KubeCon | CloudNativeCon

North America 2023

KubeCon | CloudNativeCon

North America 2023

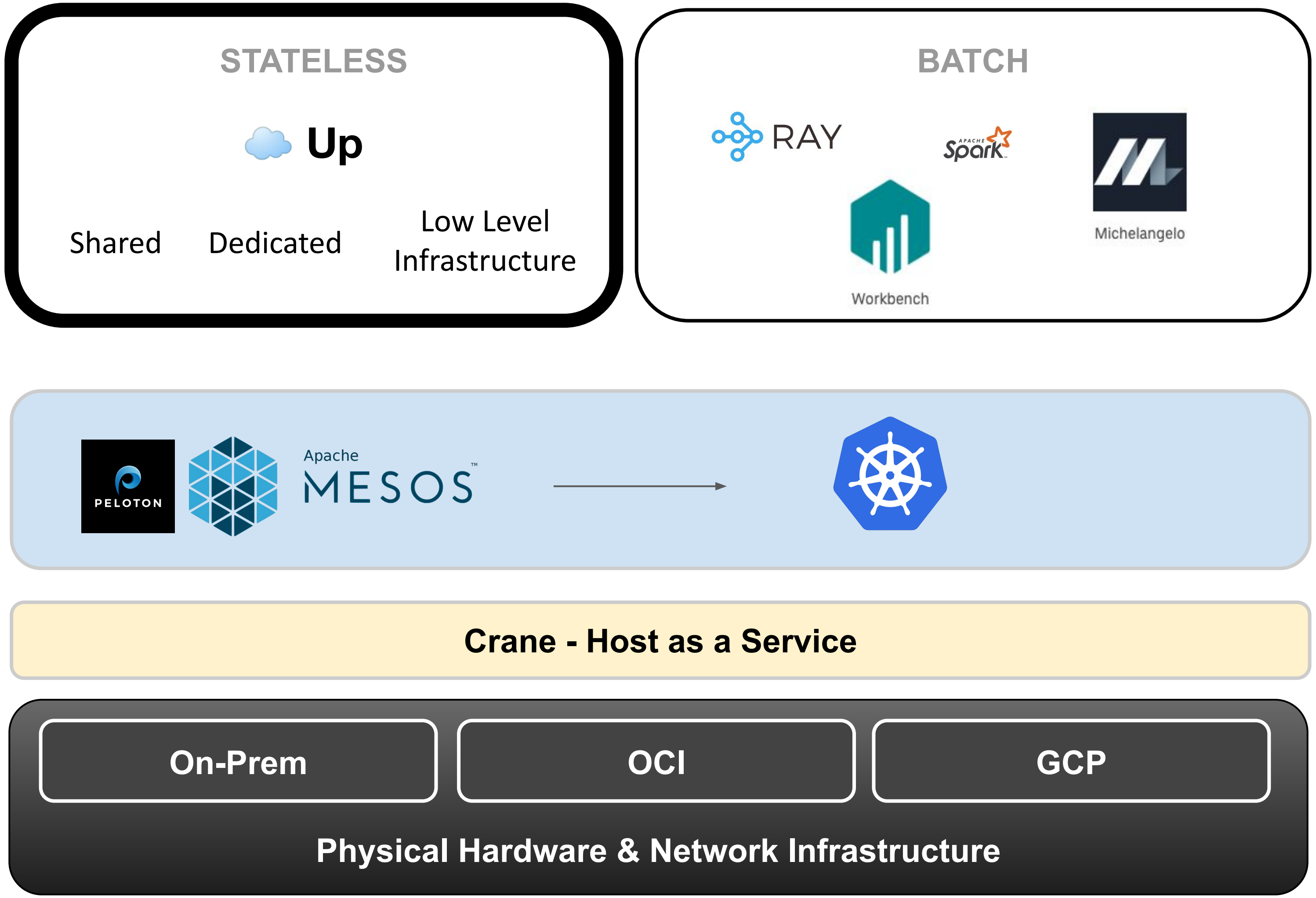
On the right tack: Kubernetes at Uber Scale

Aditya Bhave
Apoorva Jindal

Agenda

- Uber Container Platform Overview
- K8S Migration Status
- Migration Principles
- Works out of the box!
- Uber specific features
- Scale
- Migration Learnings
- What’s next?
- Acknowledgements
- Q&A

Compute Team @ Uber



STATELESS

Up

Shared

Dedicated

Low Level Infrastructure

BATCH

Michelangelo

Workbench

Crane - Host as a Service

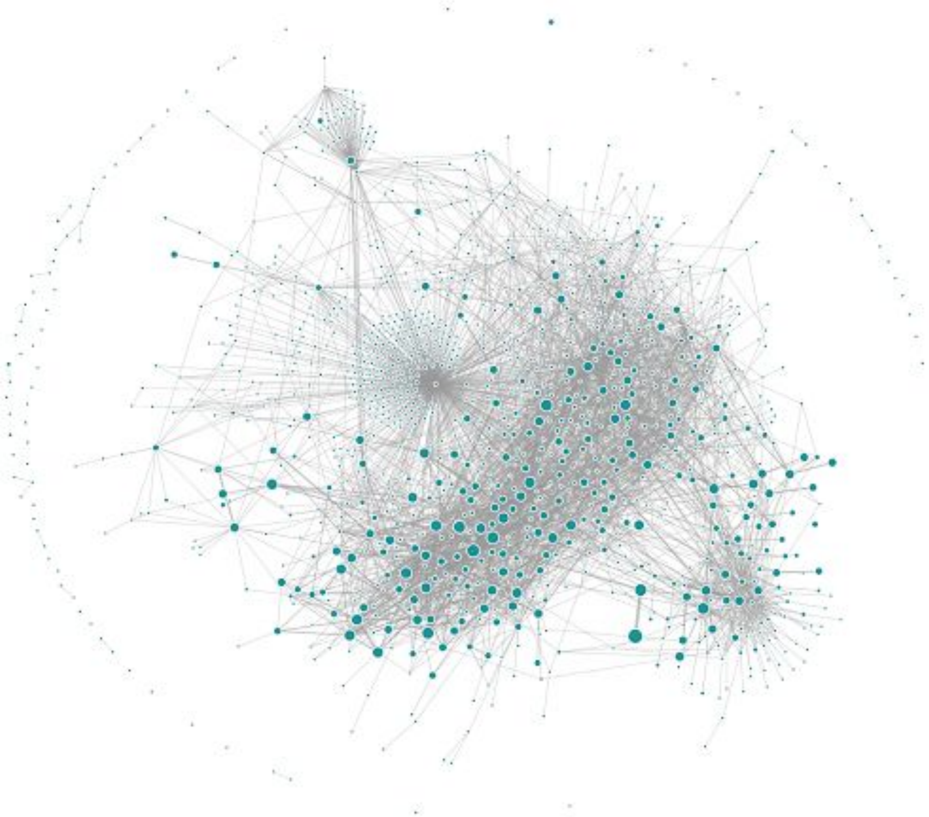
On-Prem

OCI

GCP

Physical Hardware & Network Infrastructure

Stateless Overview



4000+ microservices

2M+ cores

100K+ service deploys per day

1.5M+ containers deployed per day

500K+ containers

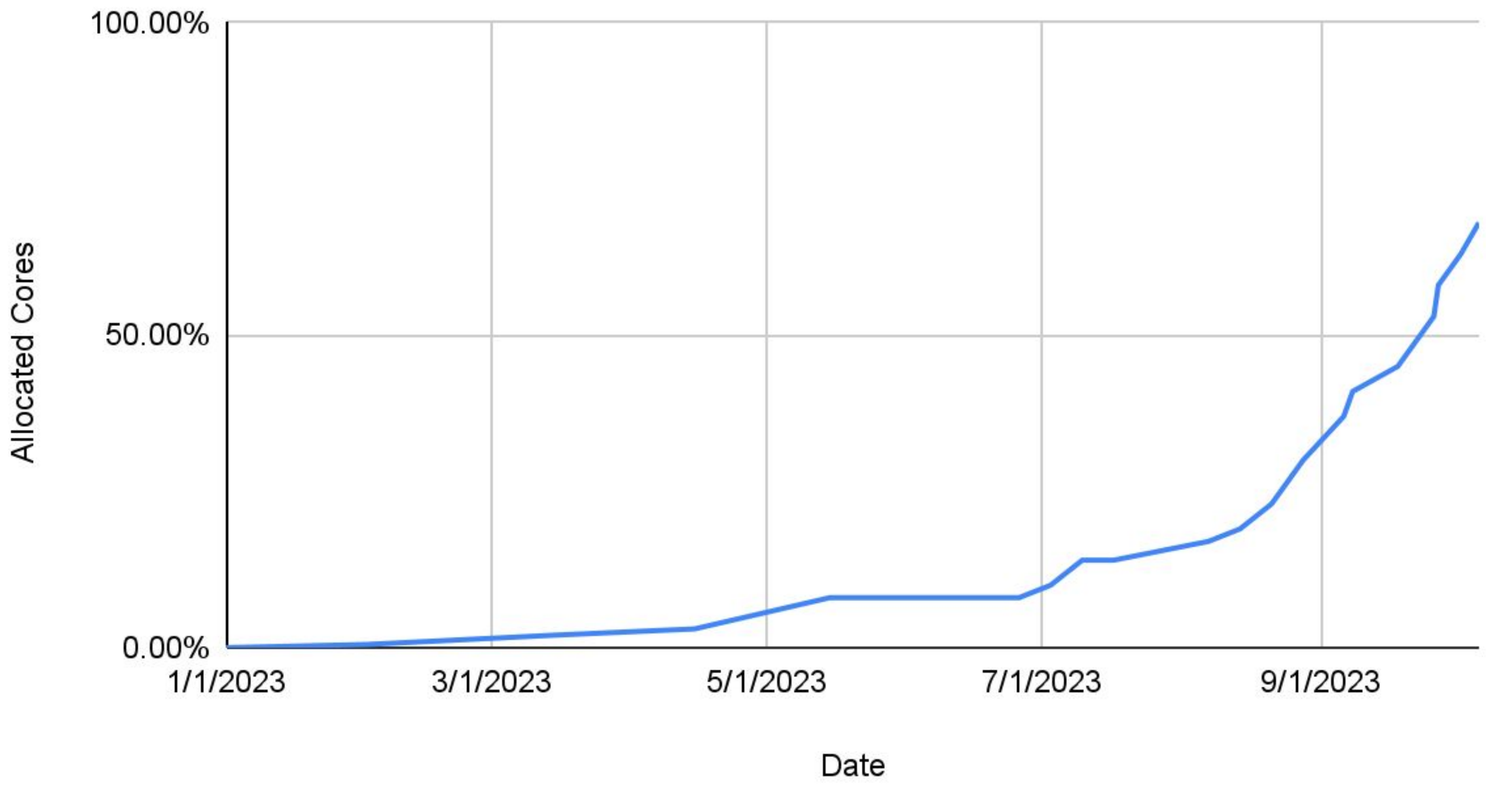
K8S Migration Status

Migration Progress

- > 70% of the fleet
- Multiple 5K node clusters
- Largest cluster stats:
  - 50k+ pods
  - 7k+ deployments
  - 5.5k+ nodes

End Target - 100% MY’2024

Kube Stateless Migration over Time



Migration Principles

Easy Upgrades

- Run same Kubernetes version as cloud providers
- Minimal changes to upstream
- Use Kubernetes native extensibility to add Uber specific customizations
    - Scheduler plugins
    - CRDS

Reliable Upgrades

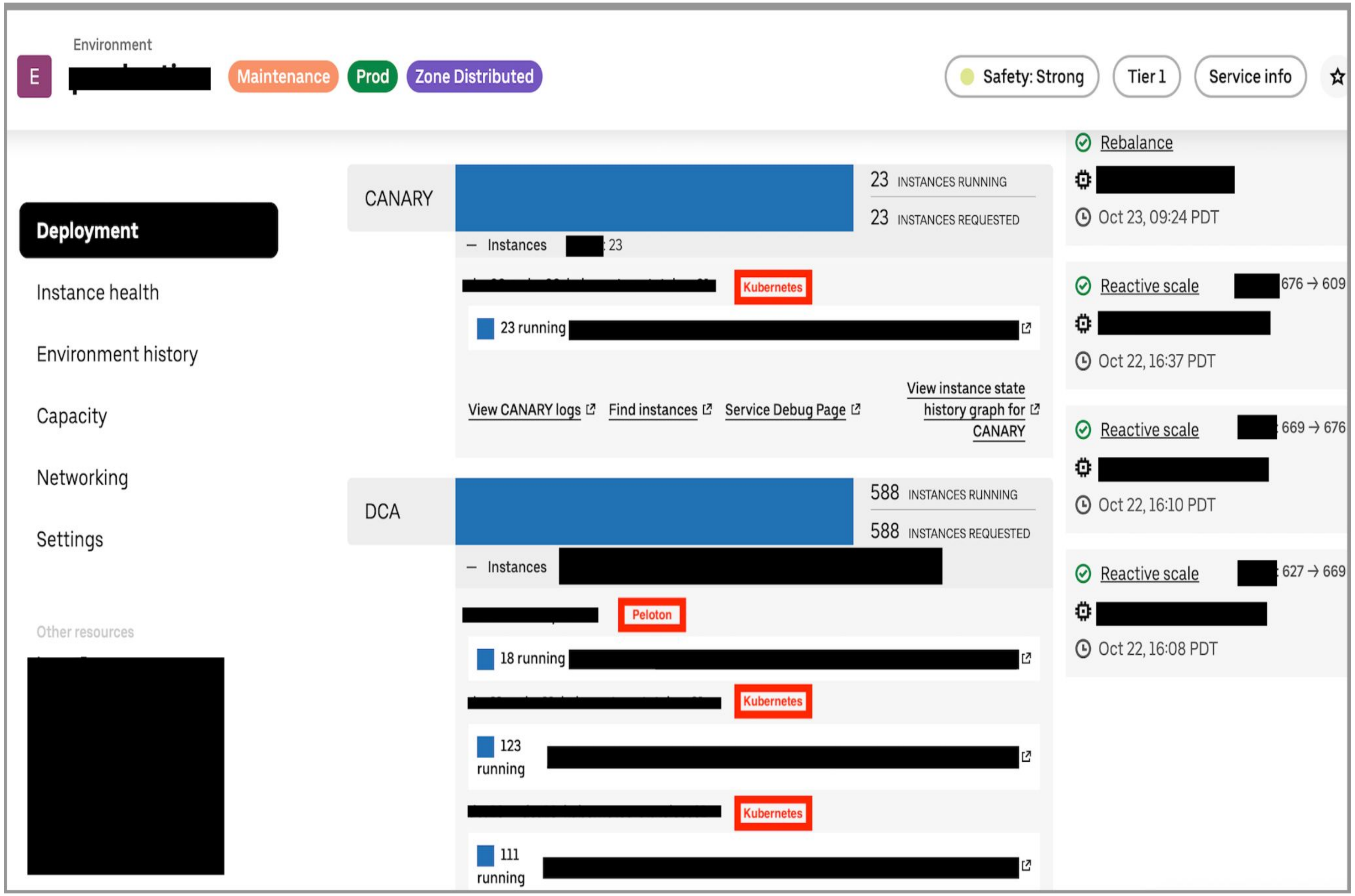
- Extensive release validations
    - 100s of integration, end-to-end and performance tests
    - Continuous probes running in production clusters

Transparent Migration

- Transparent to Uber developers
    - No difference between Mesos and Kubernetes
- Incident free
    - No business impact to Uber
- Automated
    - No manual effort required

Up ☁️

- Uber's global stateless federation layer
  - Primary service owner interface
  - Provides features like safe rollouts, continuous deployment etc.
  - Abstracts cluster technology from developers
- Cluster Selection
  - Rebalances services to clusters with low allocations
- Enables automated migration
  - Add capacity to Kubernetes ⇒ Move services from Mesos to Kubernetes



Environment

E [redacted] Maintenance Prod Zone Distributed

Safety: Strong | Tier 1 | Service info | ☆

Deployment

Instance health

Environment history

Capacity

Networking

Settings

Other resources

[redacted]

CANARY

23 INSTANCES RUNNING

23 INSTANCES REQUESTED

– Instances [redacted] 23

[redacted] Kubernetes

23 running [redacted]

View CANARY logs | Find instances | Service Debug Page | View instance state history graph for CANARY

DCA

588 INSTANCES RUNNING

588 INSTANCES REQUESTED

– Instances [redacted]

[redacted] Peloton

18 running [redacted]

[redacted] Kubernetes

123 running [redacted]

[redacted] Kubernetes

111 running [redacted]

✓ Rebalance

⚙ [redacted]

Oct 23, 09:24 PDT

✓ Reactive scale [redacted] 676 → 609

⚙ [redacted]

Oct 22, 16:37 PDT

✓ Reactive scale [redacted] 669 → 676

⚙ [redacted]

Oct 22, 16:10 PDT

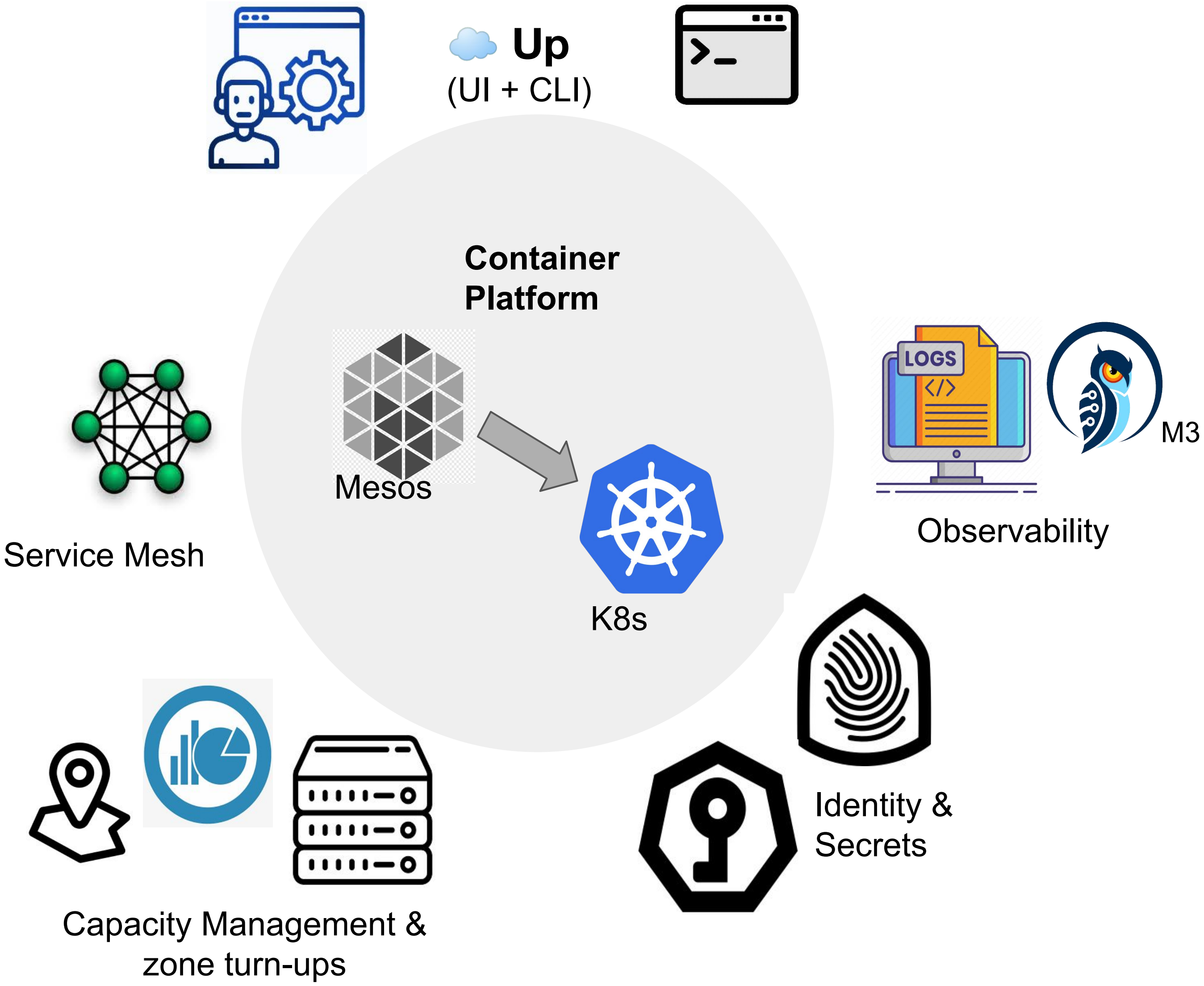
✓ Reactive scale [redacted] 627 → 669

⚙ [redacted]

Oct 22, 16:08 PDT

Transparent Migration

- Compute is a central infrastructure piece integrating with numerous Infrastructure platforms
- Rebuild all these existing integrations (for example rebuild Up -> Mesos integration).
- Kubernetes and Mesos are very different.
    - Hence each integration requires a complete re-design and implementation



Works out of the box!

- Default scheduler is awesome!!!!!!!!!
  - 100 pods/sec
  - Heavily leverage the plugin architecture for Uber specific customizations

- AuthN/AuthZ
  - Use certs for authN and role/cluster bindings for authZ
    - Very intuitive and granular
    - Huge upgrade over the current security posture
  - Now exploring authentication proxies to potentially setup personnel access control

- Priority queues and flow control
  - Protect APIServer and ETCD
  - Both operator + operation specific rate limits
  - Regulate access patterns to our clusters

- Controller-runtime ecosystem
  - All controllers/operators use this framework
    - Really intuitive to use
    - Great telemetry & no performance hit

- Support for separate events DB

Uber Specific Features

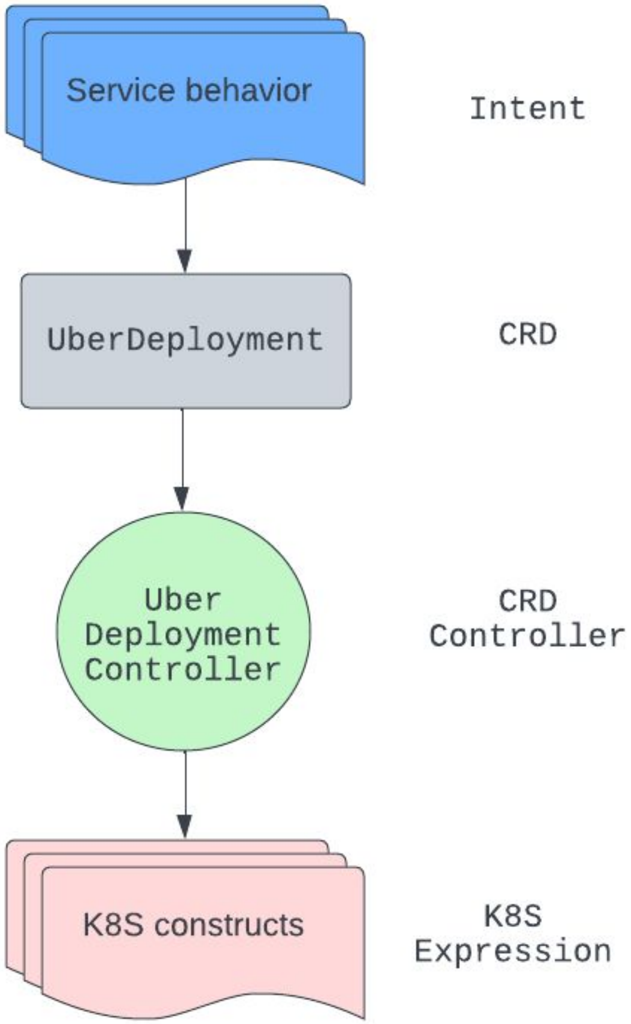
- **Developer experience**
  - **Abstracting developer intent*
  - **Container Artifact retrieval (eg: logs, heap dump, core dump etc.)*
  - K8S UI scale and stability improvements
  - Container access CLI

- **Developer velocity**
  - **Faster updates and rollbacks*
  - Speed up pod topology spread placement by 3x

- **Deployment Safety**
  - **Controlled Scaling*
  - Load Aware Placement

- **Misc. Features**
  - **Unique instance IDs*
  - Support for ulimit

#1 Abstracting Developer Intent



Uber Deployment (CRD)

- Define service specific intent

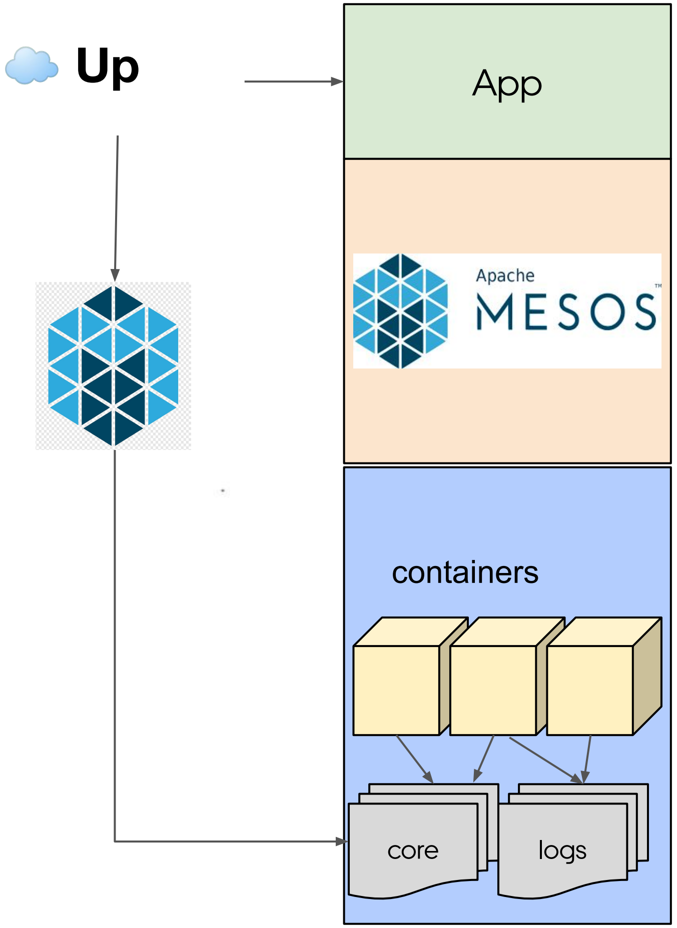
Controller

- Translate intent to appropriate K8S expression

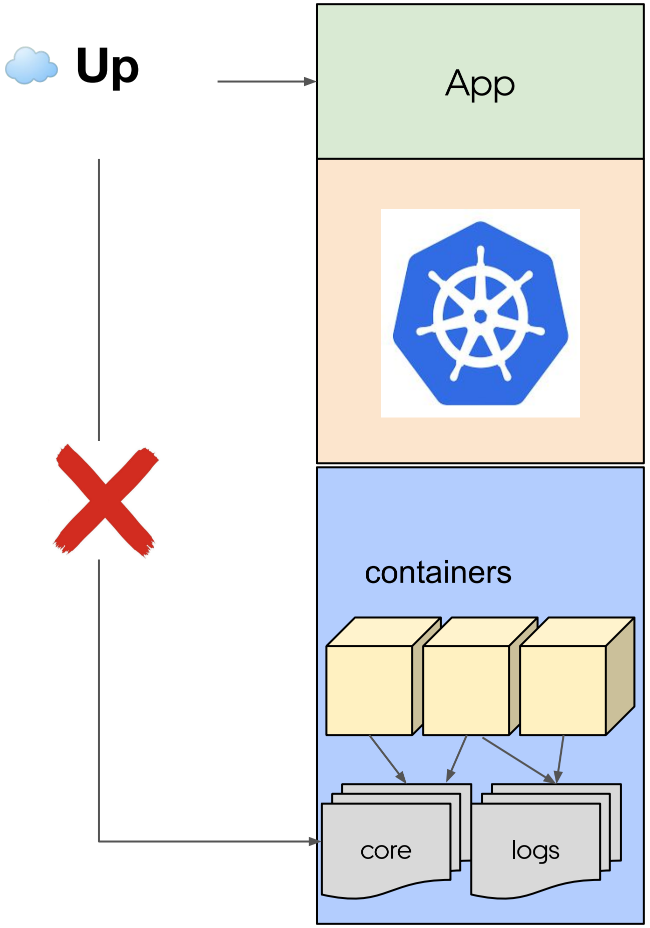
Service Intent

- Image prefetch
- In-place upgrades
- Controlled scaling
- Setting ulimits
- Dedicated Hosts, custom SKUs etc.

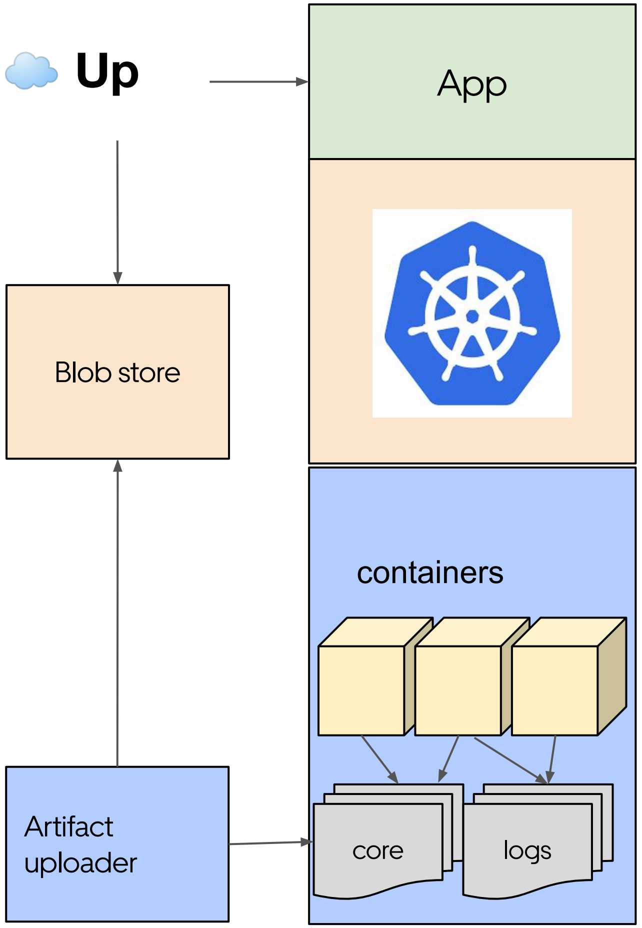
#2 Container Artifacts



Today
Container artifacts (core, heap, logs) not cleaned up on exit.
Accessible on Up via Mesos.



Native Kubernetes
Container artifacts are deleted on container exit.

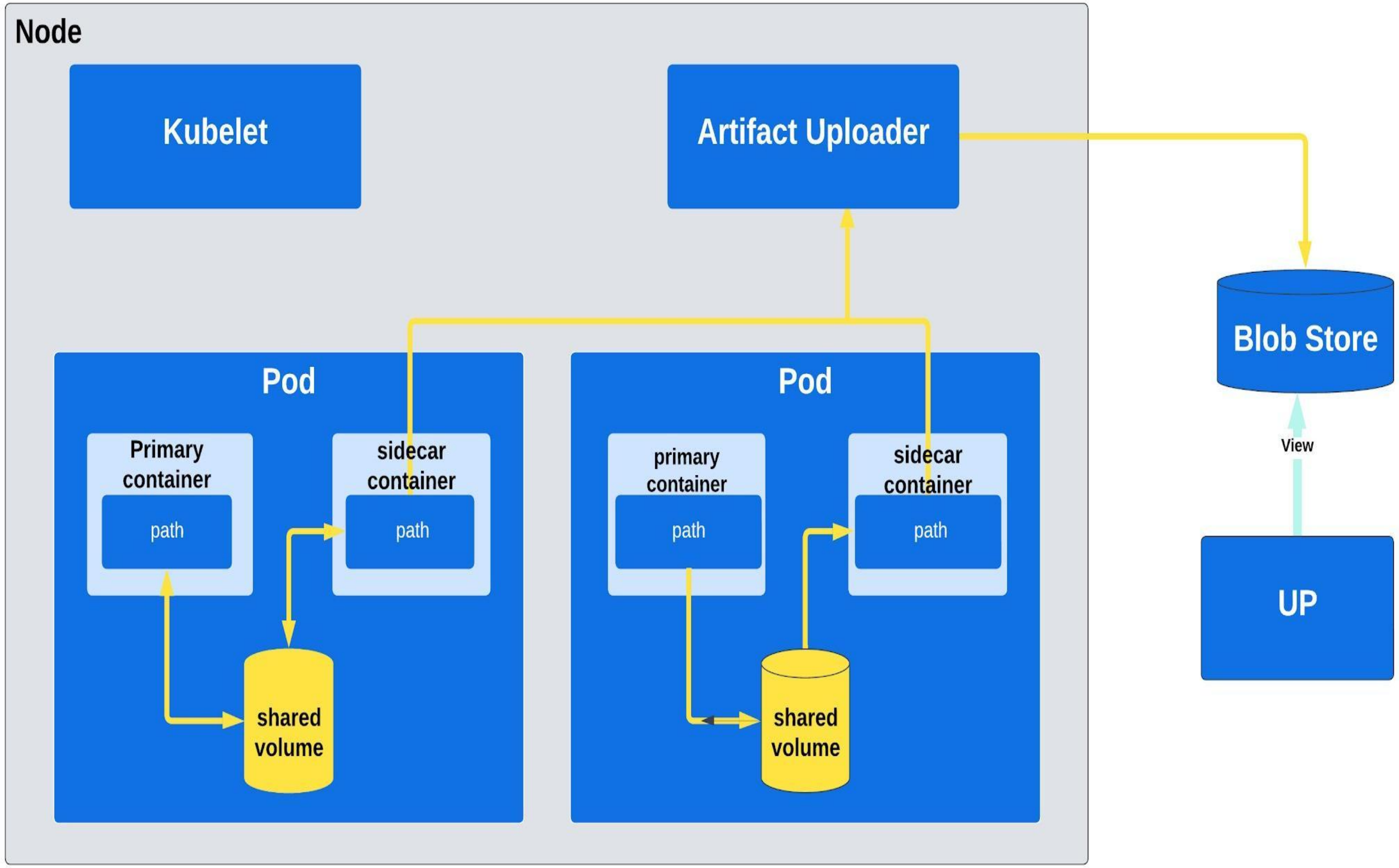


Uber Kubernetes Engine
Container artifacts uploaded to blob store on container exit.
Accessible on Up via blob store.

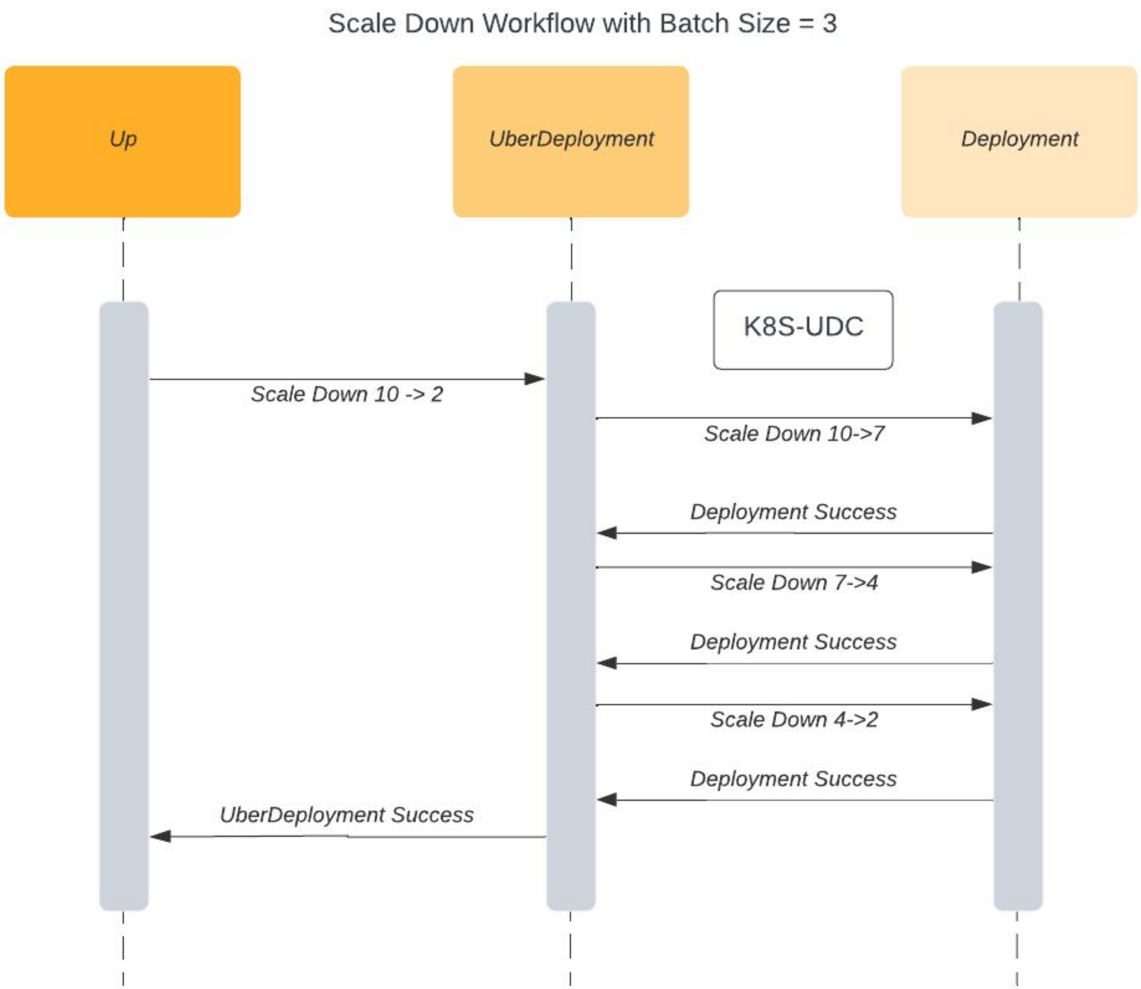
#2 Container Artifacts

- Sidecar Container
  - Share a volume with the primary container
  - Pauses pod deletion after primary container exits

- Artifact Uploader
  - Upload artifacts after primary container exists
  - Kill sidecar container, enable pod deletion



#3: Controlled Scaling



Why controlled (slow) scaling?

- Rapid scaling operations causes service instability in some sharded services
- Ex. Apache Helix based, celery worker based services.

Solution

- CRD controller batches the scaling update into multiple steps to control scaling based on service intent

Closest analogs

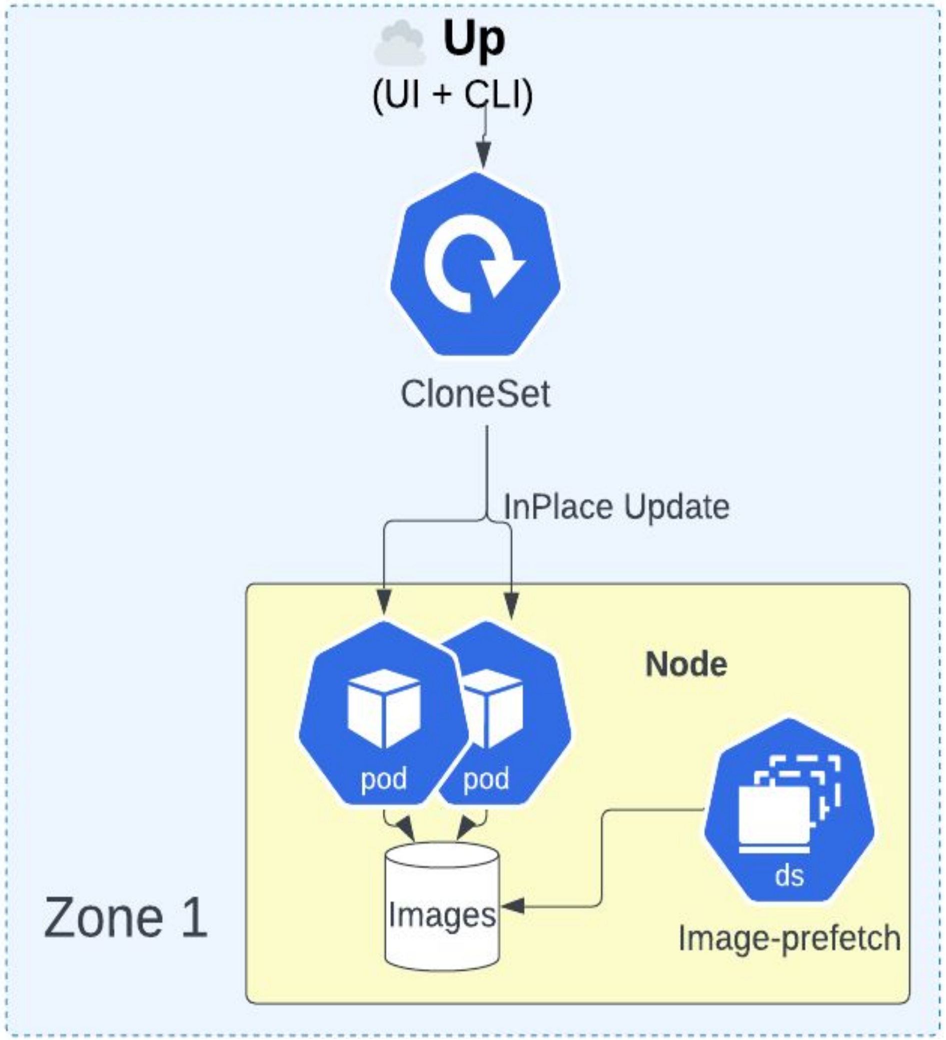
- Rolling update spec
  - only upgrades ❌
- Horizontal pod autoscaler config
  - only for autoscaling ❌
  - not intent based, but demand (metrics) based ❌

#4: Faster Rollouts

- **Problem**
  - Placement delays for large containers
  - Image fetch delays for large images

- **Solution:**
  - Placement delays: Clonesets
    - In-place updates
    - Avoid rescheduling pods (larger the pod, harder the placement)

  - Image Prefetch
    - Image fetch daemon fetches new image for currently running pods



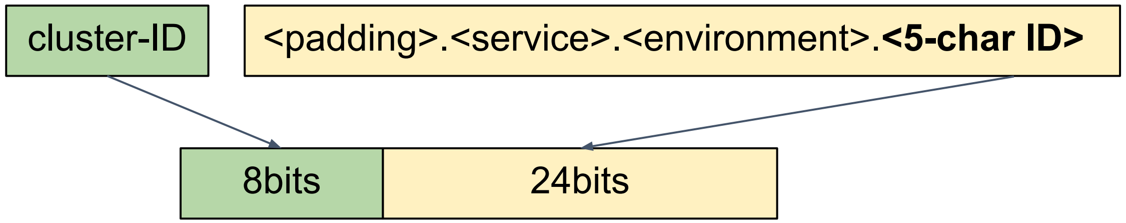
#5: Unique Instance IDs

Problem

- Services expect uint32 unique ID for a pod per (service, environment, cluster)
  - Per instance metrics (CPU, Mem etc.)
  - Per instance Logs
  - Networking (sharding, debuggability etc.)
- Use Pod-IDs (last 5 random characters)?
  - They’re random but not unique

Solution

- Make service+environment part of pod name to ensure uniqueness within that scope (Hack!!!)


cluster-ID

<padding>.<service>.<environment>.**<5-char ID>**

8bits

24bits

- **Community ask**: is it possible to provide a unique ID within a label scope?

State-of-the-art Benchmarking

Validated Scale

- K8S scales well with minimal code changes + tweaks to control plane settings
- 7500 nodes
- 200k pods
- 150 pods launched per sec

Cluster Setup

- Modified Kubemark + Clusterloader to support host network
- Bootstrap cluster with ~100 real Kubelet nodes
- Run 50-100 virtual kubelets per node
- Separate Benchmarking control plane (don’t run control plane as K8S pods)
- Dedicated 48-core hosts for control plane
- Etcd:
  - NVMe SSD hosts
  - Separate events cluster
  - “quota-backend-bytes” set to 8G

Software/Config changes

- Scheduler optimizations to improve throughput for pod topology spread
- Controller-manager / scheduler api rate limit / burst settings to 300
- Switch from default json to proto (improved LIST performance)

Migration Learnings

Cluster Health

- No visibility on fragmentation or noisy neighbors
- More vulnerable to degraded hosts
- Need better explicit reconciliation
  - Lost status changes
  - Orphaned pods & PDBs
  - Failed pods
- Our usage (make-before-break) of K8S is more vulnerable to fragmentation issues

Slow Rollbacks

- ProgressDeadlineSeconds (PDS) doesn’t work well for Uber
- Need deterministic rollbacks (eg: 10% containers crashed > 5 times, rollback the deploy)

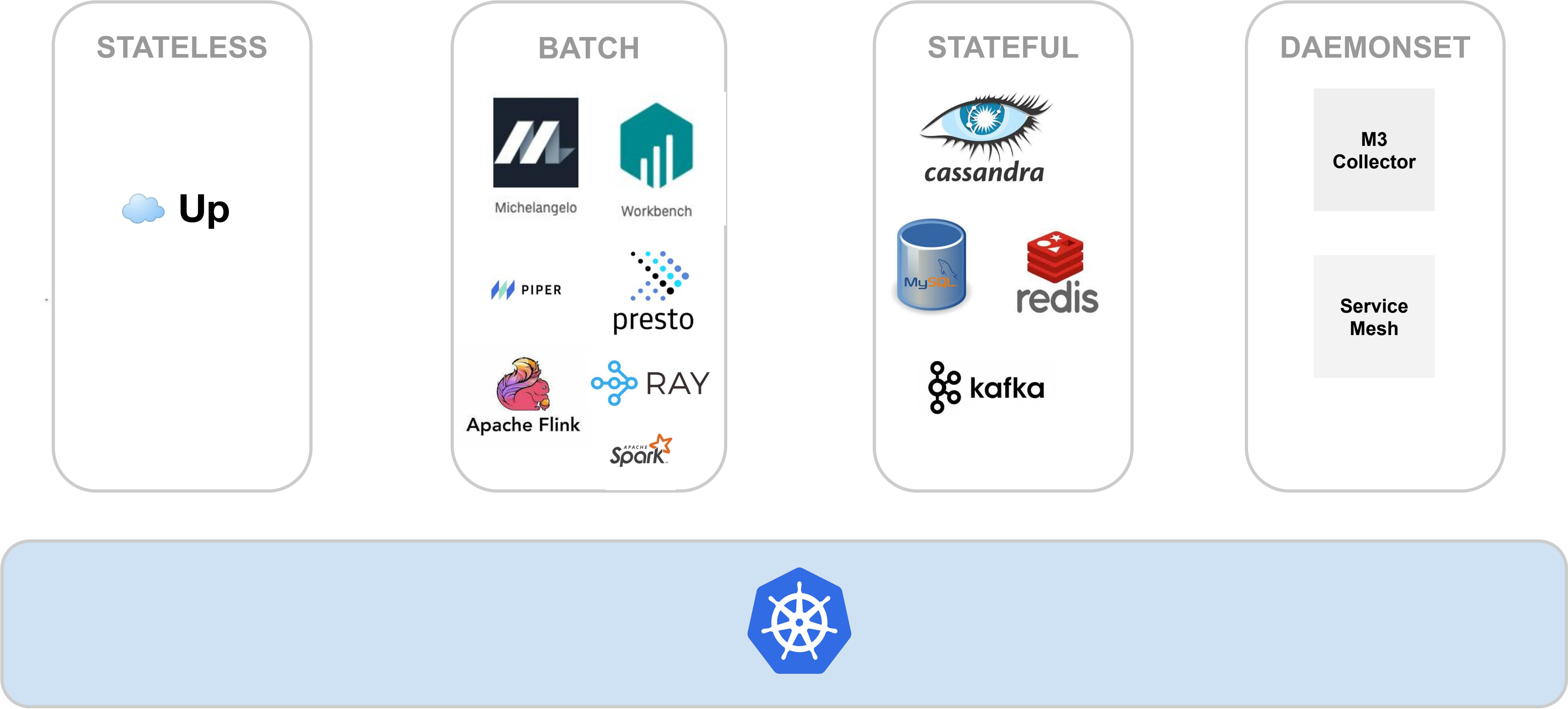
Health Check quirks

- Health check differences between Peloton & K8S cause delays (initial delay seconds)
- Kubelet restarts (marks node not ready momentarily)

Speed of migration

- Global federation + portable services are a game changer
- At peak, we moved ~250k+ cores per week

What's next?



STATELESS

Up

BATCH

STATEFUL

DAEMONSET

M3 Collector

Service Mesh

Acknowledgements

Container Platform

Service Lifecycle

- Up
- Michelangelo (Uber AI teams)
- Software Networking

Host Lifecycle

- Foundations Engineering
- Capacity Engineering

Security

- Workload identity
- Secrets & PKI infrastructure

Observability

- M3 (metrics)
- Logging

Q&A

Please scan the QR Code above
to leave feedback on this session