
  - Waits for kustomizations
  - Waits for Helm reconciliation
  - Debugs the failures (if any) by dumping logs

End to End

- Kustomize controller
  - validates manifests against k8s api
  - multi-tenant safety via Kubernetes service account impersonation
  - health assessment of deployed resources
  - dependency ordering
  - pruning
  - it can target remote clusters as well (remote does not need Flux but can have it installed)
- Helm controller
  - source controller acquires the Helm charts
  - Go client implementation of the Helm package library; full compatibility with all released Helm features
  - Support for Helm chart hooks, health checking

End to End

- Image Reflector
  - Scans image repositories
  - Specify how to scan OCI image repos like authentication etc...
- Image Automation Controller
  - Updates YAML files based on the latest images scanned by the reflector
  - It commits those updates to git
  - Custom resource: ImageUpdateAutomation
    - define how to do automated commits
  - ImagePolicy to define what image tags go where
  - You need to mark fields in your YAML files