
KubeCon
CloudNativeCon
Europe 2025
LONDON BUS

KubeCon | CloudNativeCon
Europe 2025

From metal to apps:
LinkedIn's Kubernetes-based compute platform

Ahmet Alp Balkan (@ahmetb)
Ronak Nathani (@ronaknathani)

LinkedIn

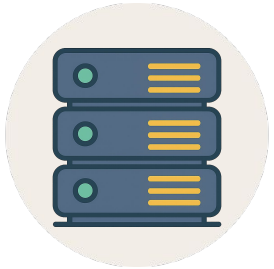
About us

	Ahmet Seattle 🇺🇸	**Ronak** Toronto 🇨🇦
First KubeCon	2016, Seattle	2022, Detroit
# of KubeCons	7	3
Hobbies	Gardening, kubectl plugins	Racket sports, podcasting

What is LinkedIn's scale?

1B+

members

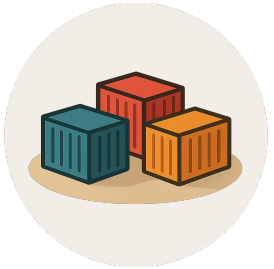


500,000+

servers

3,000+

services

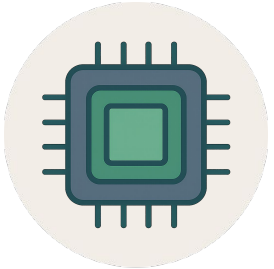


1.5M+

containers

50,000+

deploys/day



Everything on bare metal

Multiple datacenters

Next-gen compute platform

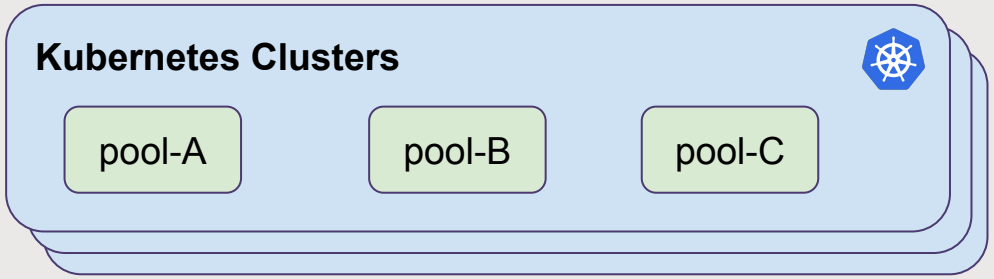
Stateless Workloads Platform

Stateful Workloads Platform

Jobs Platform

Kubernetes-as-a-service

Workload Platform Layer



Cluster/Pool/Node Lifecycle Controllers

Multi-cluster workload routing

Kubernetes Cluster Management

Datacenter Inventory Manager

Compute Broker (machine allocator)

Host Health Monitoring & Remediation

Maintenance Orchestrator

Infrastructure as a Service (IaaS)

Next-gen compute platform

Stateless Workloads Platform

Stateful Workloads Platform

Jobs Platform

Kubernetes-as-a-service

Workload Platform Layer

Kubernetes Clusters

pool-A

pool-B

pool-C

Cluster/Pool/Node Lifecycle Controllers

Multi-cluster workload routing

Kubernetes Cluster Management

Datacenter Inventory Manager

Compute Broker (machine allocator)

Host Health Monitoring & Remediation

Maintenance Orchestrator

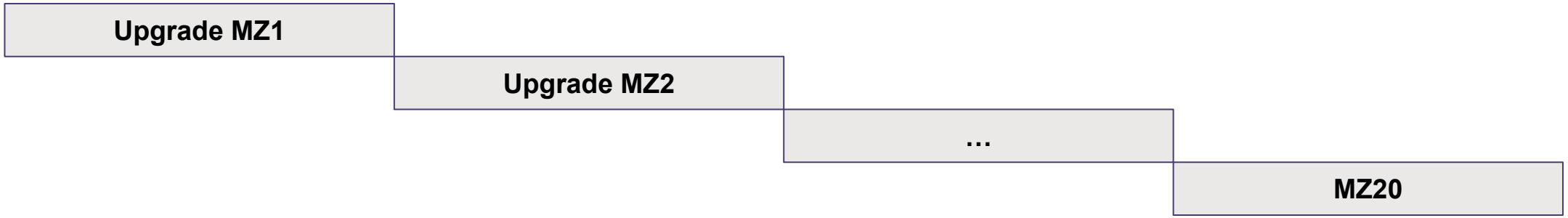
Infrastructure as a Service (IaaS)

Datacenter/machine layer

- **Inventory Manager** manages our datacenter inventory and machine properties.
- **Compute Broker** (machine allocation API)
  - A declarative gRPC API to manage **machine pools**, add/remove capacity
    - **Pools** have heterogeneous (but interchangeable) hardware
    - Each pool specifies a “node profile” (minimum machine type + configuration)
  - Source of truth for machine maintenance operations.
- **Host health monitoring** & **remediation**
  - No humans in the loop to detect unhealthy hosts and remediate-or-replace them.
- **Maintenance orchestrator** to ramp node upgrades gradually across the fleet.

Node Maintenance Zones

- Datacenters are striped to 20 **maintenance zones** (MZs) to perform software rolling update on the fleet (OS, kernel settings, kubelet…)
  - it's not a physical fault domain like AZs



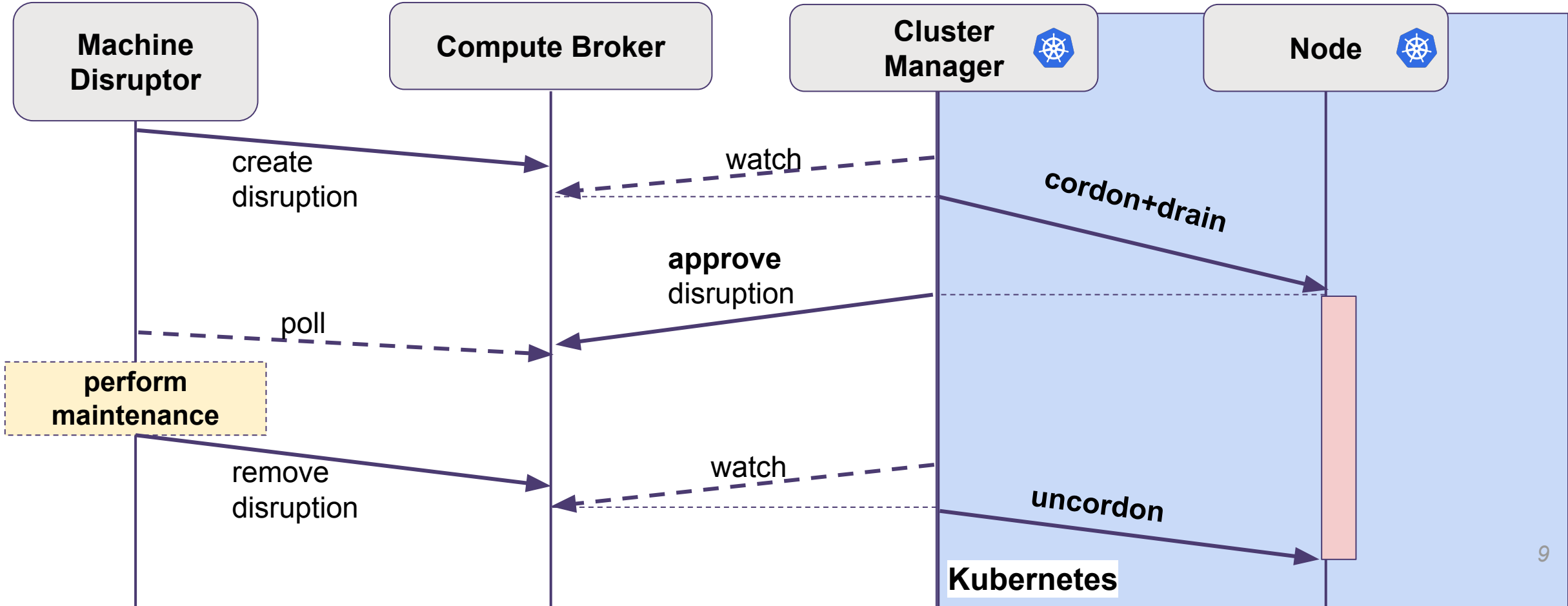
- Compute pools span multiple MZs (has nodes from every MZ, balanced)

- Kubernetes clusters are still a *fault domain* due to cluster-wide configs/policies
  - CustomResourceDefinition, MutatingWebhookConfiguration, ClusterRole, …

Coordinated node maintenance

Disruptions coordinate transferring control of the machine from K8s to **maintenance actor** (and back)

1. ***Planned***: kubelet/OS upgrades, switch upgrade, hardware decomm.
2. ***Unplanned***: host health remediation



Cluster organization and scale

- **No Kubernetes distro**
  - OSS Kubernetes configured with an in-house setup (no kubeadm, or Cluster API)
  - Works better with our machine provisioning. We also customize apiserver/etcd setup.
- Large clusters with **~5k nodes** (planning to push further)
  - Helps reduce **hardware fragmentation** across clusters, allows in-place growth
  - Clusters are **multi-tenant** with mixed workload types (stateless+stateful+batch+...)
- Kubelet upgrades happen as part of **OS maintenance**.
- **Centralized** “hub” clusters to manage *workload routing and* the clusters
  - Each app gets a separate `Namespace`, routed to a specific cluster.

KRM-style APIs for Pool Management

- Custom resources (CRDs) and controllers to **manage pools and clusters**, or **coordinate node maintenance** activities.

- Pools/Clusters are declared on the Hub cluster (managed via GitOps), reconciled asynchronously by in-house controllers.
  - *Adjusting capacity in a pool is as simple as a field update:*

KubernetesPool CR

```
spec:
  poolTemplate:
    capacity: 1200
    nodeProfile: gpu
  nodeConfig: kubelet-1.25-gpu
  nodeLabels: {...}
  requiredDaemonSets: [{...}]
  …
status: …
```

ComputeBrokerPool CR

```
spec:
  capacity: 1200
  nodeProfile: gpu
status: …
```

Compute Broker gRPC API

How we scale Kubernetes

API Server is a shared resource

- Restrict access via RBAC
- Use API Fairness and Priority (APF)

Etcd is a shared resource

- First bottleneck to hit scaling beyond 5,000 nodes
- Increased the storage limit from 8G → 16G (planning for 32G on SSDs)
- In-house etcd backup/restore system as DR strategy

Controller scalability

- Many controllers watching/caching Pods (memory-bound)
- Controller sharding isn't a solved problem *yet*

Next-gen compute platform

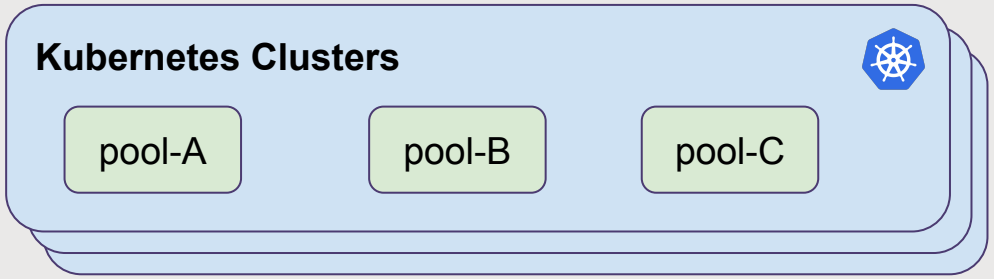
Stateless Workloads Platform

Stateful Workloads Platform

Jobs Platform

Kubernetes-as-a-service

Workload Platform Layer



Cluster/Pool/Node Lifecycle Controllers

Multi-cluster workload routing

Kubernetes Cluster Management

Datacenter Inventory Manager

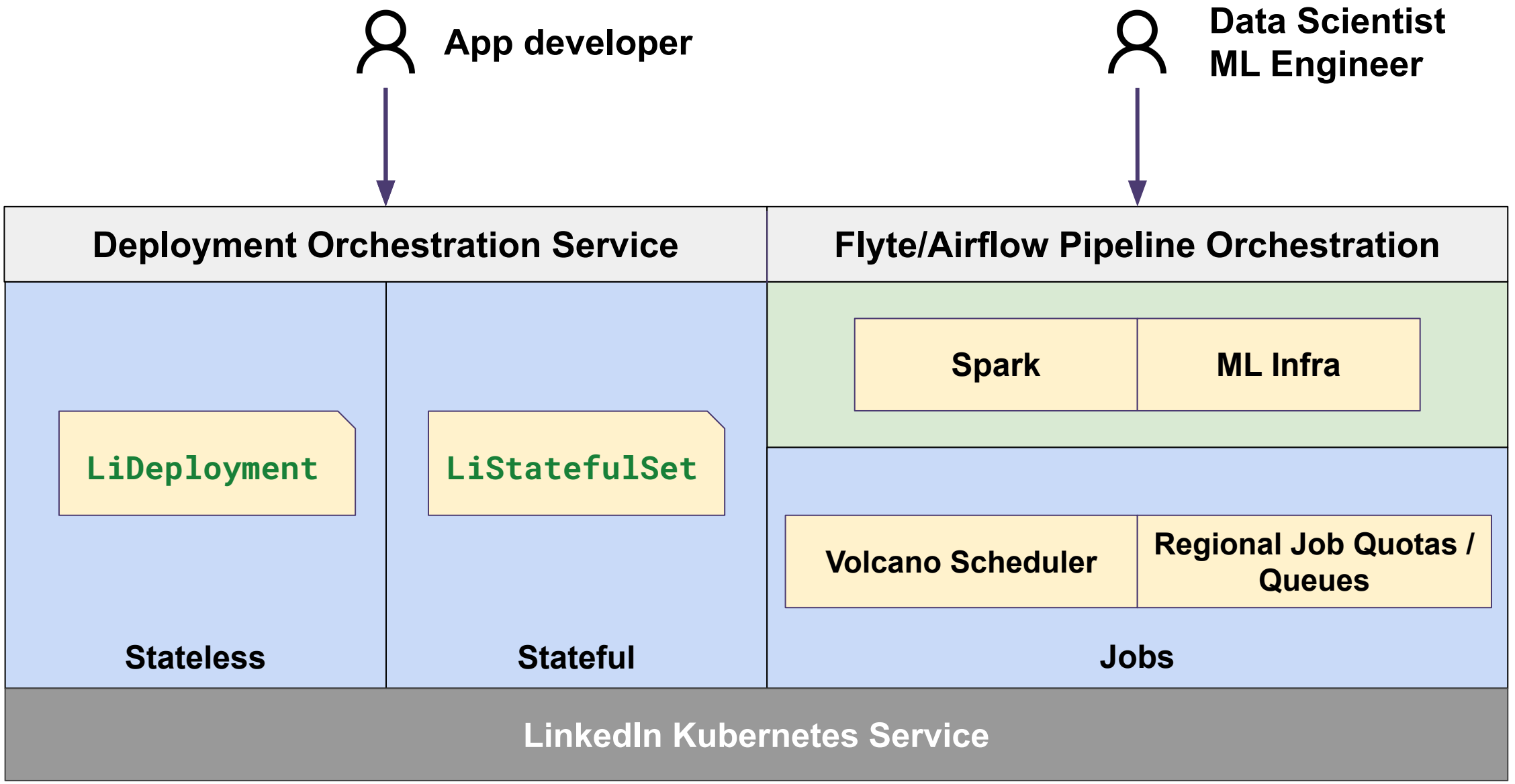
Compute Broker (machine allocator)

Host Health Monitoring & Remediation

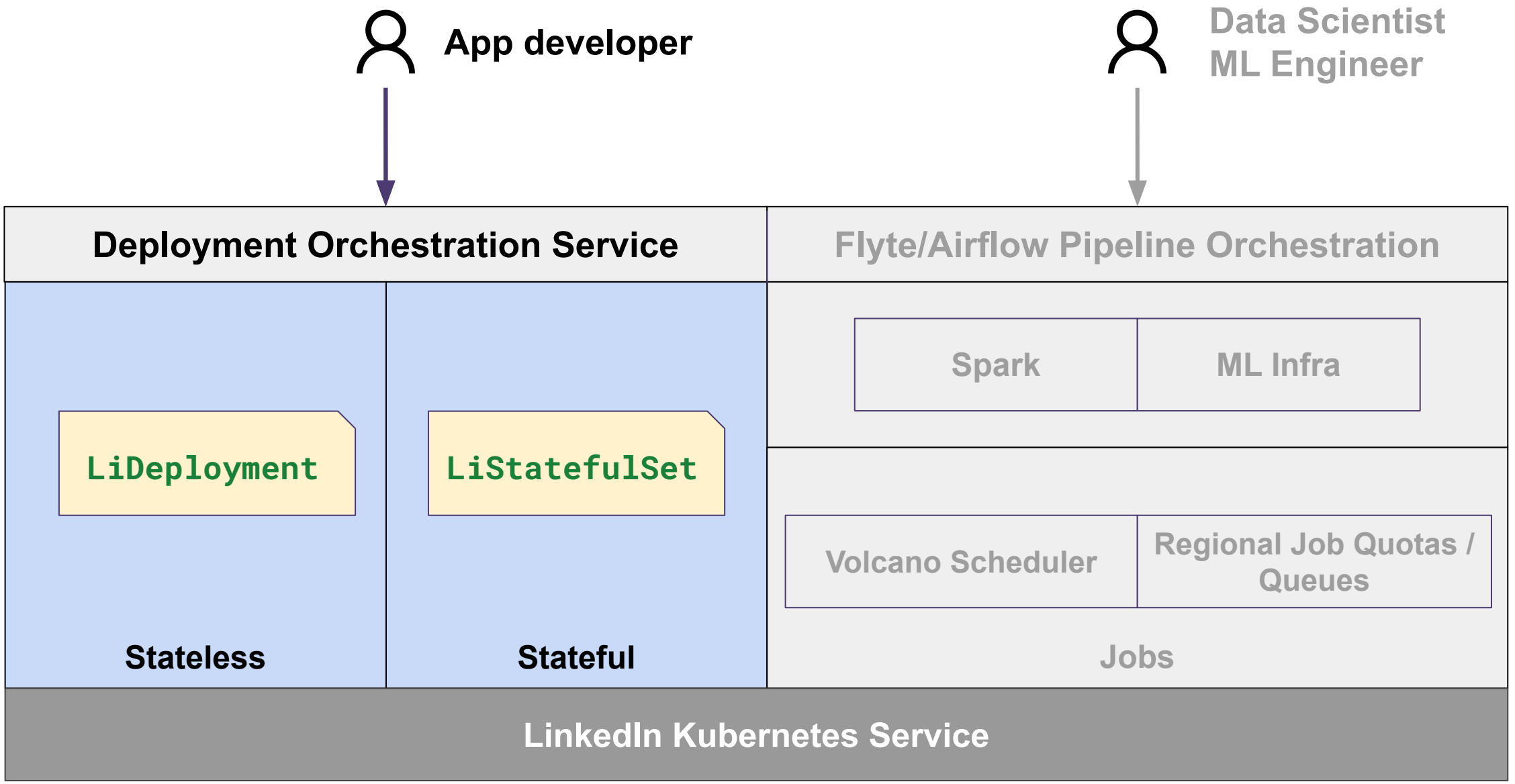
Maintenance Orchestrator

Infrastructure as a Service (IaaS)

How we use Kubernetes



How we use Kubernetes



Migration principles and progress

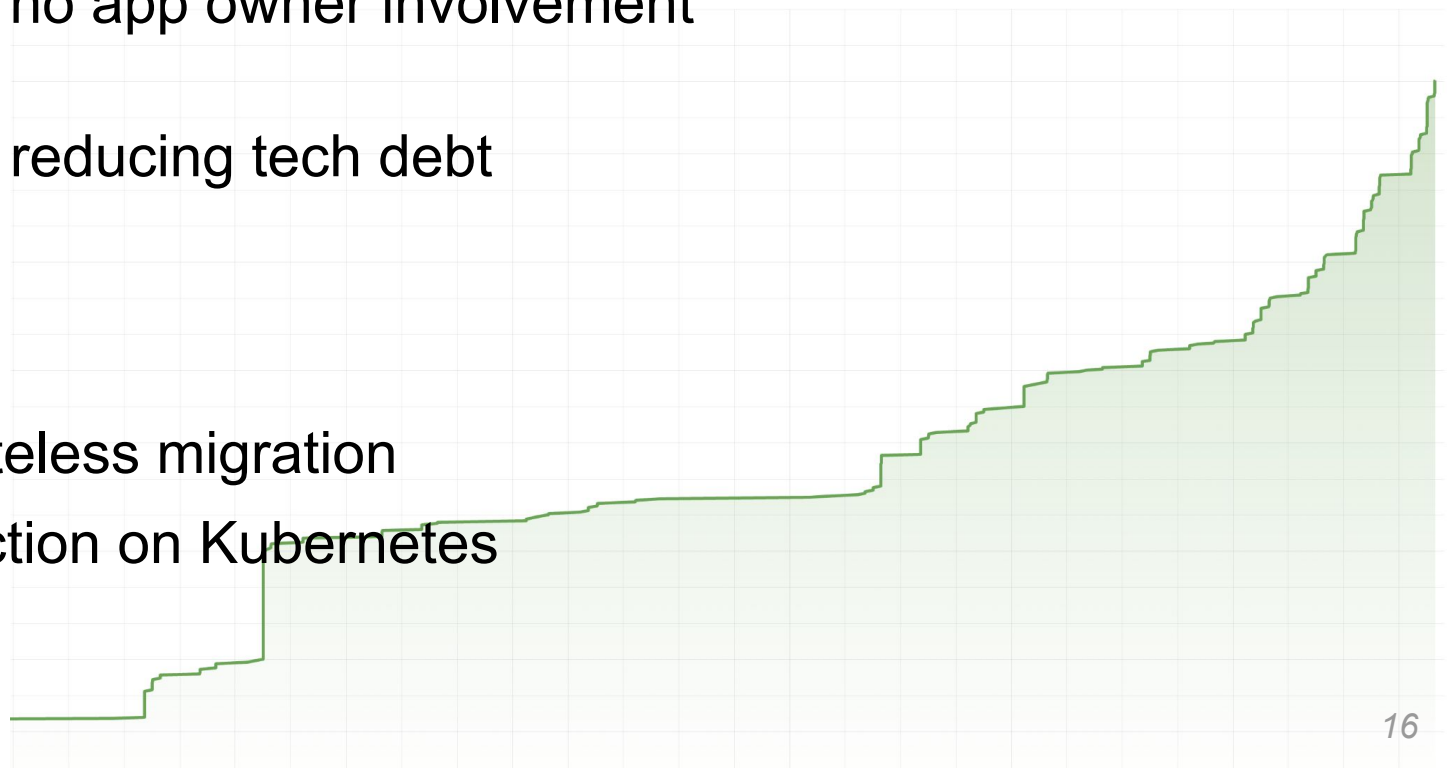
We are **migrating all services** to Kubernetes.

Principles

- No downtime to the live site
- Centrally driven, fully automated with no app owner involvement (for stateless)
- Challenge legacy requirements while reducing tech debt

Progress

- More than halfway through in our stateless migration
- Some stateful apps running in production on Kubernetes



Internal Service Infrastructure

	Cloud Kubernetes	LinkedIn Kubernetes
PKI	cert-manager	*Current*: internal PKI *Future*: cert-manager (w/ custom CA/approver)
Service Discovery	Kubernetes `Services` + coredns (cluster-scoped)	In-house (regional) based on xDS
Monitoring	Prometheus	In-house (Moving to OTel)
CNI	Many options (cluster-scoped)	Current: Host network Future: IPvLAN (global)
Network Policy	CNI-provided (cluster-scoped)	In-house (global)
Config & Secrets	ConfigMap/Secret (cluster-scoped)	In-house (regional)

Internal Service Infrastructure

Kubernetes primarily orchestrates pods for our stateless and stateful workloads.

We don’t use several Kubernetes features that only work within the cluster boundary and heavily leverage the flexibility it offers to extend it for our needs.

Monitoring	Prometheus	In-house
Config & Secrets	ConfigMap/Secret (cluster-scoped)	In-house (global)

Stateful on Kubernetes

LinkedIn has many **in-house data systems** (Kafka, Pinot, Samza, Ambry…)

- Data stored on **local SSDs** (not network-attached/block storage)
- Evicting a pod is not straightforward and requires coordination.
  - **PDBs/StatefulSets don't work here**, Pods run different shards.

One generic stateful workload operator to manage stateful pod lifecycle:

- Operator coordinates with "shard managers" of each stateful system
- A custom protocol between the operator ⇔ shard manager

Watch our KubeCon NA 2024 talk + read LinkedIn Engineering blog to learn more:

Cooperative Scheduling for Stateful Systems - Michael Youssef & Zhantong Shang, LinkedIn

784 views • 4 months ago

CNCF [Cloud Native Computing Foundation]

Cooperative Scheduling for **Stateful** Systems - Michael Youssef & Zhantong Shang, **LinkedIn** At **LinkedIn**, we develop many ...

LiStatefulSet CR

```
spec:
  application: kafka
  acmEndpoint: <endpoint>
  …
```

Stateless on Kubernetes

~10 lines

LiDeployment CR

```
spec:
  application: feed
  version:
    app: 3.1.126
    config: 1.1.10
  replicas: 1000
  resources:
    cpu: 24
    memory: 48G
  canary :
    configuration:
  …
status:
  conditions:
  - type: Ready
  …
  stable:
    ready: 990
  …
```

→

CloneSet

```
spec:
  podTemplateSpec:
  …
  volumeClaimTemplates:
  …
```

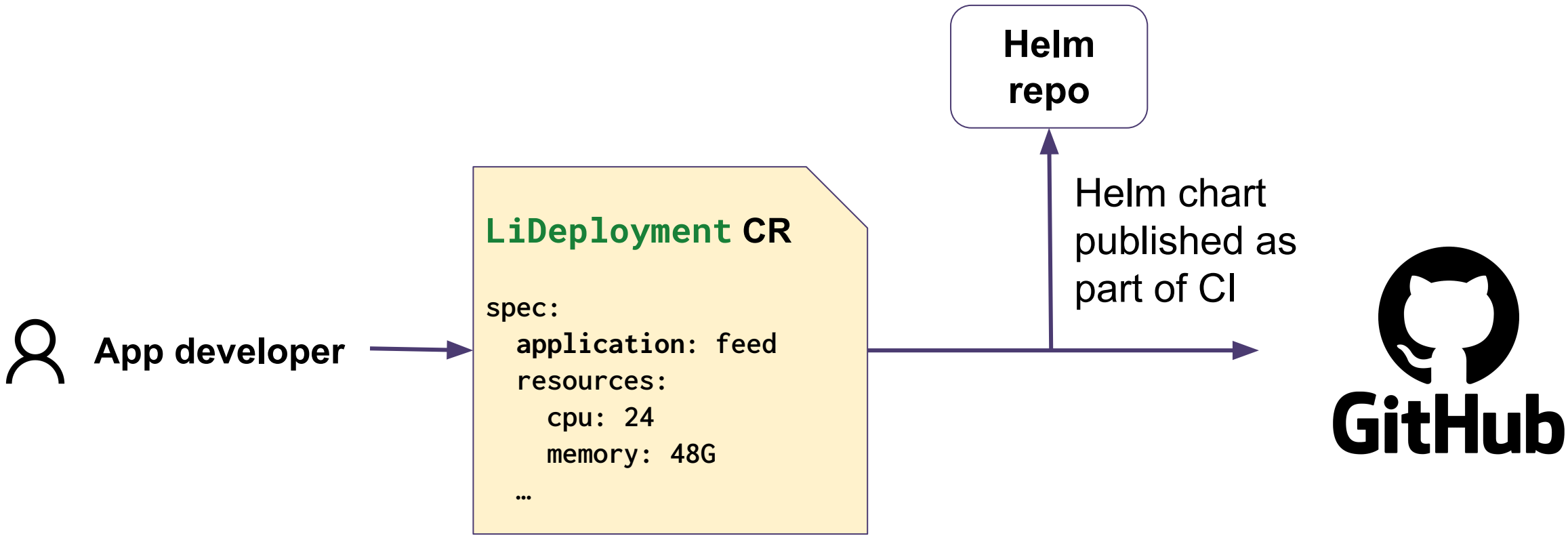
→

Pod

```
spec:
  # infra
  initContainers:
  …
  containers:
  …

<500+ lines>
```

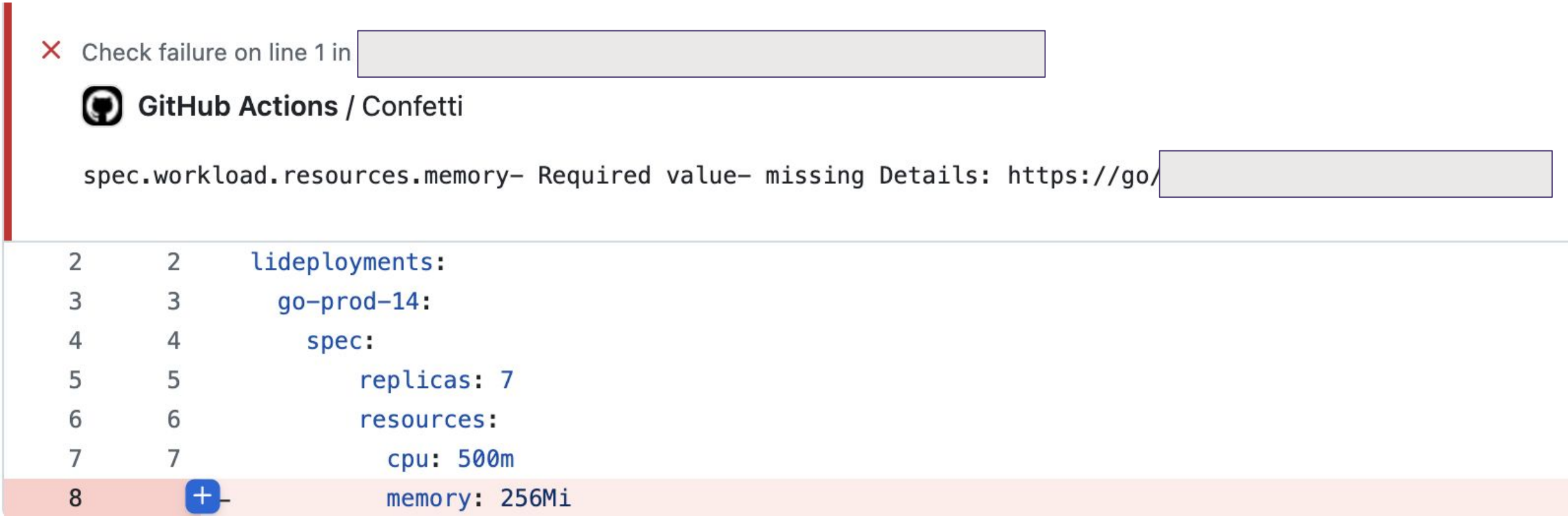
Manifest authoring



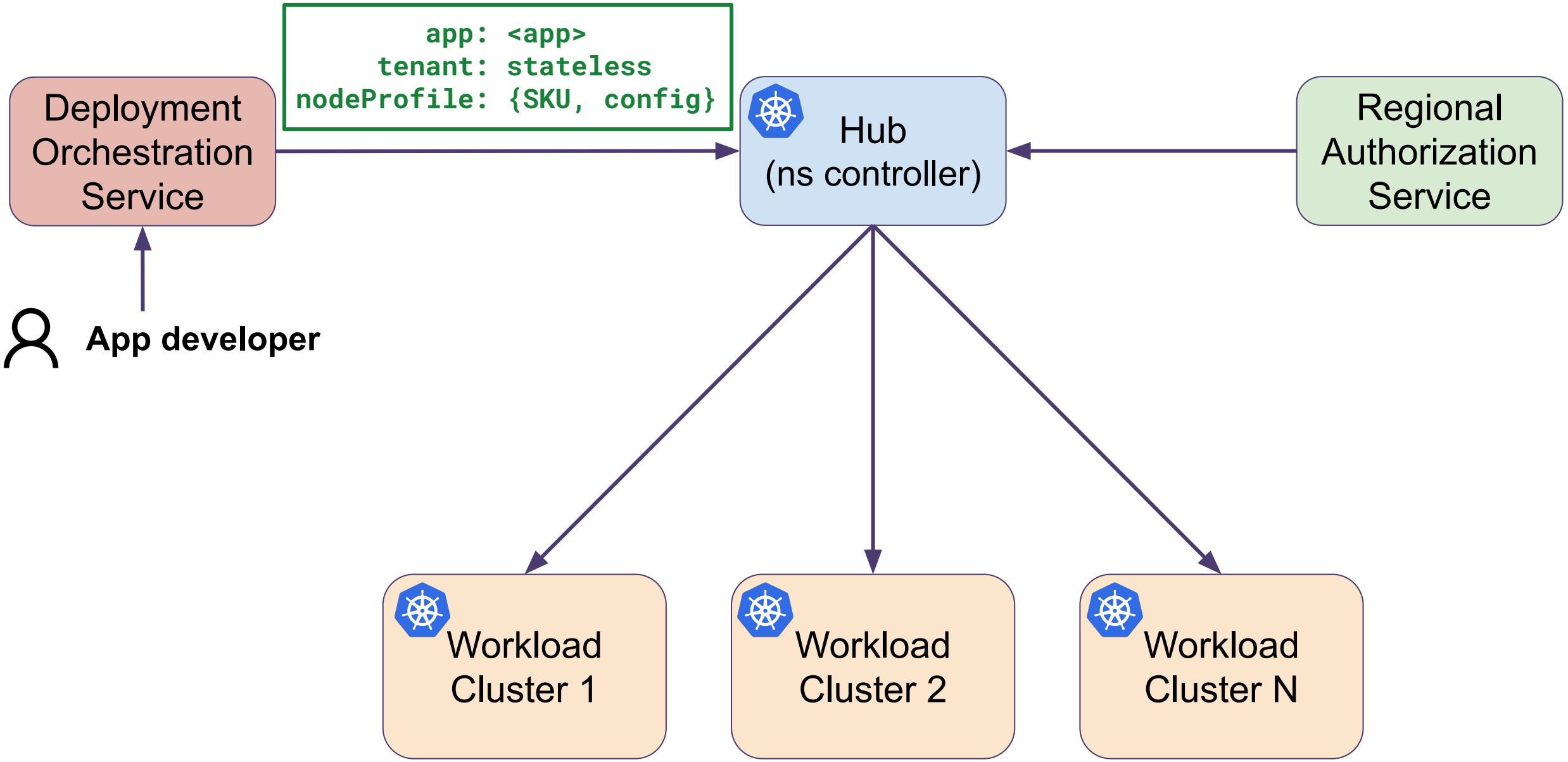
Manifest Authoring

KubeCon CloudNativeCon Europe 2025

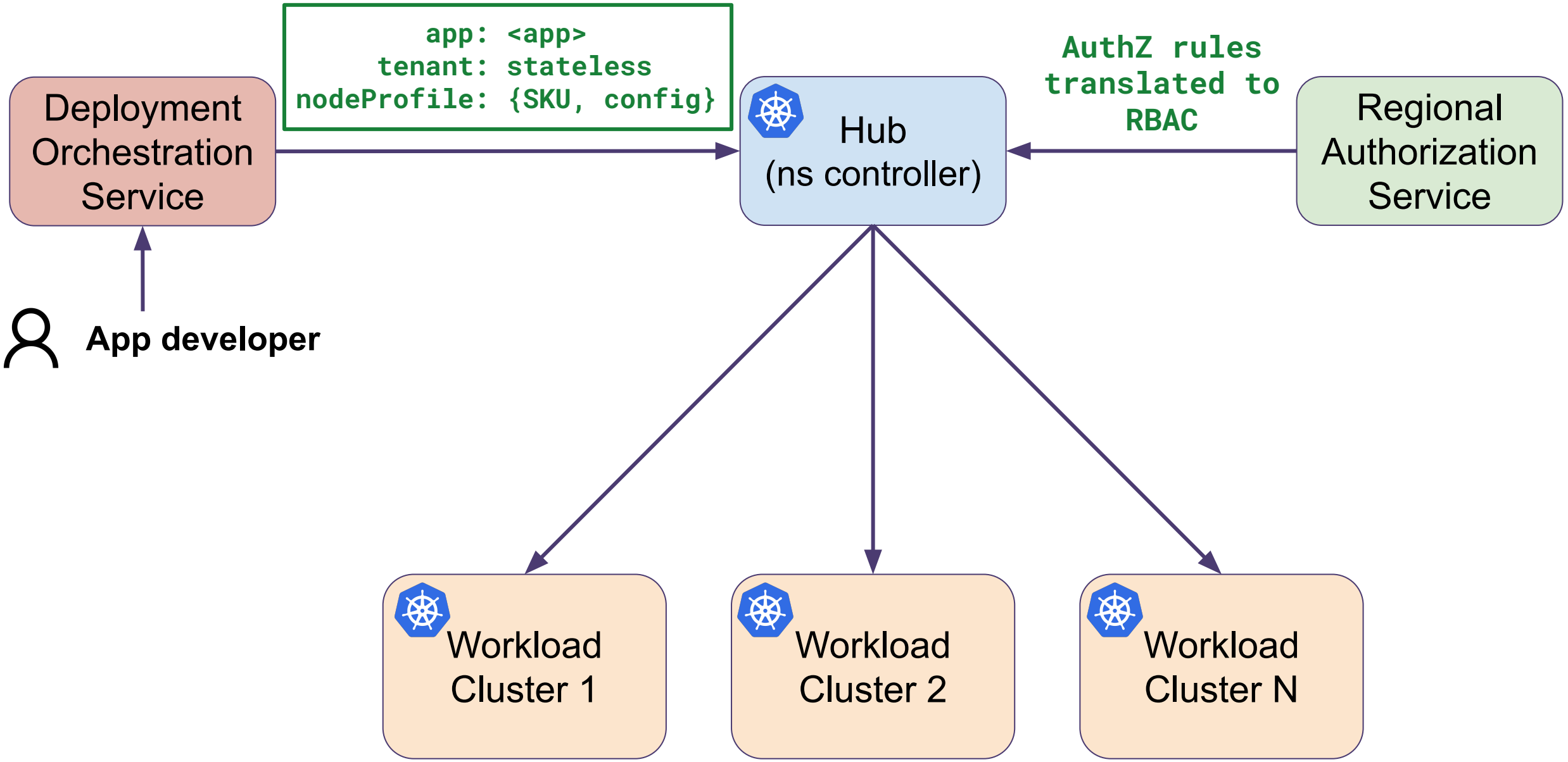
Shift left for validating user inputs/manifests.



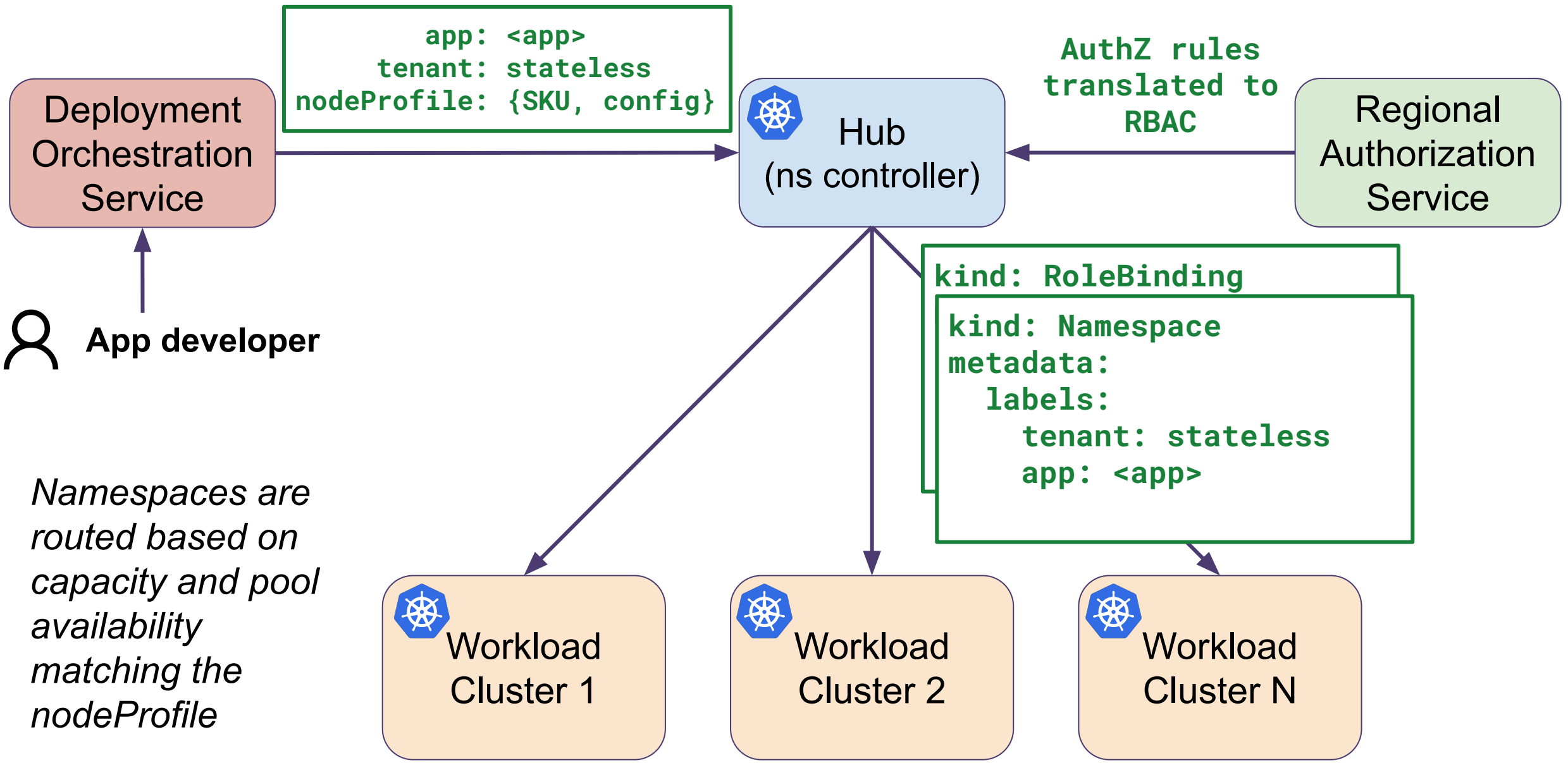
Namespace Onboarding



Namespace Onboarding

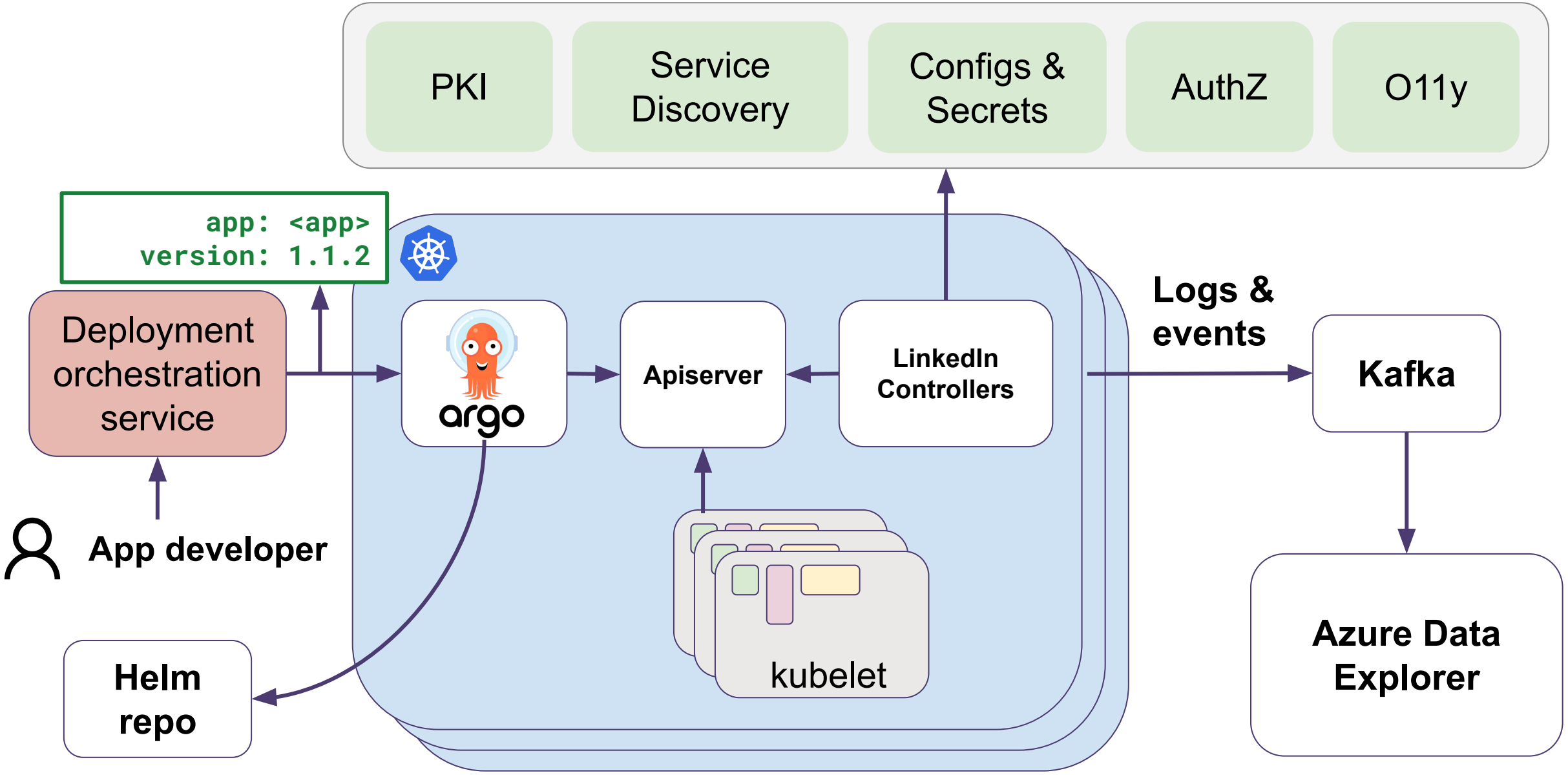


Namespace Onboarding



Namespaces are routed based on capacity and pool availability matching the nodeProfile

App owner workflow



A note on ArgoCD

End users don’t see ArgoCD but we use it heavily.

We configure ArgoCD to manage only 1 cluster.

- Served us well when the scale was smaller.
  - Replacing it with our own gitops engine for app deployments.
  - Will continue using it to deploy Kubernetes addons, policy objects etc.

- Deployments got **slower** with growing cluster size and replica counts
  - As **number of objects** in the cluster grew, application syncs slowed down.
  - As size of replicas in `Application`s grew, health status syncs slowed down.

Failures and categorization

Need to distinguish between infra vs app failures to reduce support load.

- `ProgressDeadlineSeconds` to identify rollout failures.
- `status.conditions` reflect source of failures and category.

```
- type: Ready
  status: "False"
  reason: ProgressDeadlineExceeded
  message: |-
    Deployment failed because the application failed to start.
    FailureDetails: 18 Pod(s) are UnHealthy because of unready Containers: <redacted>:0.1.425
    FailureCategory: App

    Run 'kubectl in status lideployment' to get more details.
    Refer to https://<redacted> for more information on how to debug further.
    Rollout timed out. Configured timeout: 600 seconds.
  observedGeneration: 25
  lastTransitionTime: "2025-03-18T00:03:23Z"
  lastUpdateTime: "2025-03-18T12:10:41Z"
```

UX

Internal kubectl plugin that only exposes internal custom resources and pods.

- Automatically figures out which cluster/namespace for an app.
- Custom troubleshooting subcommands.

Internal UI for browsing/troubleshooting workloads

- Watching/aggregating data from all clusters into a centralized storage to power this UI with near-real time information.

```
Status for LiDeployment: "comms-onboarding-backend-0"
-----------------------------------------------------------------
  Name                          | comms-onboarding-backend-0
  Namespace                     |
  Cluster                       |
  Desired App Version           | 0.1.441
  Desired Config Version        | 0.848.0
  Replica Health Status         | Healthy replicas: 2/4
  Replica Rollout Status        | Replicas on desired version: 2/
  Rollout Status Reason         | ProgressDeadlineExceeded
  Rollout Status Message        | Deployment failed because the d
                                | FailureDetails: 2 Pod(s) are Un
                                | comms-onboarding-backend:0.1.44
                                | FailureCategory: App
                                |
                                | Run 'kubectl in status lideploy
                                | Refer to https://go/
                                | further.
                                | Rollout timed out. Configured t
  InitContainers Status         | AllReady
  Application Container Status  | 0/2 comms-onboarding-backend:0.
                                |       2 Waiting, Reason: Crash
                                | 2/2 comms-onboarding-backend:0.
                                |
  Replica Phases                | 4/4 Running
  Unhealthy Replica Reasons     | 2/4 ContainersNotReady
                                | [ContainersNotReady]: Please fo
-----------------------------------------------------------------
```

API Guardrails

Delete protections: If “kubectl delete” something can cause an outage, prevent it.

- All user-facing custom resources
- Namespaces that have resources in them
- CRDs that have CRs

Other accident preventions

- scaling down >X% in one shot is forbidden
- upper bound for allowed maxSurge or canary percentages

What's next

Workload federation

- Aligning clusters with maintenance zones, tolerate single cluster failure.
- Customers growing in-place hit safe scaling limits for a cluster.
- Helps with machine types fragmented across different clusters.

Better resource isolation

- CPU pinning to address noisy neighbor problem.

IPv6 Pod IPs with a flat network spanning multiple regions

- Using ipvlan CNI

Kubeception

- Run Kubernetes control plane itself as Pods in another cluster.
- Makes cluster creation and management easier at scale.
- Stacking components of different clusters on the same node.

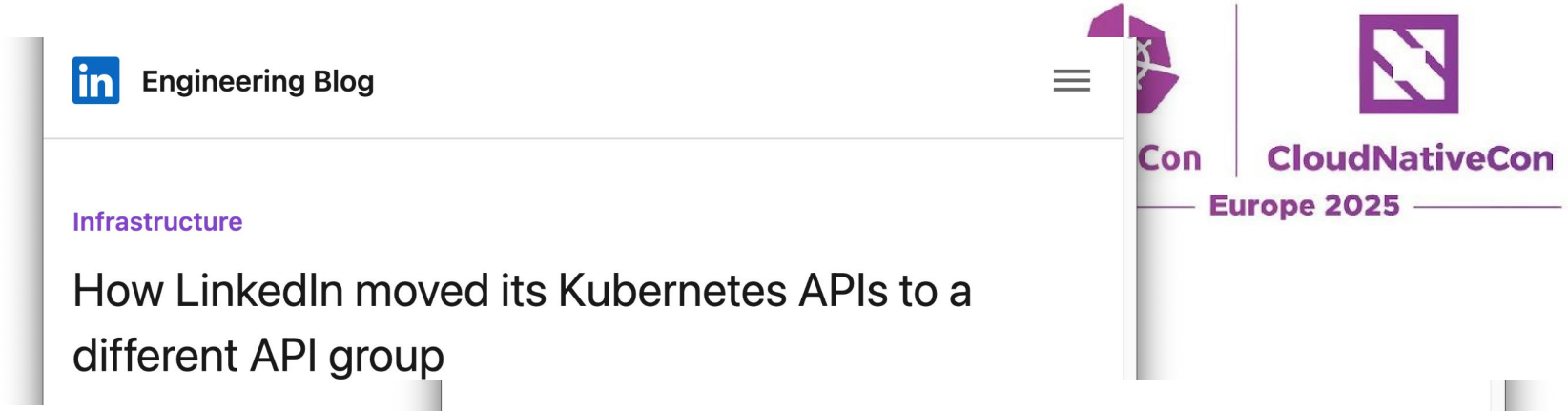
Migration Lessons

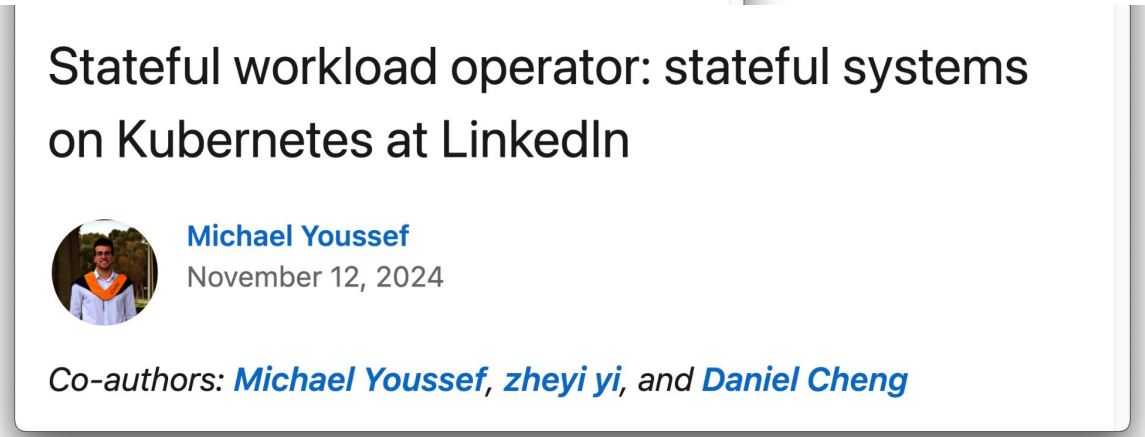
- Start early and make incremental progress.
  - …there will be a long tail.
- Figure out which tech debt to solve now vs later.
- Be intentional about what features to use in Kubernetes.
- Don’t give raw Kubernetes to your customers
  - Invest in building abstractions
- Invest in guard-rails to prevent user errors
- Develop good user guides for self-serve troubleshooting

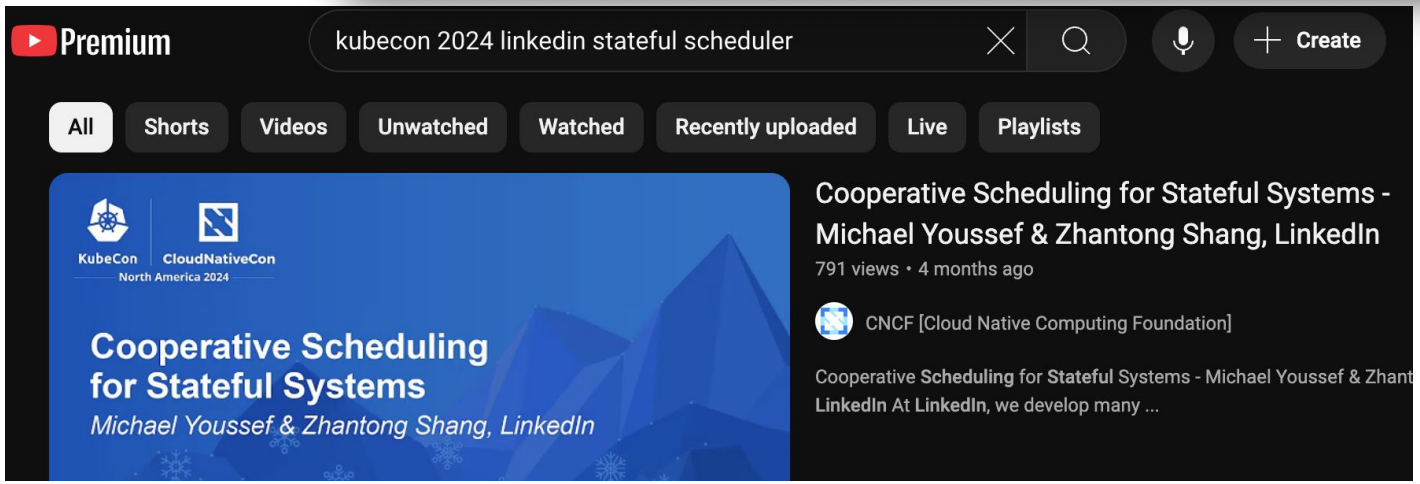
Thank you!

We are hiring
in the US
(Bay Area/Seattle)

Feedback:





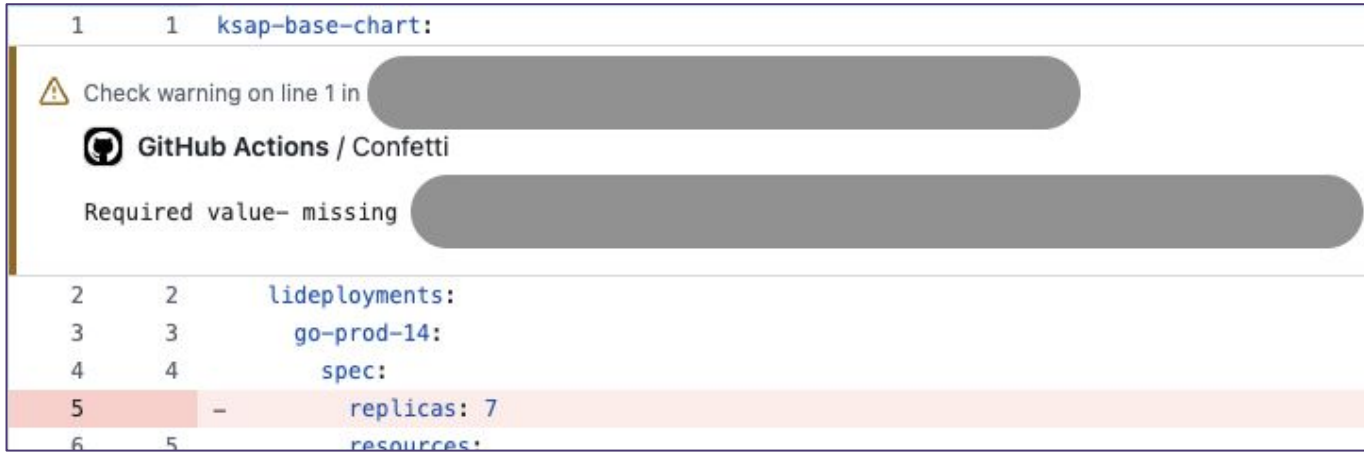


Migration - Challenges

- Generating container images
  - App owners don’t write Dockerfiles, it’s all auto-generated for them.
- Thousands of microservices to migrate
  - …without involving application owners.
- Deployment failure categorization
  - surfacing Kubernetes specific failure points to non-Kubernetes-savvy app owners
- Different Debugging UX for customers had to change

Failures and categorization

Shift left for validating user inputs/manifests.



Need to distinguish between infra vs app failures for app deployments.

- `ProgressDeadlineSeconds` to identify rollout failures.
- `status.conditions` reflect source of failures and category.

```
- type: Ready
  status: "False"
  reason: ProgressDeadlineExceeded
  message: |-
    Deployment failed because the application failed to start.
    FailureDetails: 18 Pod(s) are UnHealthy because of unready Contai
    FailureCategory: App

    Run 'kubectl in status lideployment' to get more details.
    Refer to https://<redacted> for more information on how to debug
    Rollout timed out. Configured timeout: 600 seconds.
  observedGeneration: 25
  lastTransitionTime: "2025-03-18T00:03:23Z"
  lastUpdateTime: "2025-03-18T12:10:41Z"
```