ARGOCON NA 2026 · SALT LAKE CITY

Minimizing Deployment Blast-Radius With Argo CD Progressive Sync



Alexandre Gaudreault
Staff Software Developer & Argo Maintainer @ Intuit



Kanika Rana
Senior Software Engineer & Argo CD Maintainer @ Red Hat

AGENDA

What We'll Cover

- The Problem: How Deployments Grow Up
- Progressive Sync: How It Works
- Limitations
- What's Next

THE PROBLEM · CHAPTER 1

The Proof of Concept: You Are Happy

One Application syncs to one Kubernetes cluster. This is Argo CD's core use case.



Git commit



Argo CD Application

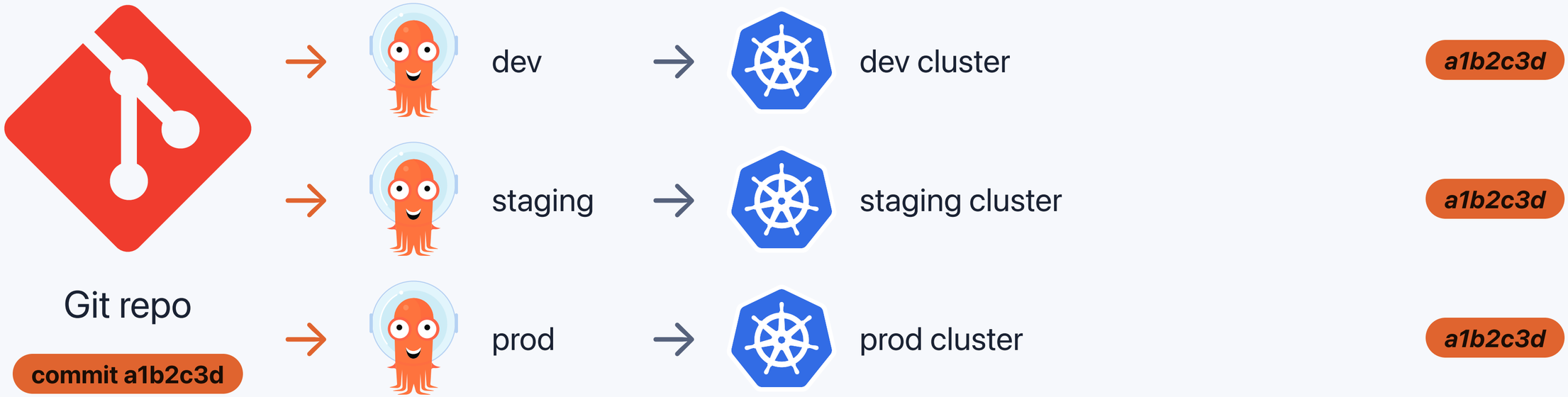


Kubernetes cluster

THE PROBLEM · CHAPTER 2

Adoption: Dev → Staging → Prod

The natural next step: one Application per environment. But all three track **the same Git commit**, and Argo CD syncs them all at once.

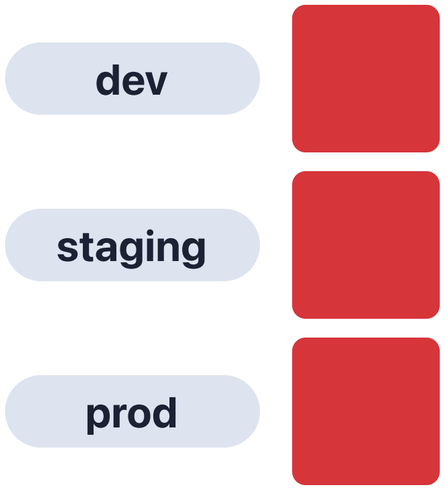


Same commit, every environment, at the same time. A bad commit reaches prod as fast as dev.

Auto-Sync vs Progressive Sync

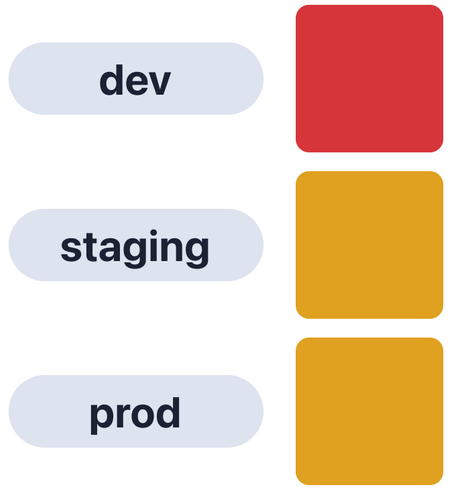
Auto-Sync

Every environment syncs the new commit at the same time.



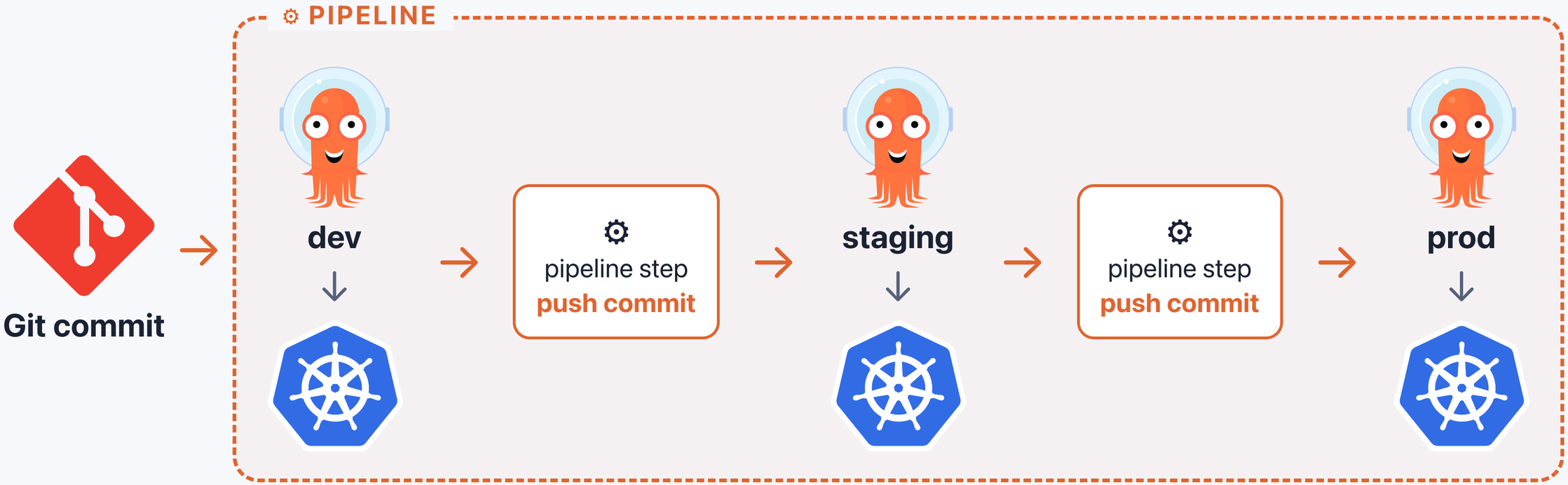
Progressive Sync

Environments sync one at a time, in order.



A Progressive Deployment Pipeline

A pipeline step sits between the Applications and **pushes the commit** to the next environment only when the previous one succeeded.



Each environment is a gate: a bad commit is caught in dev or staging **before** it reaches prod.

First Prod Outage: The New Version Is Bad

Readiness gates are passing, but the application is failing. Even with multiple environments, the new code can still break production.

Outage

Deployment Rolls Out to 100% of Pods



Blast radius: 100% of pods, no rollback.

Progressive Delivery With Argo Rollouts

Canary at the workload level: shift traffic gradually, analyze metrics, auto-abort on failure.

Fix Canary 10% → 50% → 100%

Analysis fails at the first step, so the rollout is aborted.



Blast radius: 10% of pods, then rollback.

THE PROBLEM · CHAPTER 4

Second Prod Outage: It Wasn't the Pods

Sometimes the culprit isn't application code: an Ingress, a Service, a CRD, a ConfigMap.



Git commit





prod





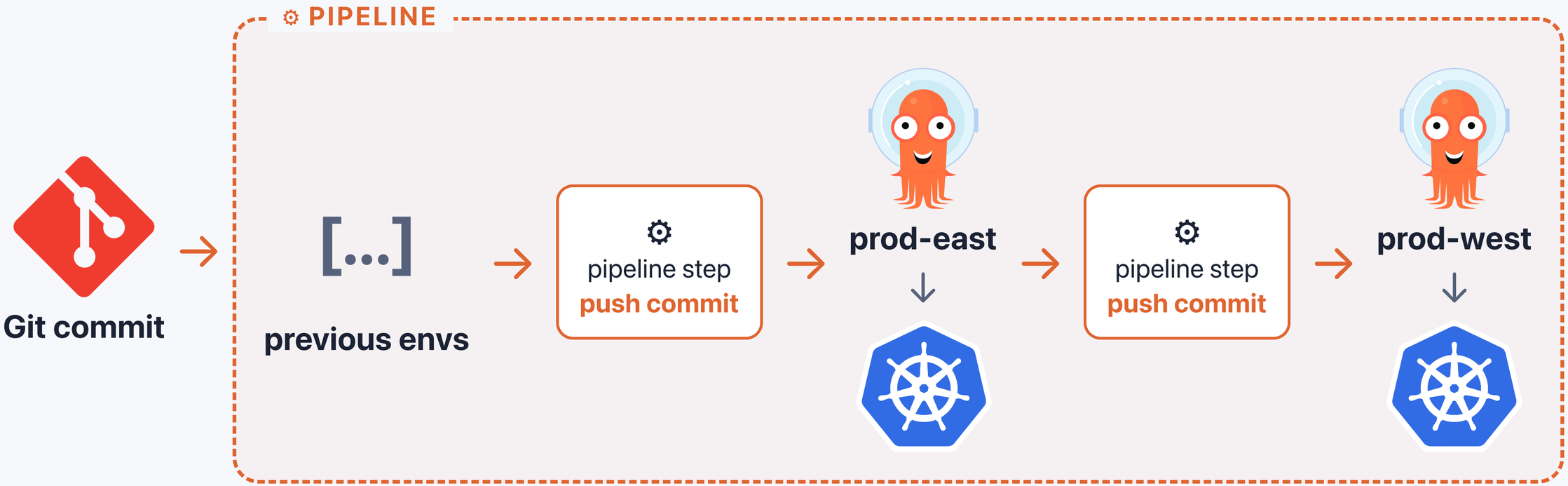
prod cluster

Ingress broke 💥

full outage

Failover Strategy: Sync Prod Sequentially

Two prod clusters. The second one only gets the commit after the first one succeeded.



> A bad Ingress, Service or CRD change takes down **one** region, not both.

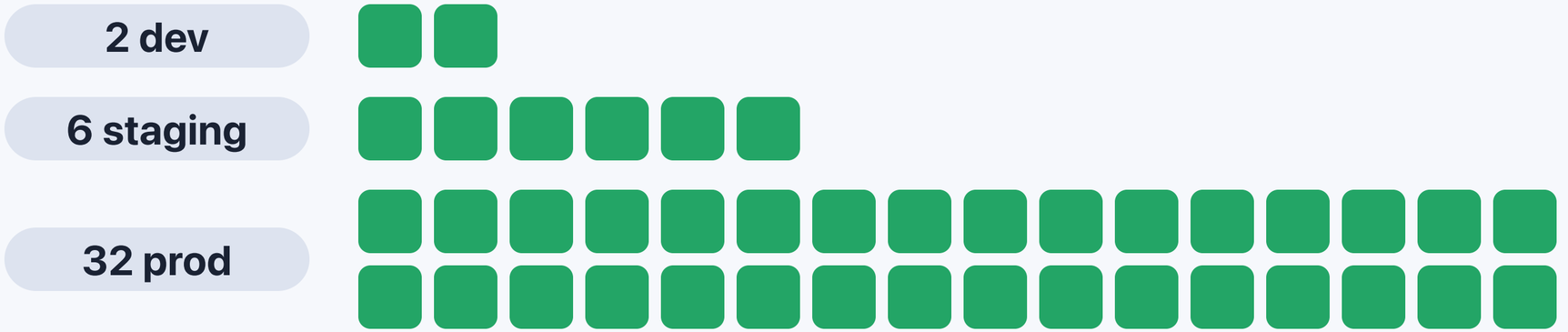
THE PROBLEM · CHAPTER 5

Scaling All Over the World

Multiple regions, edge locations, prod instances per tenant.

40

clusters



Sequential

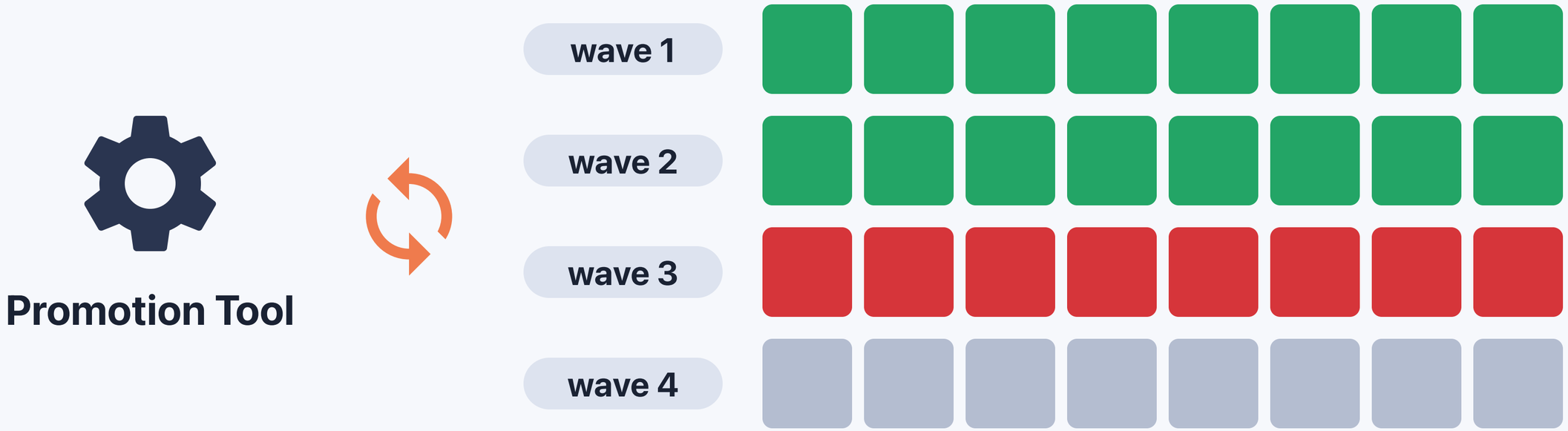
40 steps. Hours or days of lead time.

Parallel

One bad change hits all 32 prod clusters.

Deploy in Waves

A good time to adopt a promotion tool such as **Kargo** or **GitOps Promoter**: the complexity of a custom pipeline keeps growing. 32 prod clusters become 4 waves of 8.

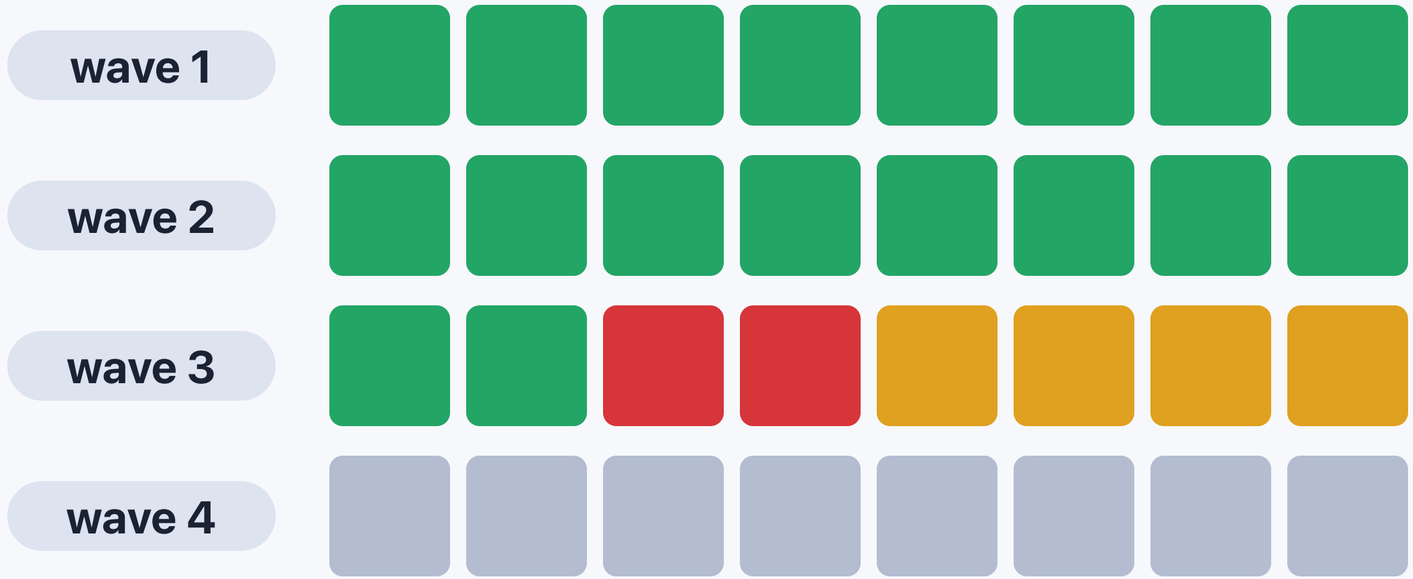


Progressive Sync Within Waves

What do we do inside a wave? **Progressive Sync**: sync the clusters one at a time, and only move on when each one is healthy.

Promotion Tool
+
Progressive Sync

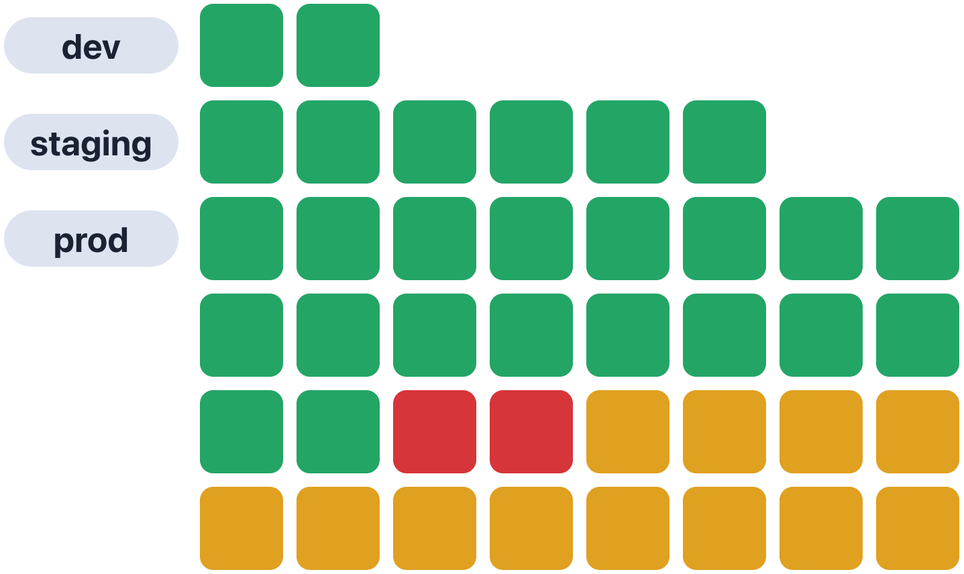




Progressive Sync and Promotion Tools

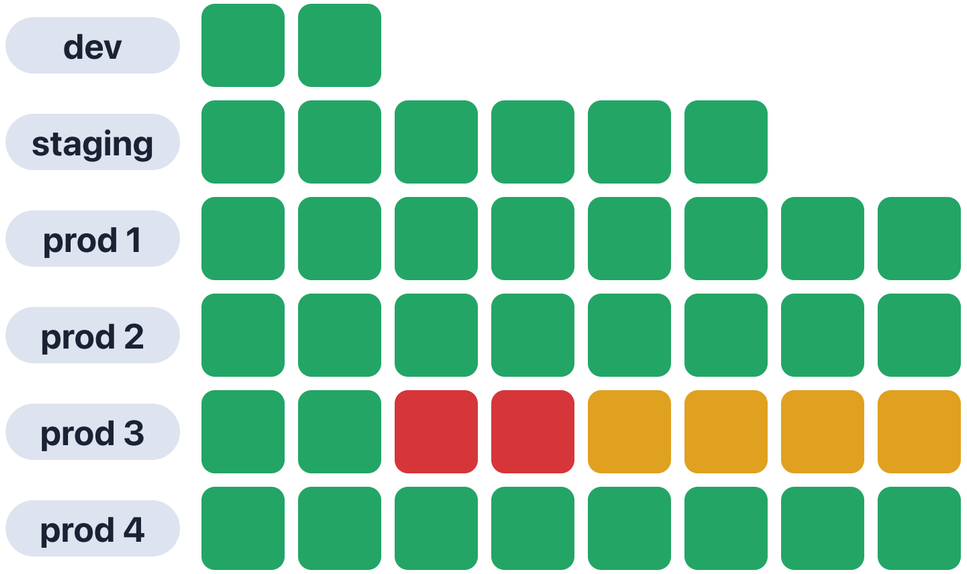
Without Promotion Tool

All apps are out of sync at once. The controller works through the batches slowly.



With Promotion Tool

Only the **current environment** is out of sync. The rest stay on the old revision until promoted.



Progressive Sync
How It Works

HOW IT WORKS

Wait, Disable Auto-Sync?!

> It's OK. The progressive sync strategy **replaces** auto-sync. Both are automated processes; only one of them may drive syncs.

- Applications generated by ApplicationSet using Rolling Sync strategy have auto sync disabled
- The ApplicationSet controller decides **when** each Application syncs
- Feature is opt-in on the controller (Beta)

Generated Application

```yaml
kind: Application
spec:
  syncPolicy:
    # no "automated" block
    # = auto-sync disabled
    retry:
      limit: 2
```

HOW IT WORKS

Turn It On

Step 1

Enable It on the Controller

```
# argocd-cmd-params-cm
data:
  applicationsetcontroller.enable.progressive.syncs: "true"
```

Step 2

Set the Strategy in the ApplicationSet

```
kind: ApplicationSet
spec:
  strategy:
    type: RollingSync
```

Docs: argo-cd.readthedocs.io/en/stable/operator-manual/applicationset/Progressive-Syncs

GROUPING APPS

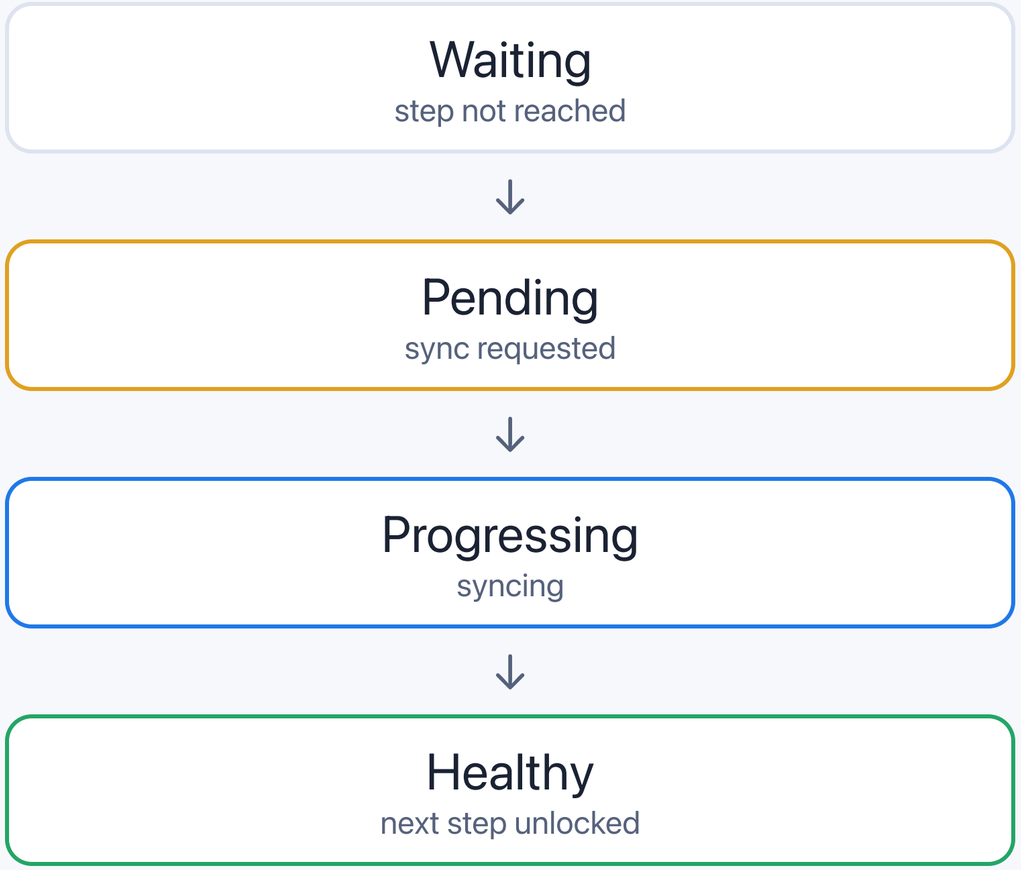
Steps Select Applications by Label

```yaml
kind: ApplicationSet
spec:
  strategy:
    type: RollingSync
    rollingSync:
      steps:
        - matchExpressions:
            - {key: wave, operator: In, values: [dev]}
        - matchExpressions:
            - {key: wave, operator: In, values: [staging]}
          maxUpdate: 1
        - matchExpressions:
            - {key: wave, operator: In, values: [prod]}
          maxUpdate: 25%
  template:
    metadata:
      labels:
        wave: '{{.wave}}'
```

- Label each generated App with its wave
- Steps run **in order**
- **maxUpdate** caps apps syncing at once inside a step (number or %)
- Next step starts when the previous one is **Healthy**

HOW IT TRIGGERS SYNC

The AppSet Drives Each Application's Status



`kubectl get applicationset guestbook -o yaml`

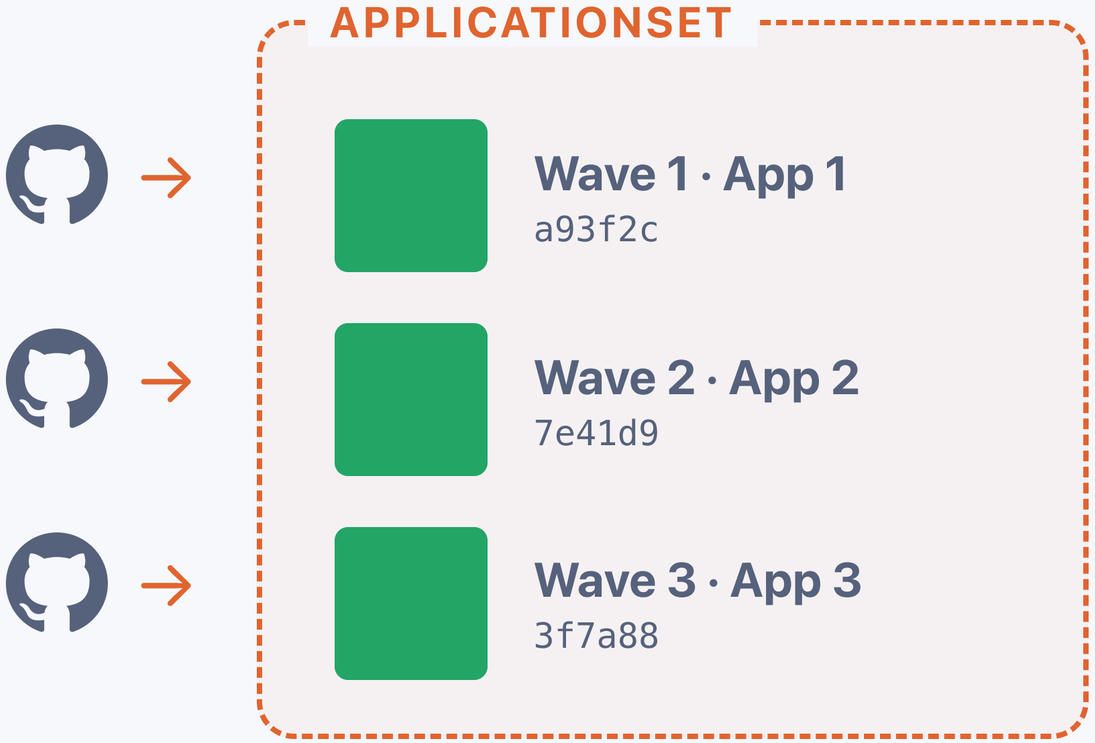
```yaml
status:
  applicationStatus:
    - application: guestbook-dev
      status: Healthy
      step: "1"
      targetRevisions:
      - 1c42dc7
    - application: guestbook-staging
      status: Progressing
      step: "2"
      targetRevisions:
      - 1c42dc7
    - application: guestbook-prod
      status: Waiting
      step: "3"
      targetRevisions:
      - 1c42dc7
```

COMPLEXITY

How Do You Detect “A Change”?

Is a change a new Git revision? **Nooo.** It's more complicated.

- Several generators/sources can contribute to one Application
- The controller must tell "needs a rollout" from "just refresh"
- A **refresh** is triggered as soon as we detect an Application may have a new revision



More info: github.com/argoproj/argo-cd/pull/27467

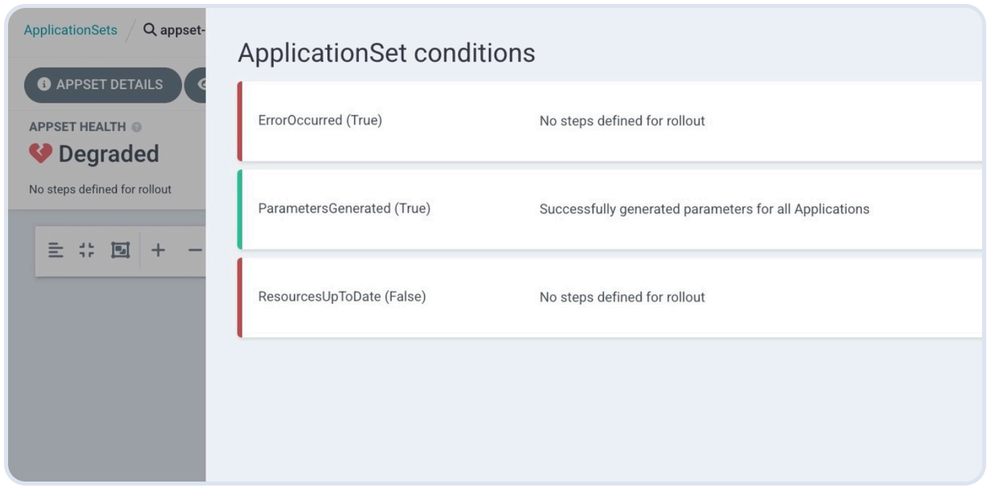
COMPLEXITY

What Went Wrong?

- Invalid rollout config surfaces as an **ApplicationSet condition**

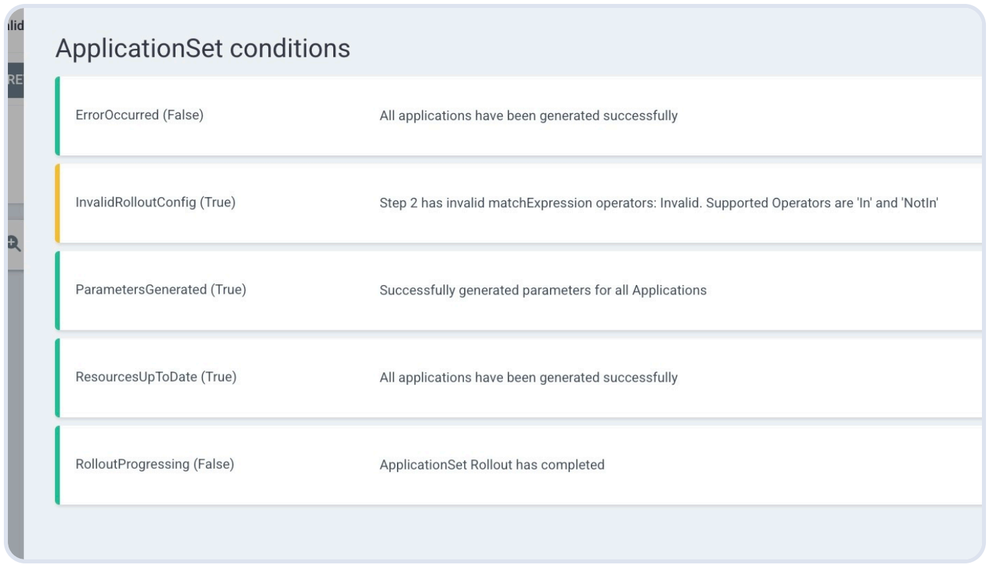
Error

Blocks the rollout until the config is fixed.



Warning

Surfaced for visibility; the rollout continues.



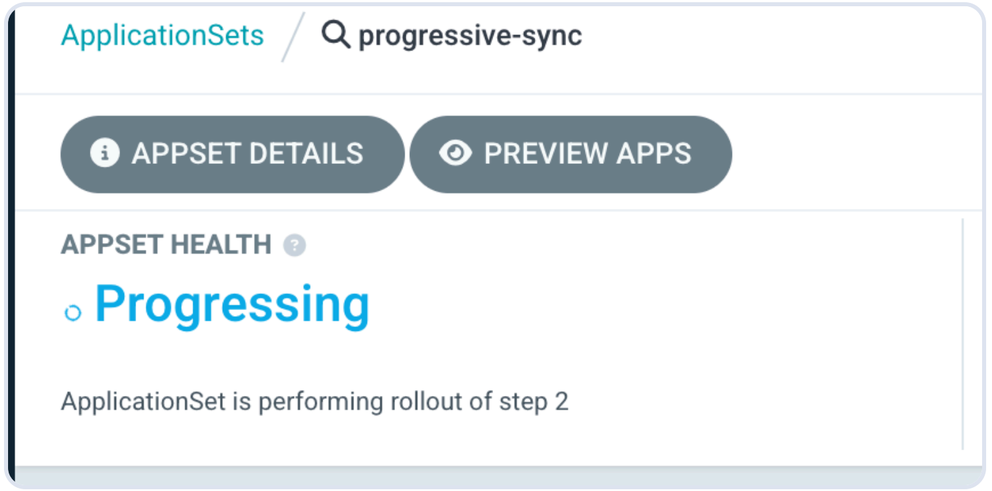
More info: github.com/argoproj/argo-cd/pull/27820

APPSET UI INTEGRATION

Deployment Progress, in Real Time

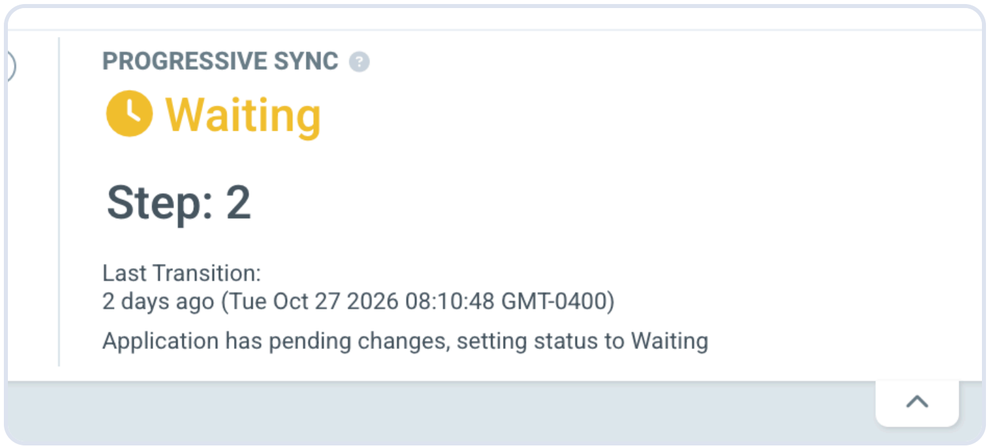
ApplicationSet Health

Overall rollout status for the ApplicationSet.



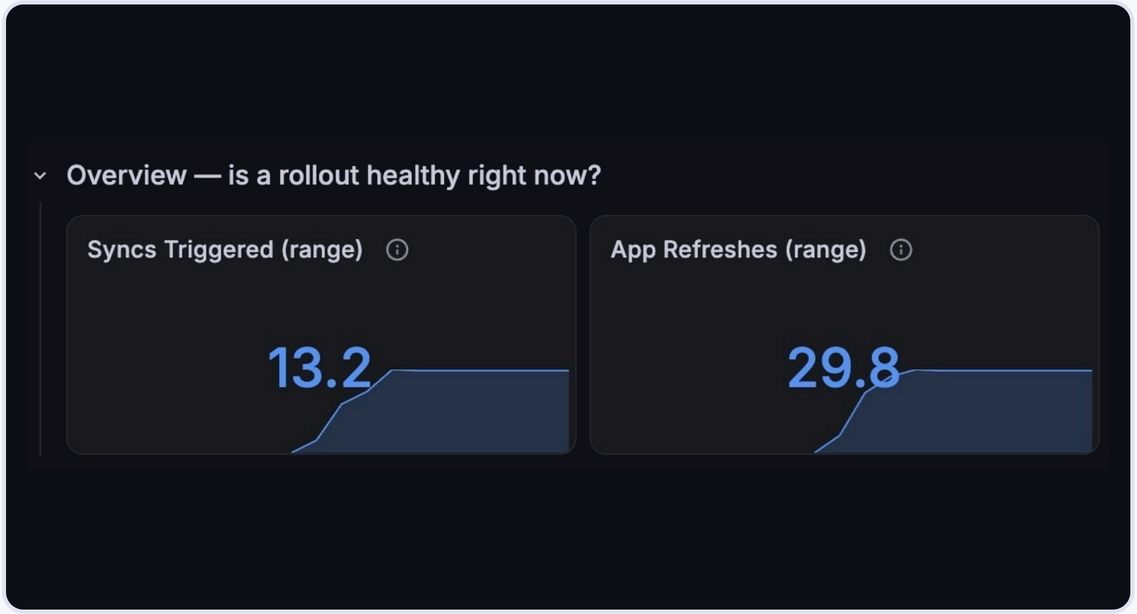
Application Progressive Sync

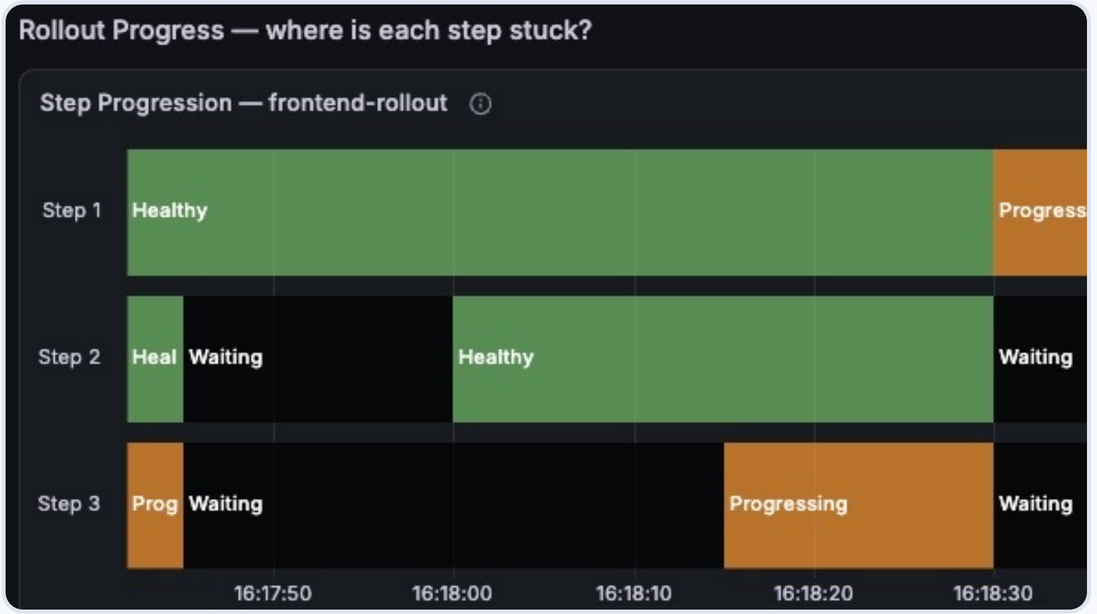
Current step and status for each Application.

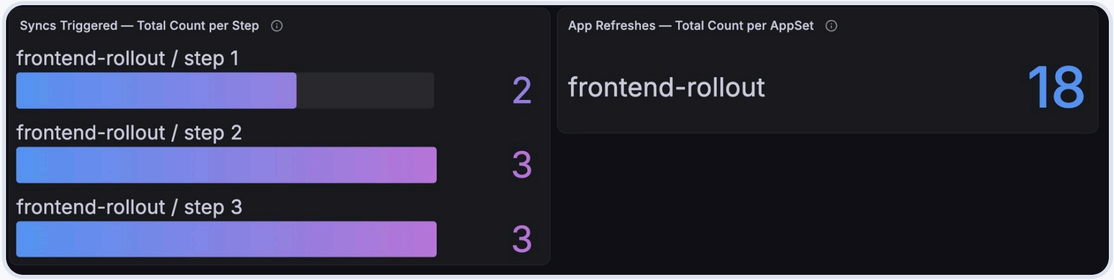


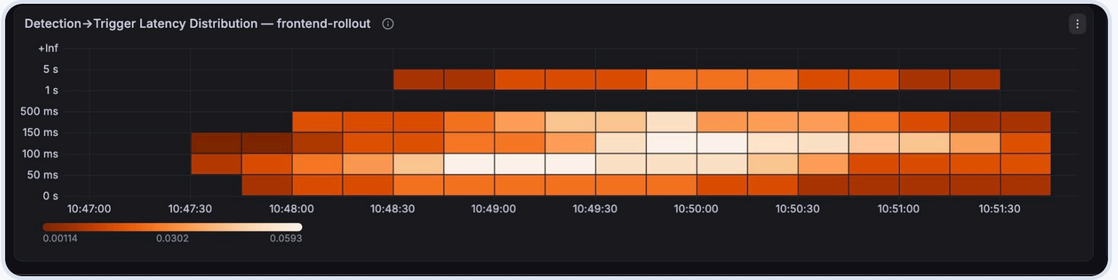
OBSERVABILITY

New Metrics for Progressive Sync









LIMITATIONS

What to Know Before Adopting

Coupled to ApplicationSet

Apps not generated by an ApplicationSet can't be progressively synced.

1 AppSet ≈ 1 Microservice

Assumed model. Not ideal when one ApplicationSet deploys different microservices that should roll independently.

What's Next

WHAT'S NEXT

Re-Architecture Under Discussion

Extract the functionality out of the ApplicationSet controller.

- Select by **app selector** so any Application can use it
- A **standalone CRD**
- UI plugin and experience **decoupled** from the AppSet UI

RECAP · IS IT STABLE?

Stable or Not Stable, That Is the Question

Kubernetes has navigated this before. Two options, two precedents.

Promote to Stable

- Promoted to stable (GA)
- API **frozen**: no new features
- Bug fix only
- Alternative built with new features

Strategy adopted for **Ingress**

Stay in Beta

- Never promoted to stable
- **Deprecated while in beta**
- Longer grace period than what is mandatory
- Alternative built in parallel

Strategy adopted for **PodSecurityPolicy**

RECAP

Progressive Delivery at Three Levels

LEVEL	TOOL	PROTECTS AGAINST
Workload	Argo Rollouts	Bad pods
Environment	GitOps Promoter · Kargo · custom pipelines	Bad revision reaching prod
Application	Argo CD Progressive Sync	Bad change hitting every cluster at once

Thank You
Questions?

Promote Progressive Sync to Stable · Issue #28927

Join the #argo-cd community!

