GitOps Manager CI/CD

Build and deploy globally across clouds

WHAT YOU GET

Three promises, each kept by a small, readable action

Build once

Every image in a repo built with Docker in one step, signed in with OIDC — no stored registry keys.

Deploy through Git

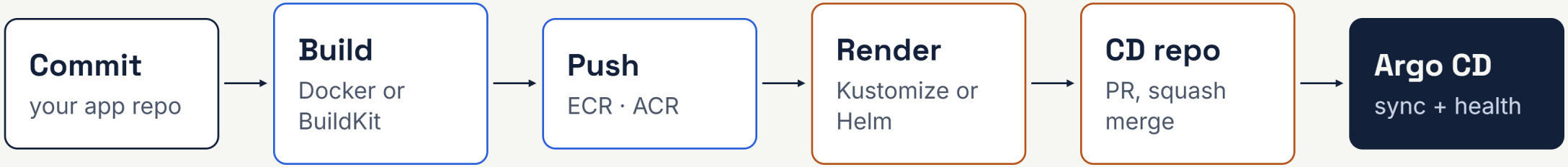
The exact manifests that run are committed to a CD repo. A rollback is a revert.

Know it’s healthy

The pipeline waits for Argo CD to report Synced and Healthy — and says why when it isn’t.

THE FLOW

Six steps from commit to a healthy app



BUILD

`multicloud-build-action`

Docker on a GitHub runner · or Bake on BuildKit, self-hosted

DEPLOY

`k8s-deploy · Auto-Commit-Squash-Merge · ArgoCD-Manage-Applications`

BUILD · MULTICLOUD-BUILD-ACTION

One step builds every image in the repo

Every image, one step

List your images once; Docker builds them together on GitHub’s runners, each with its own Dockerfile and context.

No stored registry keys

GitHub OIDC signs in to Amazon ECR and Azure ACR. Missing ECR repositories are created for you.

A cache that outlives the runner

The layer cache is kept in your registry, so the next build reuses what hasn’t changed.

Traceable tags

Every image is tagged with the GitHub run that built it, plus any tag you choose.

BUILD · WHERE IT RUNS

One action, two places to run it

`runs-on: ubuntu-latest`

GitHub-hosted, Docker

- Docker on GitHub’s runner, set up by the action
- Same images, same OIDC sign-in
- Cache pulled from your registry each run
- Free to start, nothing to install

`runs-on: your self-hosted runners`

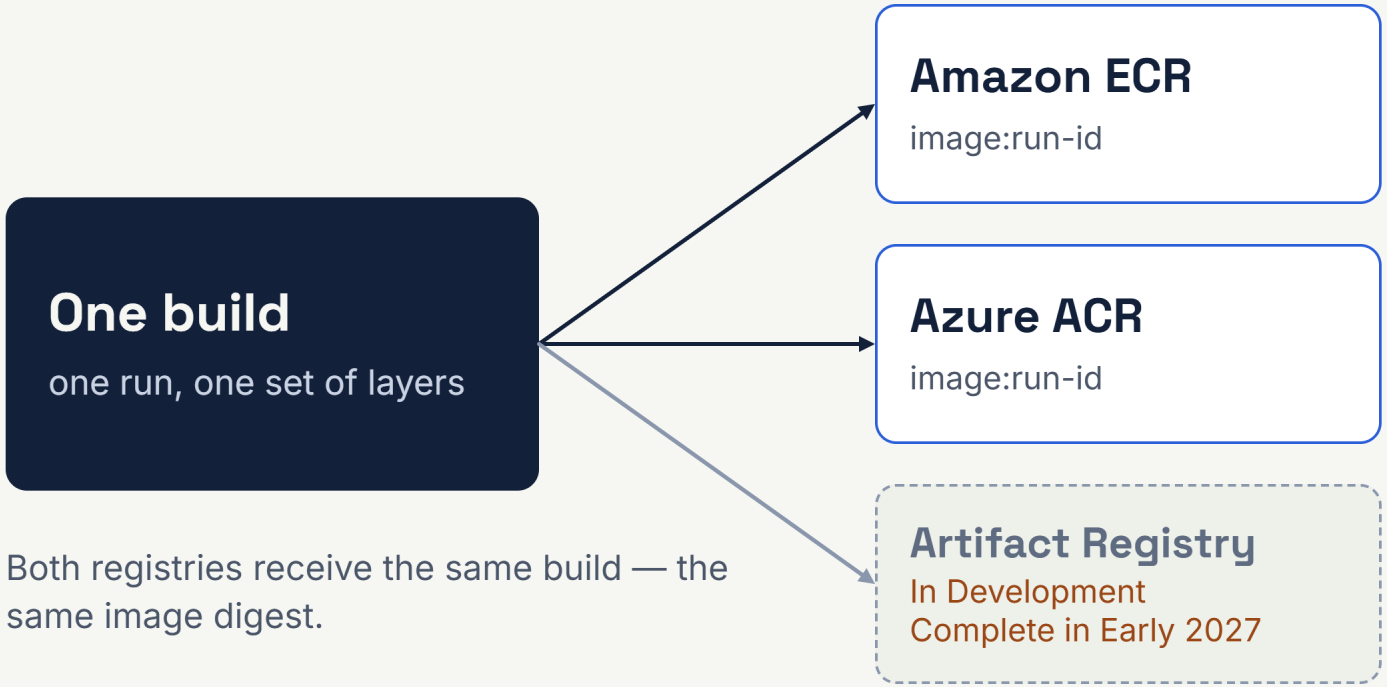
Self-hosted, Bake on BuildKit

- A dedicated BuildKit engine beside each runner
- Pushes with your cluster’s workload identity
- Source and images stay in your cloud
- No hosted build minutes

The action picks the engine itself — the workflow doesn’t change, only **runs-on**.

BUILD · ONE CLOUD OR BOTH

Build in one cloud. Push to both if you need to.



Both registries receive the same build — the same image digest.

DEFAULT

Push to the registry in the cloud your cluster runs in. Fastest, no cross-cloud traffic.

OPTION

One setting pushes to both — for a second region, DR or a migration.

THE COST

Egress for the second push, and OIDC trust in the second cloud.

DEPLOY · RENDER

Render the manifests for one cluster — your choice of tool

`k8s-deploy`

Kustomize + templates

- Ingress, namespace and identities from agreed variables — no chart
- Image set to the tag the build pushed
- Output validated with kustomize build

`k8s-helm-deploy`

Helm, as a renderer only

- Values files per environment, then per cluster
- Nothing is helm-installed — no hidden release state
- Secrets carrying data are refused

DEPLOY · INGRESS, TWO OPTIONS

Option 1 · Kustomize with k8s-deploy

One ingress file works on every cluster

ingress.yaml

every environment, every cluster · no chart, no values files

```
rules:
  - host: myapp.{{ dns_zone }}
  - host: {{ cluster_name }}-myapp.{{ dns_zone }}
```

Filled in per cluster from the env map (next slide)

Variable	Value
{{ cluster_name }}	the chosen cluster
{{ cloud }}	aws or azure
{{ dns_zone }}	that cluster's zone
{{ namespace }}	the deploy's namespace
{{ app_uami }} ...	identity client IDs

RENDERED INTO THE CD REPO

```
client-eks-prod → myapp.prod.example.com, client-eks-prod-myapp.prod.example.com
client-aks-prod → myapp.prod.example.com, client-aks-prod-myapp.prod.example.com
```

DEPLOY · INGRESS, TWO OPTIONS

Option 1 · Kustomize with k8s-deploy

The env map fills in the variables for each cluster

```
env map · orange = feeds a variable
"prod": { "clusters": [
  { "cluster": "client-eks-prod", "cloud": "aws",
    "dns_zone": "prod.example.com",
    "container_registry": "123456789012.dkr.ecr.eu-west-1.amazonaws.com" },
  { "cluster": "client-aks-prod", "cloud": "azure",
    "dns_zone": "prod.example.com",
    "container_registry": "clientprod.azurecr.io" },
  { "cluster": "client-gke-prod", "cloud": "gcp", // early 2027
    "dns_zone": "prod.example.com",
    "container_registry": "europe-west1-docker.pkg.dev/client-prod/apps" }
] }
```

One cluster per deploy

1. A named cluster wins
2. Else a named cloud narrows it
3. Else the runner's cloud breaks a tie
4. Still two? It stops and says so

Open source: you write the map.

Service: every new cluster joins its environment for you.

DEPLOY · INGRESS, TWO OPTIONS

Option 2 · Helm with k8s-helm-deploy

Values files say it per environment, then per cluster

1 · values.yaml

every environment

```
ingress:
  enabled: false
```

2 · values-prod.yaml

every prod cluster

```
ingress:
  enabled: true
  hosts:
    - host: myapp.prod.example.com
```

3 · values-prod-client-eks-prod.yaml

client-eks-prod only · replaces the list, so repeats the shared host

```
ingress:
  hosts:
    - host: myapp.prod.example.com
    - host: client-eks-prod-myapp.prod.example.com
```

3 · values-prod-client-aks-prod.yaml

client-aks-prod only · the same again for this cluster

```
ingress:
  hosts:
    - host: myapp.prod.example.com
    - host: client-aks-prod-myapp.prod.example.com
```

DEPLOY · THE CD REPO

Git is the record of what runs

Rendered manifests land in a CD repo under **cluster / namespace / app**, through a pull request that is squash-merged automatically.

Rollback: each deploy is one commit. Revert it and sync in Argo CD — or redeploy the previous image tag, which commits, syncs and waits for Healthy in one run.

A lost cluster is rebuilt from the repo with **k8s-restore-cluster**.

THE CD REPO

```
client-eks-prod/
  shop/
    myapp/
      deployment.yaml
      ingress.yaml
client-aks-prod/
  shop/
    myapp/
      …
```

Fully rendered: no placeholders and no values files. Each cluster's folder is exactly what Argo CD applies there.

DEPLOY · ARGO CD

Green means healthy, not just committed

1

Create

The Argo CD application is made if it doesn’t exist, pointed at its folder in the CD repo.

2

Sync

A sync with prune is started, so removed resources leave the cluster too.

3

Wait

The job waits for Synced and Healthy. A failure shows Argo CD’s own reason in the log.

4

Report

A summary table on the run shows each app’s result at a glance.

OPEN SOURCE · MIT LICENCE · GITHUB.COM/GITOPSMANAGER

The actions, one job each

Action	What it does
multicloud-build-action	Docker build and push to ECR, ACR or both
detect-cloud	Tells a job which cloud its runner is in
k8s-load-env	Picks the target cluster from the environment map
k8s-deploy	Renders Kustomize manifests into the CD repo
k8s-helm-deploy	The same, rendering Helm charts
Auto-Commit-Squash-Merge	Writes to a repo by pull request and squash merge
ArgoCD-Manage-Applications	Creates, syncs and health-checks Argo CD apps
k8s-restore-cluster	Rebuilds every app on a cluster from the CD repo

GITOPS MANAGER SERVICE

The service runs the pipeline inside your own cluster

Self-hosted runners, managed

Build and deploy runners that scale in your cluster. Your code never leaves your cloud.

Docker Bake on BuildKit

Each build runner has its own BuildKit engine beside it — no Docker daemon to share.

Cloud-native identity

Runners push with your cluster's own workload identity on AWS and Azure.

Every environment, mapped

Clusters worldwide, registries and private or public DNS zones for dev, qa, uat and prod — with wildcard certificates kept renewed.

SELF-HOSTED RUNNERS

Two kinds of runner, each sized for its job

Build runners

Heavier · fewer at once

- Docker Bake on a dedicated BuildKit engine
- Room for compiling, layers and cache
- Builds every image in a repo in parallel
- Pushes with your cluster's own identity

Deploy runners

Lighter · more at once

- Render, commit to Git, sync Argo CD
- Small footprint, quick to start
- Many deploys run side by side
- No build engine to carry

Both run in your own cluster and scale with the queue — your code, images and credentials stay in your cloud.

Yours to keep: cancel and the service shuts down and cluster-monitor is removed — your clusters, tool stack, runners and IaC workflows stay, for internal use only.

Start with two steps

1 · Build

Add multicloud-build-action to your app repo's workflow.

2 · Deploy

Call k8s-deploy or k8s-helm-deploy with your CD repo and Argo CD.

Then · Scale

Move to managed runners in your cluster when you're ready.

Brian Saltzman | Kubernetes Platform & DevOps Engineer

contact@gitopsmanager.io

gitopsmanager.io | eksmanager.io | aksmanager.io | gkemanager.io | GitHub | LinkedIn | AWS Marketplace Listing