Effective Disaster Recovery

The Day We Deleted Production

influxdata®

InfluxDB

- The platform for building time series applications
- At the heart, an opensrc db purpose-built for time-stamped data
- Start from the UI or skip right to the raw code and API.
  - APIs and client libraries for 12 languages

Rick Spencer

VP of Product @ InfluxData

rickspencer3

Wojciech Kocjan

Platform Engineer @ InfluxData

@wojciechka

@wojciechka

influxdata®

Timeline

How to delete production in a few easy steps.

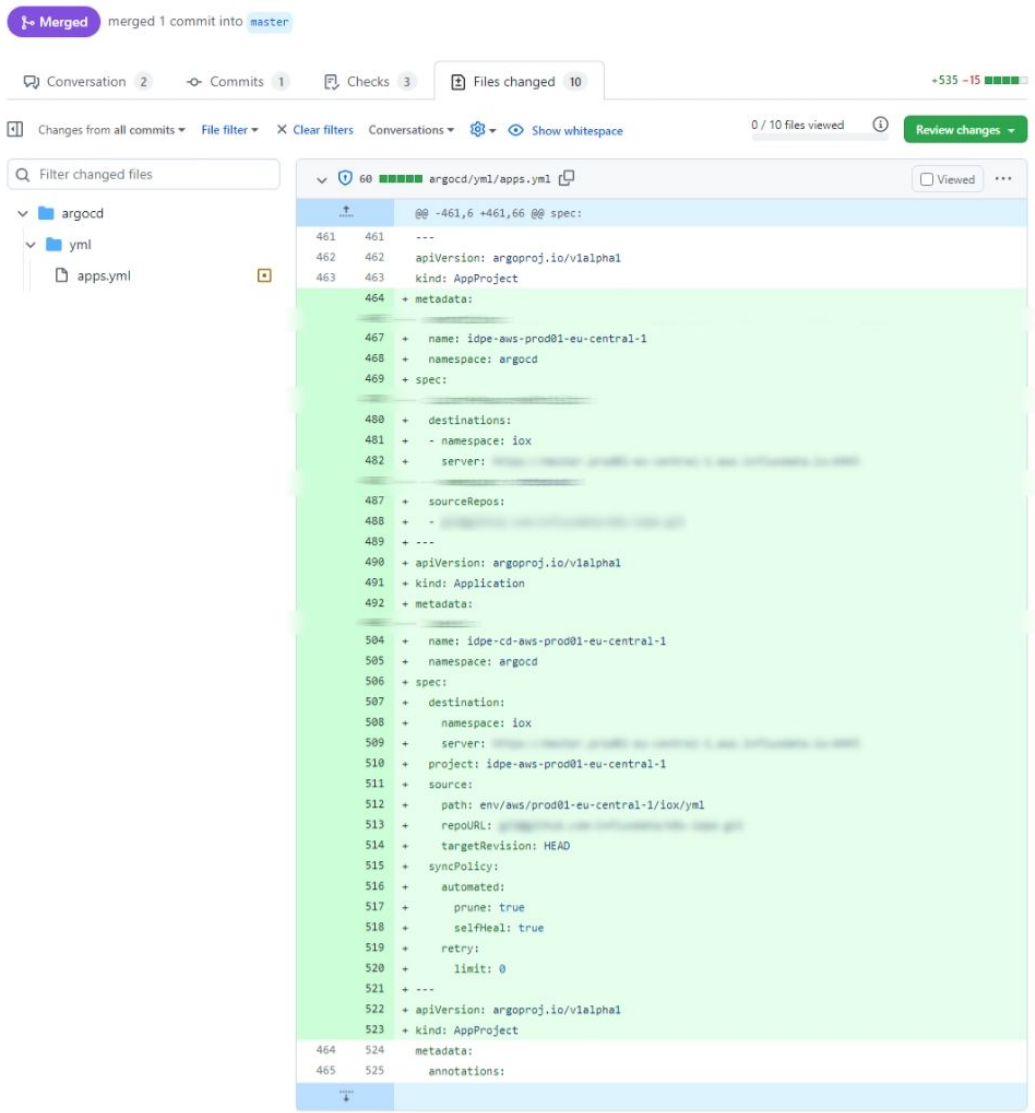
influxdata®

InfluxDB Cloud, state, GitOps & CI/CD

- Everything is in Git - separate code repository, config repository
  - CI for code builds images and commits images to code repo
- Kubernetes-based, partially stateful application
  - Storage engine with PVCs, object store for persistence
  - Kafka and Zookeper used for Write-Ahead Log (WAL)
  - Etcd (separate from K8s) used for metadata - buckets, other objects
  - Most PVCs configured to retain underlying volume, backups for others
  - Most microservices are stateless or use managed databases
- ArgoCD for deploying all instances of InfluxDB Cloud
  - Auto-sync and prune enabled everywhere by default
  - App of Apps pattern - ArgoCD apps managed by ArgoCD

How it started - a PR gets merged

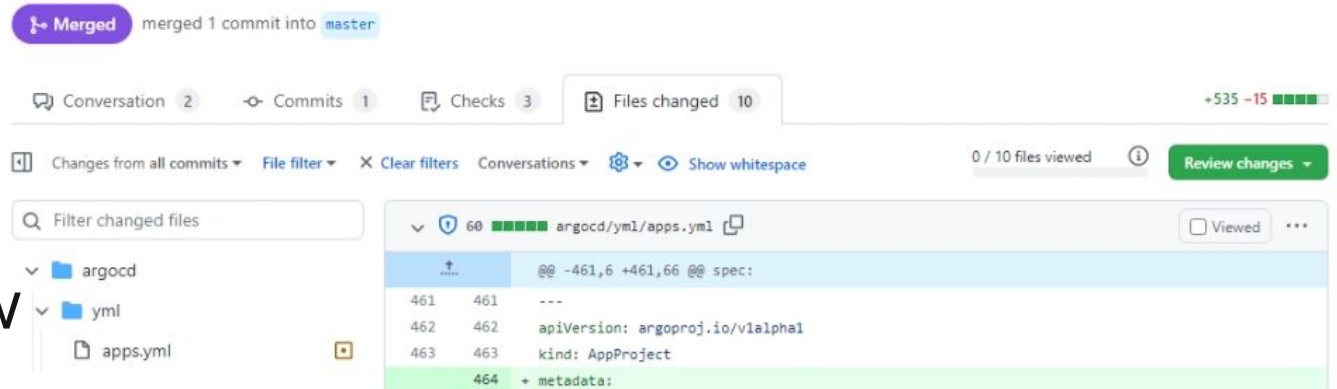
- This PR gets merged
- Only adds new YAML objects
- Does not remove anything
- Triggers an immediate deletion of a production environment



influxdata®

How it started - a PR gets merged

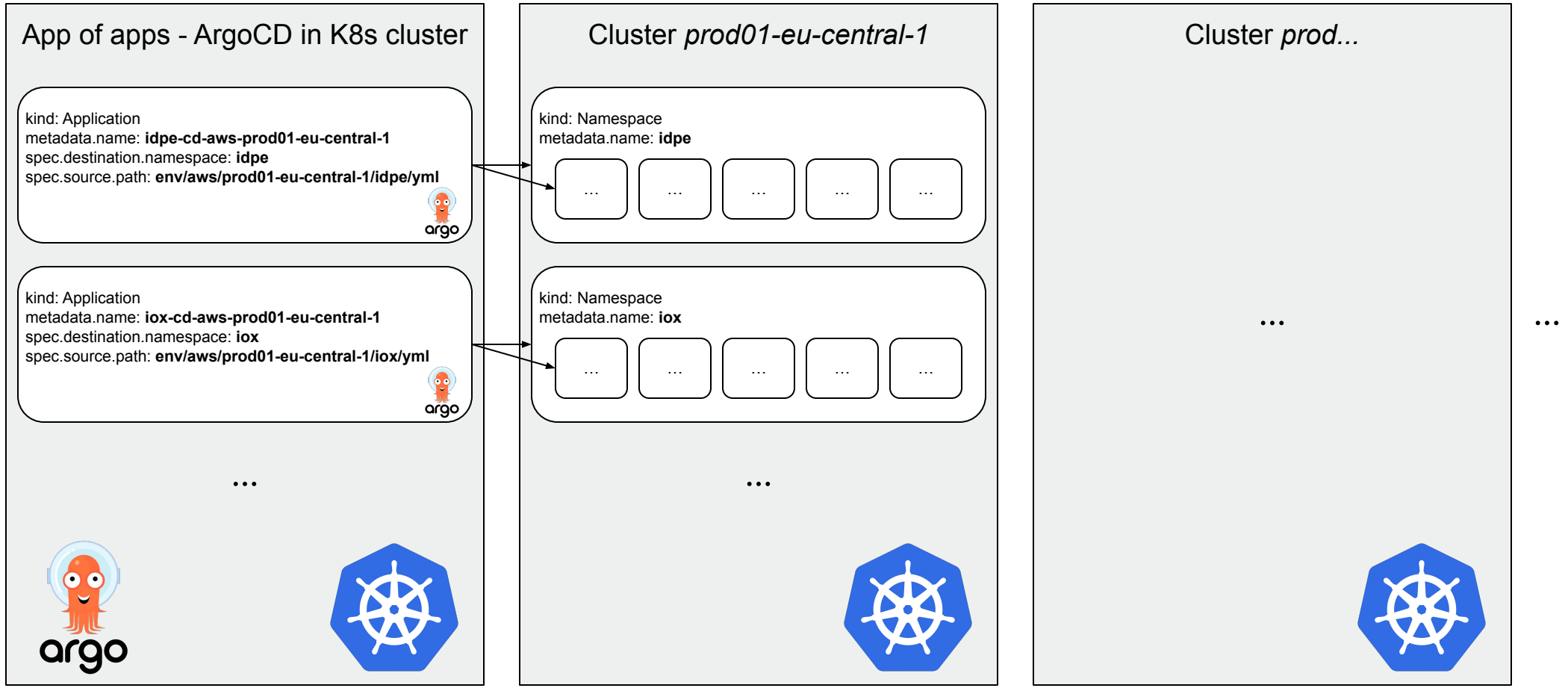
- ArgoCD app/project name collision
- Difficult to spot in review




```
431  431  apiVersion: argoproj.io/v1alpha1
432  432  kind: Application
433  433  metadata:
444  444    name: idpe-cd-aws-prod01-eu-central-1
445  445    namespace: argocd
446  446  spec:
447  447    destination:
448  448      namespace: idpe
449  449      server:
450  450    project: idpe-aws-prod01-eu-central-1
451  451    source:
452  452      path: env/aws/prod01-eu-central-1/idpe/yml
453  453      repoURL:
454  454      targetRevision: HEAD
455  455    syncPolicy:
456  456      automated:
457  457        prune: true
458  458        selfHeal: true
459  459      retry:
460  460        limit: 0
```

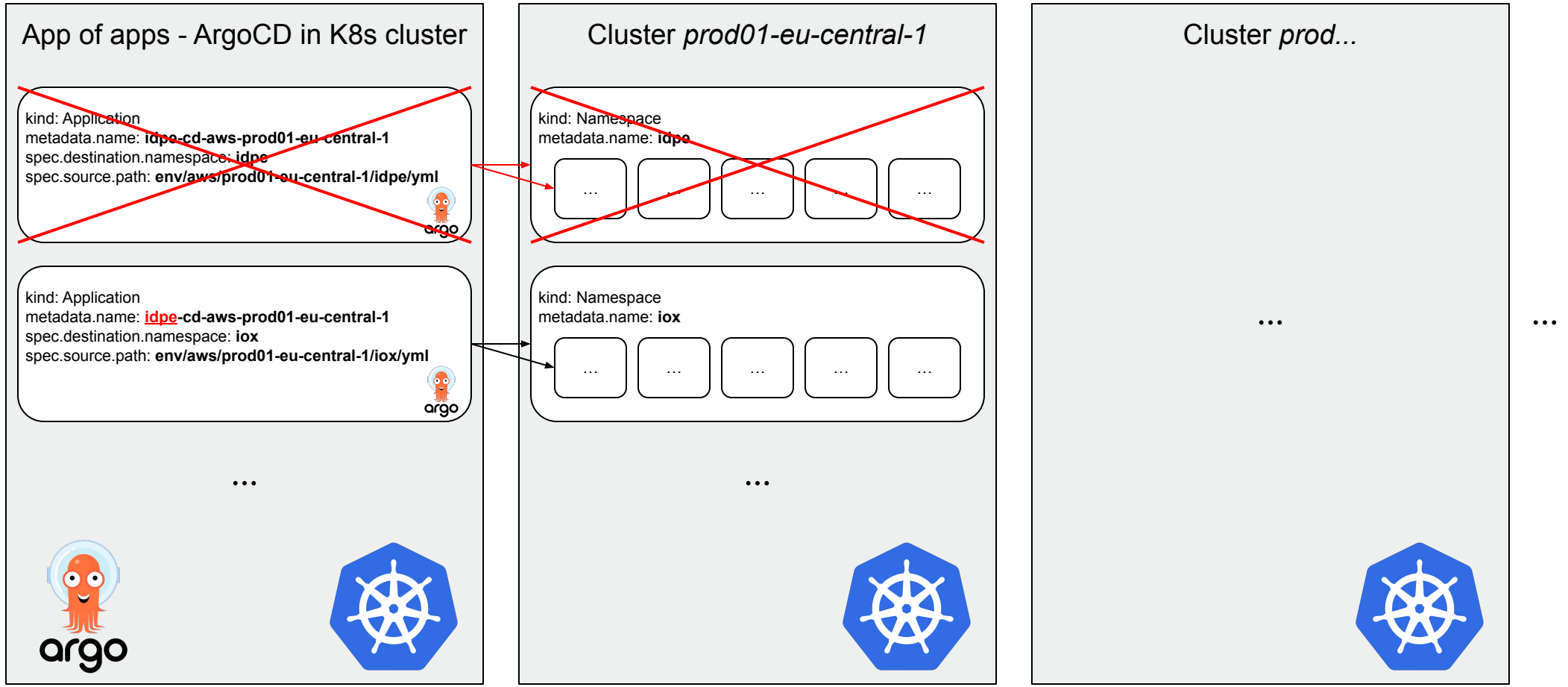

ata®

How ArgoCD works



influxdata®

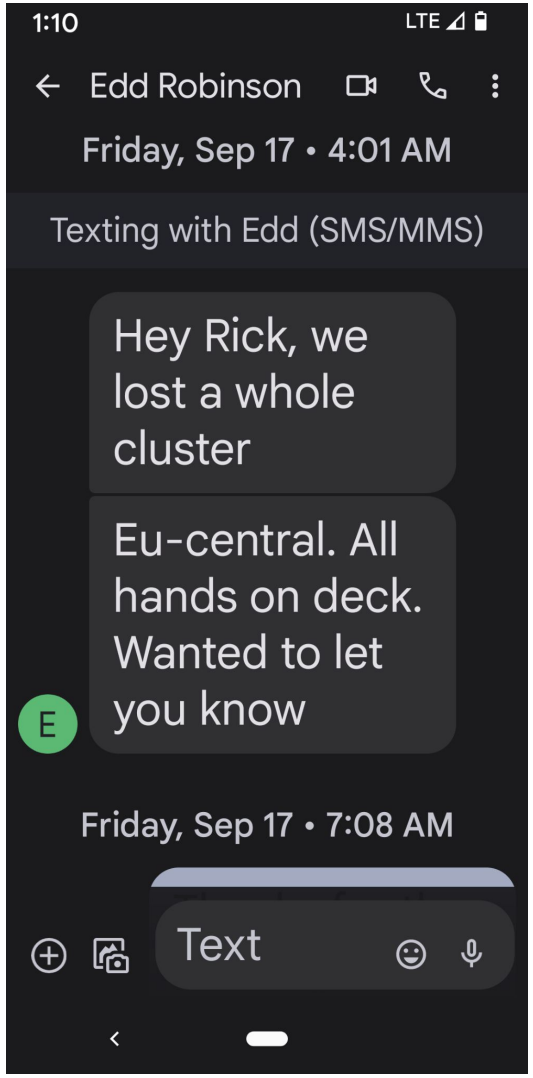
What ArgoCD did



 |

influxdata®

Good Morning!



influxdata®

Phase 1: The Damage is Done

10:22 UTC: The PR was merged

10:25 UTC: InfluxData monitoring started reporting API failures

10:37 UTC: Support Team responded to customer escalations and started the incident response process

10:42 UTC: The PR was reverted

10:45 UTC: Engineering teams started to plan the recovery process

10:49 UTC: status.influxdata.com was updated to reflect the issue

10:56 UTC: All senior managers were engaged

Phase 2: The Rebuild, 11:17 - 15:26

Create deployment checklists and double check them.

Redeploy services carefully in order to connect them with existing volumes, preserve state, and accelerate recovery.

Additional services were re-deployed in parallel as data integrity was verified.

In some cases, services were restored from backup instead of redeploying, if this was deemed a safer strategy by the team.

Expecting a surge in traffic once restored, ingress points were proactively scaled up.

Phase 3: Return to Service, 15:26 - 16:04

Enabled the write service, tested, and validated functionality.

Re-enabled tasks to allow the backlog of task runs to complete in order to avoid overloading the cluster. Allowed all tasks to fail.

Once backlogged tasks had completed, they re-enabled the query service.

Phase 4: All Clear, 16:04 on

Alert customers that service has returned.

Apprise customers of their task failures and help them recover where possible.

Write and post an RCA and timeline.

Recovering

Undeleting a stateful Kubernetes app

influxdata®

First instinct - revert the change

- That’s what we initially did, but stopped the process early
- Not a good idea - complicates things by creating new volumes
- Teams stopped the line and listed a detailed recovery plan
- Restore stateful items manually, re-create rest via CD

Recovery - the correct way

- Restore Persistent Volumes to point to same cloud volumes
  - Initially manual, automated it as we gained confidence in process
- Recover first set of services
  - Zookeeper - restored from hourly backups, but this caused no issues
  - Kafka (WAL) and etcd - restored existing PVs, recreated pods
  - Enabled parts of InfluxDB Cloud to allow data writes to come through
- Recover remaining services
  - Storage - restored all PVs, re-indexed data, started ingest from Kafka
  - Deployed remaining services and increased number of replicas
  - Enabled rest of services - queries and tasks working again

Summary of recovery

- What went well
  - Cross-team effort to get our environment up again
  - Had downtime, but did not lose data
  - Stopped quick rollback attempts, created a plan first, then executed it
  - Velero backups worked for other, system-internal data without retain

- What went wrong
  - Our monitoring systems did not discover the issue immediately
  - Which led to some suboptimal rollback attempts
  - There was no runbook for this case

Post mortem

Can we not delete production again, please?

influxdata®

Improve processes around outages

- Write and test runbooks for restoring state kept in volumes
- Ensure volumes are retained across all of our environments
- Perform exercises / fire drills for recovering deleted resources
- Improve process for handling public facing incidents

Ensure such change wouldn’t get merged

- Change file structure to make it easier to detect a collision
  - Prior to this incident, all Kubernetes resources in one, large YAML file
  - Moved to one object per file - **v1.Service-*(namespace)*.etcd.yaml**

- Automate detection of duplicate objects
  - Basic test in tool that renders YAML files to detect duplicate resources
  - Kubernetes resources are subtle when it comes to apiVersion

- Bonus points - reviewing files with readable names is easier

Improve ArgoCD configuration

- Added annotations to not prune certain objects
  - Prevent parts of app that keep state from being deleted
  - Much easier to restore if stateful parts not deleted
  - Outcome of testing in practice - set for Namespace objects

- Refuse to update resource managed by another ArgoCD app
  - ArgoCD would refuse to have same Namespace object in two apps
  - Would prevent this at app of apps level

Links and resources

- Pull request to detect duplicate objects
  https://github.com/kubecfg/kubecfg/pull/1
- Issue to improve duplicate object detection
  https://github.com/kubecfg/kubecfg/issues/91
- InfluxDB and Kafka: How InfluxData Uses Kafka in Production
  https://www.influxdata.com/blog/influxdb-and-kafka-how-influxdata-uses-kafka-in-production/

influxdata®

www.influxdata.com