Adi Muraru

Running Kafka on Kubernetes, across three clouds at Adobe

Who I am ?

- Principal Scientist with Adobe Experience Platform
  - Been with Adobe for 10yrs
  - Big Data lead engineer for Hadoop, HBase and Kafka infrastructures
  - Led the transition of Pipeline Kafka to Kubernetes/Ethos

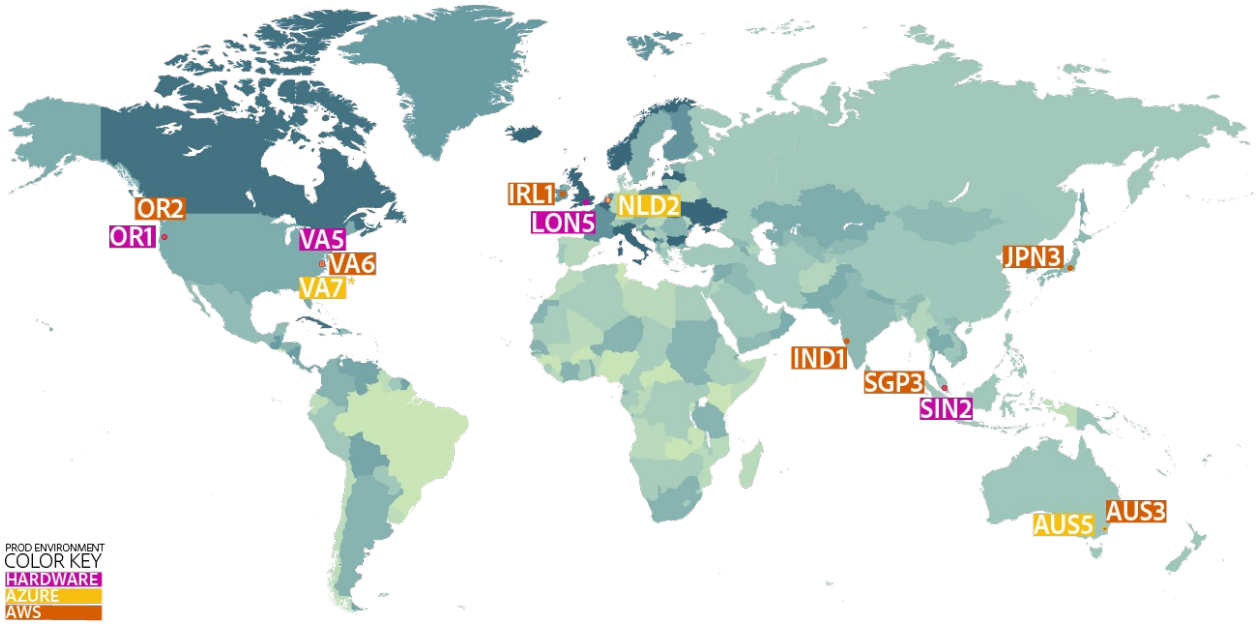
The road to running Adobe Kafka on Kubernetes

- What is Adobe Experience Platform Pipeline?
- Pipeline “v2” guiding principles
- Kafka Operator
- Mechanical Sympathy – Build for resiliency
- Lessons & considerations

Adobe Experience Platform Pipeline

- Kafka Based Message Bus
- Global deployment
  - 14 Regions (Private Cloud, AWS, Azure)
  - 25 production Clusters
- Scale
  - 300+ Billion events
  - 250+ TB / day IN
  - 800+ TB / day OUT
  - 1.7 PB stored

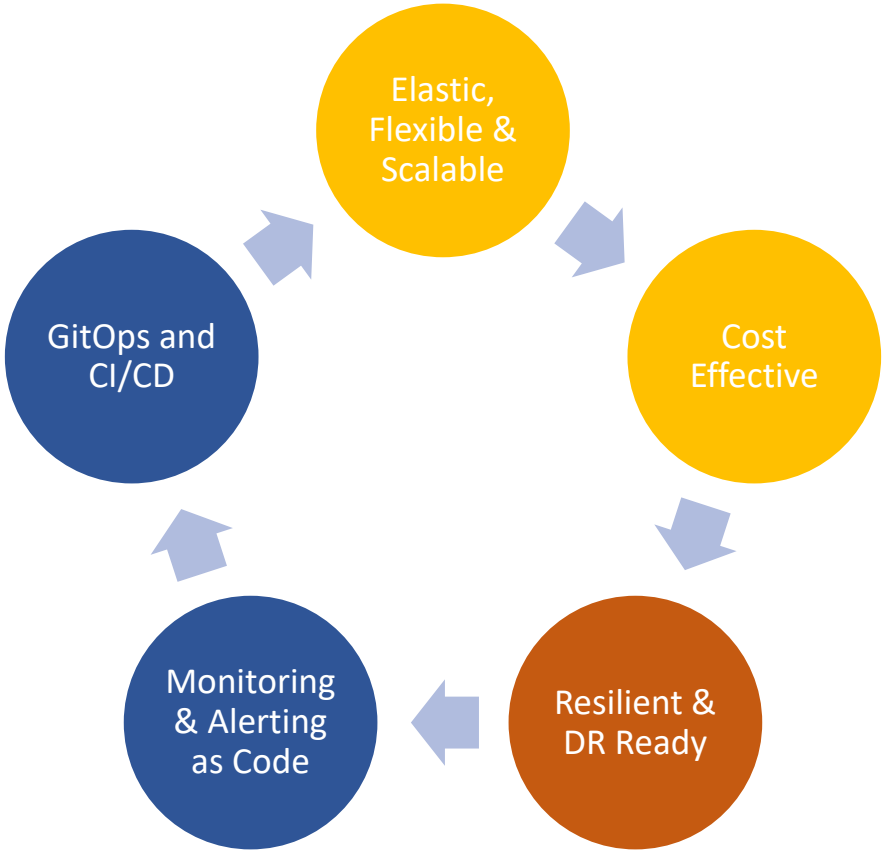
Pipeline Service Locations



Migration to Kubernetes

*A **uniform** automated deployment of Pipeline **across different clouds***

Cloud native, provider neutral guiding principles



KOperator

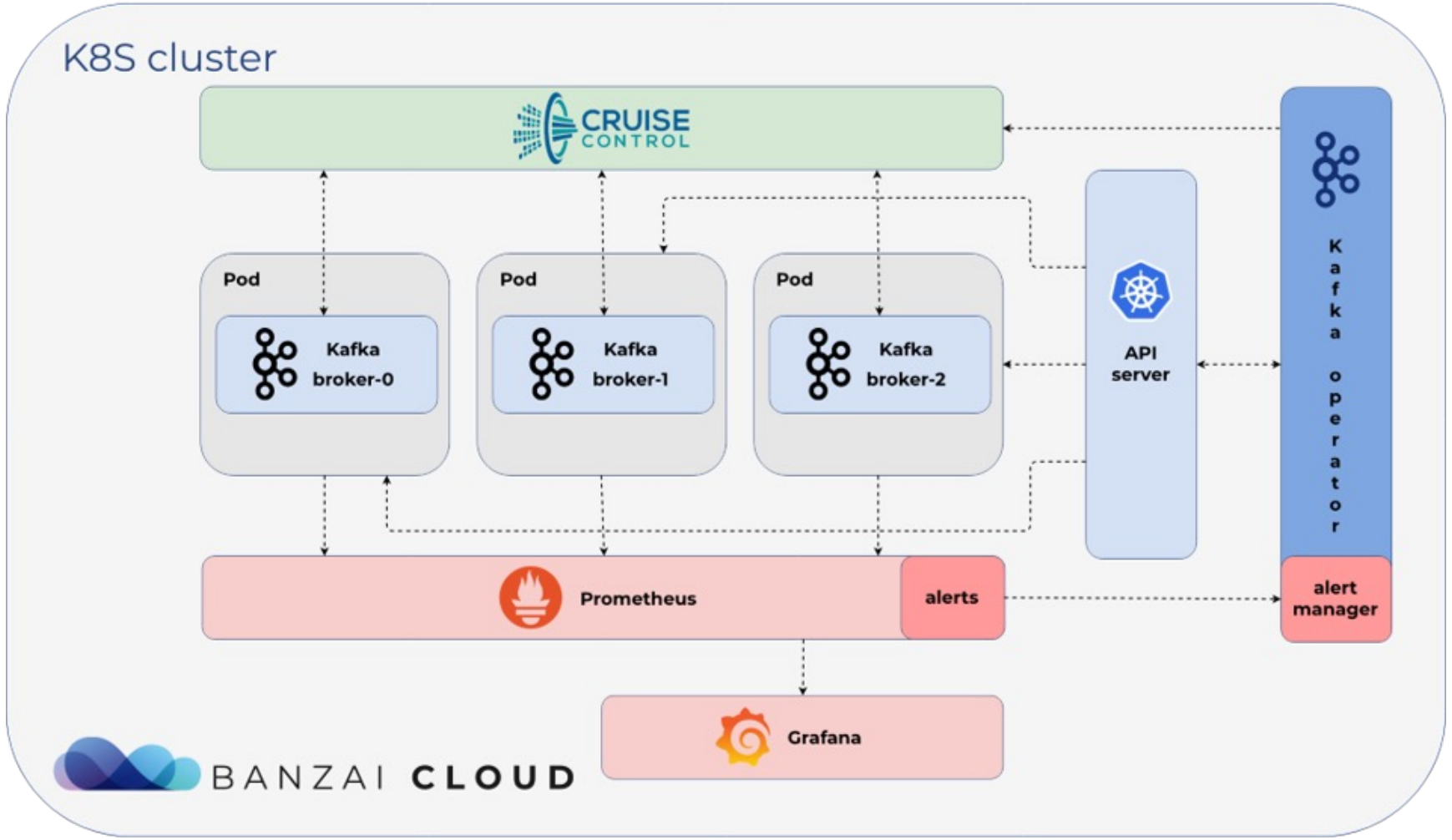
OPERATOR CAPABILITY LEVELS

Operators come in different maturity levels in regards to their lifecycle management capabilities for the application or workload they deliver. The capability models aims to provide guidance in terminology to express what features users can expect from an Operator.

Level I	Level II	Level III	Level IV	Level V
Basic Install	**Seamless Upgrades**	**Full Lifecycle**	**Deep Insights**	**Auto Pilot**
Automated application provisioning and configuration management	Patch and minor version upgrades supported	App lifecycle, storage lifecycle (backup, failure recovery)	Metrics, alerts, log processing and workload analysis	Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning

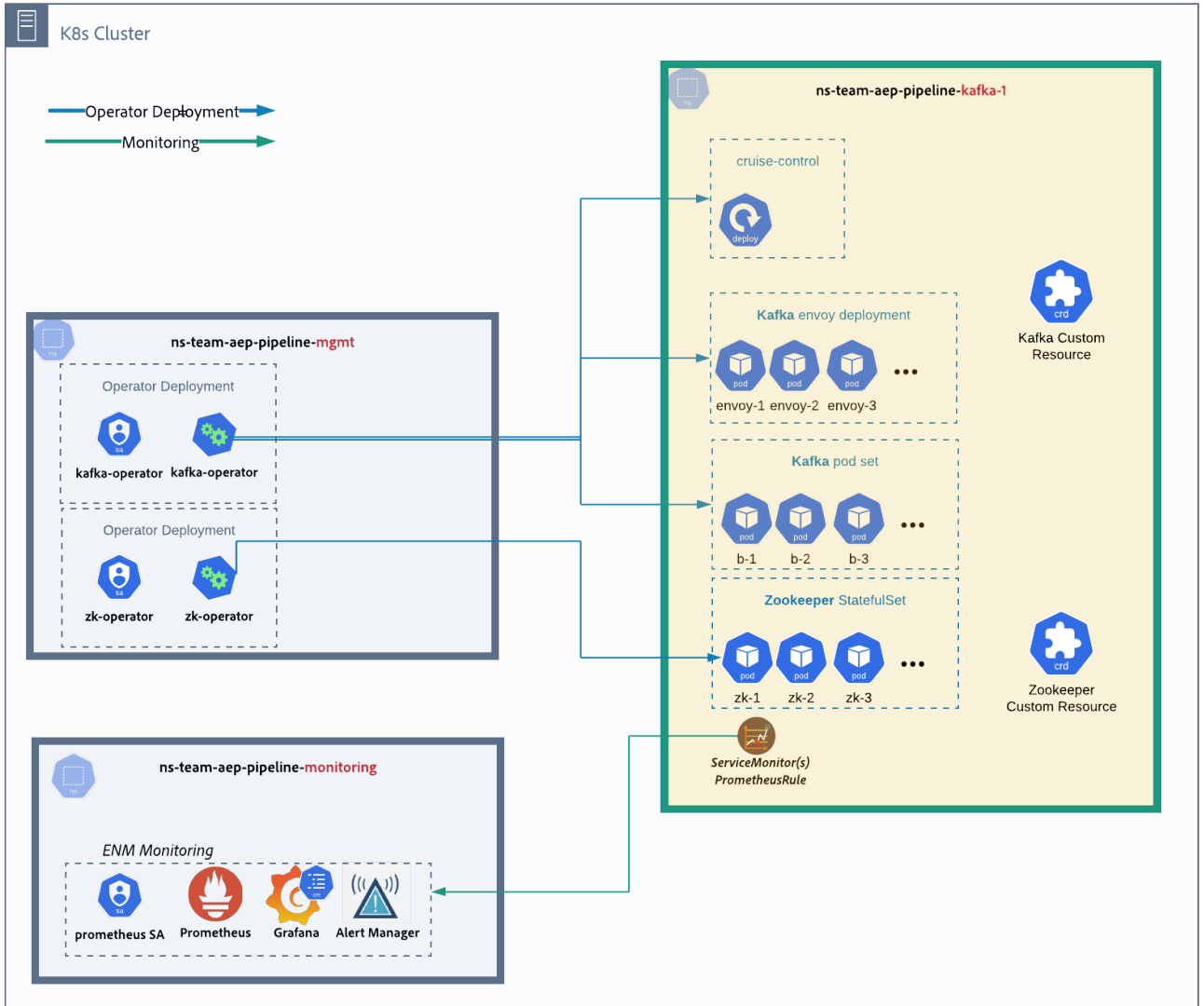
https://operatorframework.io/operator-capabilities/

Architecture

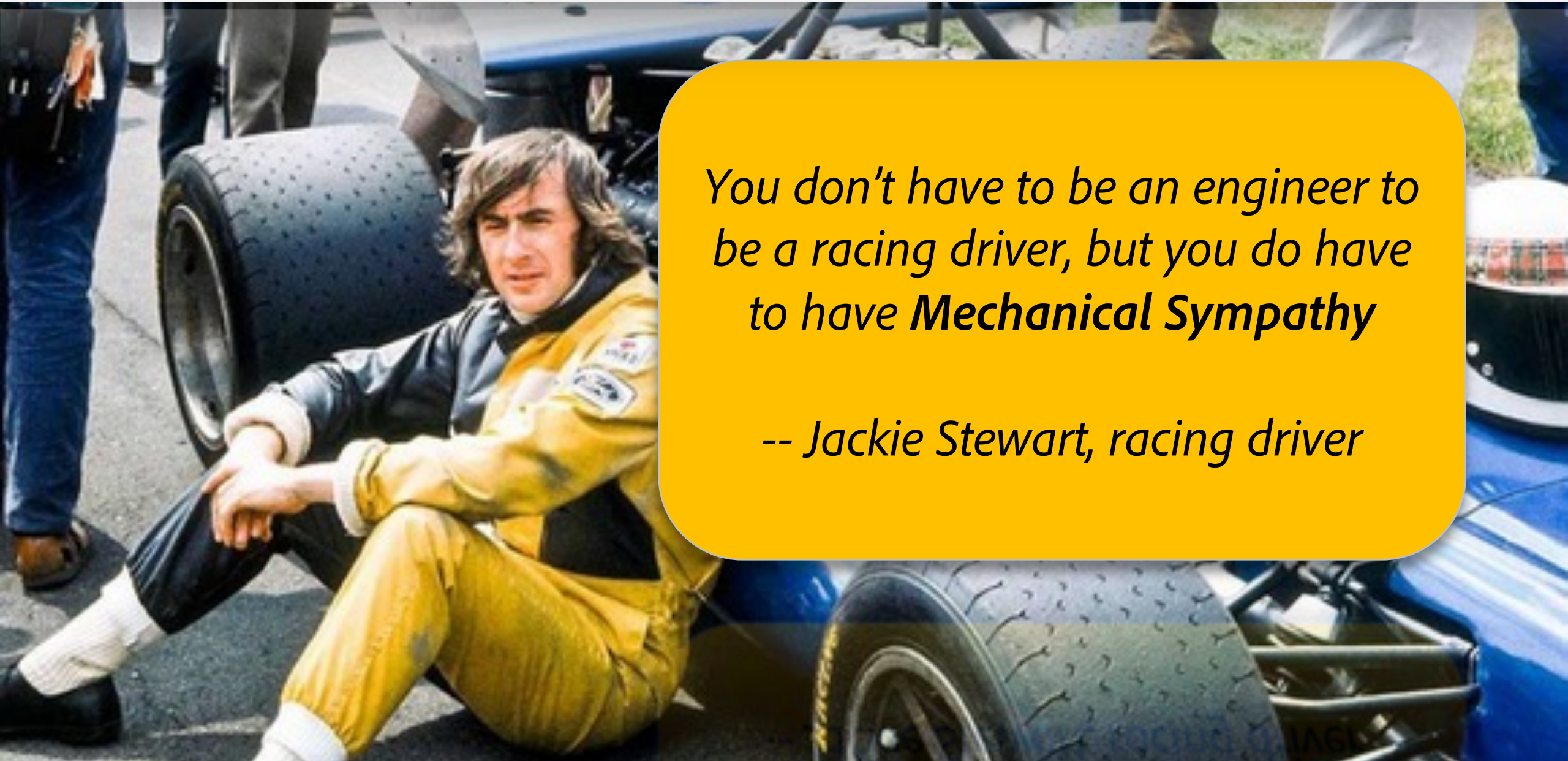


https://github.com/banzaicloud/kafka-operator

Deployment


K8s Cluster
Operator Deployment
Monitoring
ns-team-aep-pipeline-kafka-1
cruise-control
deploy
Kafka envoy deployment
pod
pod
pod
envoy-1 envoy-2 envoy-3
Kafka pod set
pod
pod
pod
b-1 b-2 b-3
Zookeeper StatefulSet
pod
pod
pod
zk-1 zk-2 zk-3
crd
Kafka Custom Resource
crd
Zookeeper Custom Resource
ServiceMonitor(s)
PrometheusRule
ns-team-aep-pipeline-mgmt
Operator Deployment
kafka-operator kafka-operator
Operator Deployment
zk-operator zk-operator
ns-team-aep-pipeline-monitoring
ENM Monitoring
prometheus SA
Prometheus
Grafana
Alert Manager

Mechanical Sympathy – Build for Resiliency



*You don't have to be an engineer to be a racing driver, but you do have to have **Mechanical Sympathy***

-- Jackie Stewart, racing driver

Lessons & Considerations

- `PersistentVolumeClaim`s and `StorageClass`es
  - Backed by Cloud managed disks (EBS, AzureDisk, Vsphere disks)
  - Be mindful of cloud specifics, e.g. Volumes larger than or equal to 334 GiB deliver 250 MiB/s regardless of burst credits

- `Immediate` and `WaitForFirstConsumer` volume binding mode
  - WFFC recommended - delays the disk binding to Node until Pod is a scheduled on a node

Lessons & Considerations

- Use Availability Zones to map to Kafka “racks”
  - Via topology.kubernetes.io/zone nodeSelector

- Use podAntiAffinity to avoid two Kafka brokers pods on the same Node
  - Avoid IO contention
  - Or consider dedicated *worker pools*

Lessons & Considerations

- Consider setting requests==limits to ensure pod gets **Guaranteed** scheduling class

- Implement clean container shutdown hooks, e.g PreStop and terminationGracePeriodSeconds

Lessons & Considerations

- If your workload exploits page-cache to I/O make sure you account for that in container requests/limits
  - For Kafka we set JVM heap 70%, the rest goes to page-cache
- Account for other pods on K8s Node contending for IO with yours
  - K8s does not (yet?) consider Node/VM IO capacity as a kubelet allocatable resource
  - Workaround for us was to overprovision the Kafka pod via CPU/MEM to avoid noisy neighbours
- Monitor slow disks
  - Instrument and Monitor latencies
  - Act on them

Lessons & Considerations

- Be mindful of cluster auto-scaler
  - This includes scale in (use cluster-autoscaler.kubernetes.io/safe-to-evict=false annotation to alleviate)

- Evaluate *Automatic Resource Configuration* controllers in cluster
  - Auto-tunes container requested resources
  - Optimizer but may conflict with your own capacity management

Lessons & Considerations

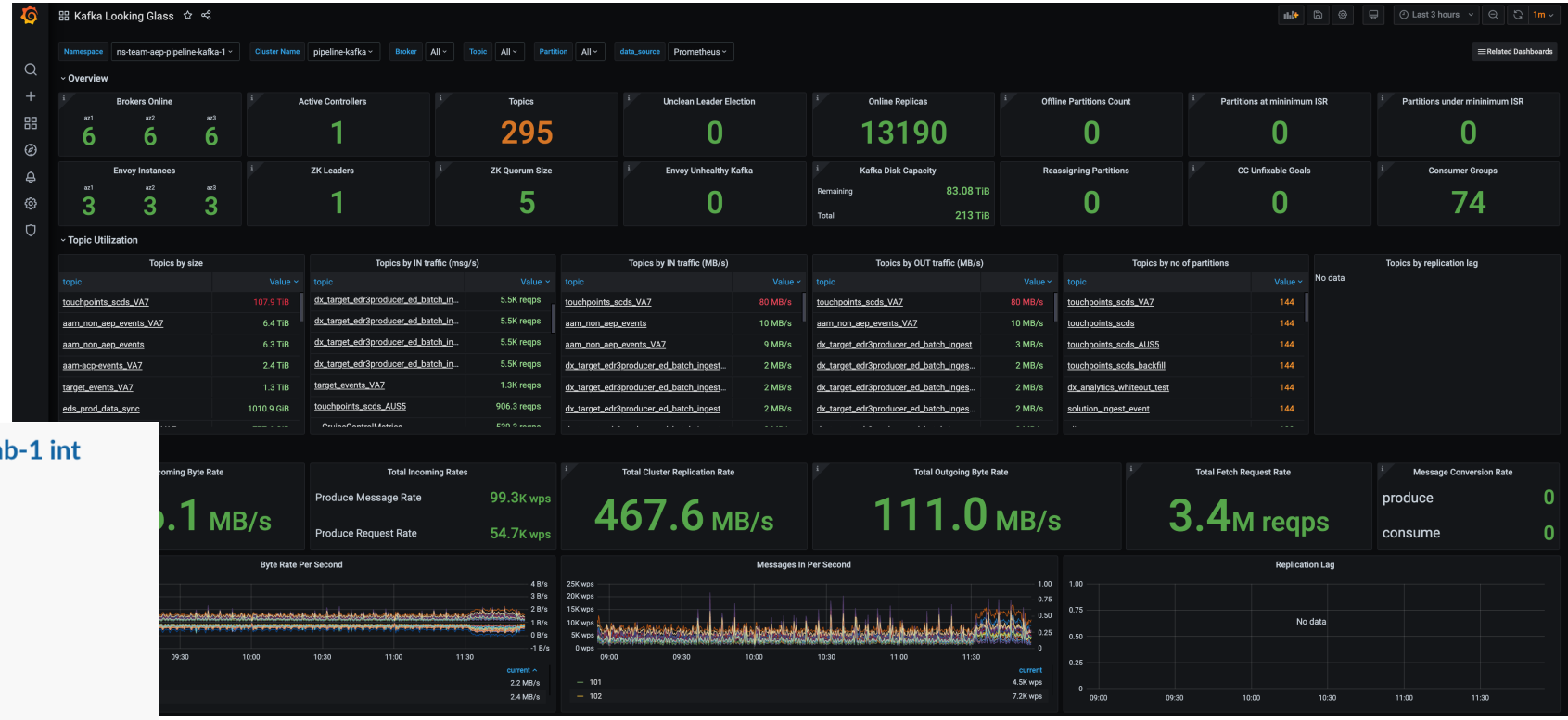
- Tolerate Kubernetes cluster maintenance
  - Resources updates
  - Full cluster upgrades (nodes draining)
  - Have readiness/liveness probes

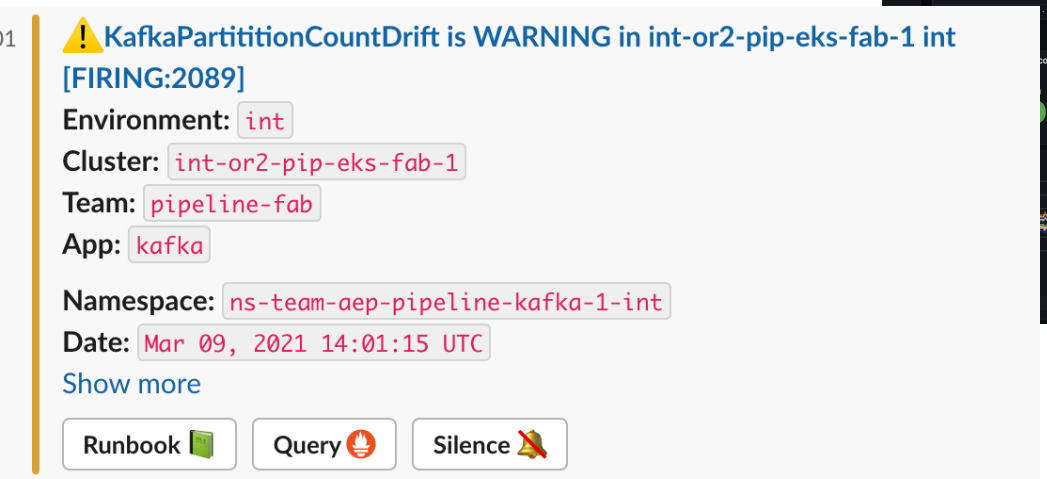
- PodDisruptionBudget
  - Kafka operator sets this dynamically based on number of brokers

- Last but least, have an off-cluster backup and disaster recovery strategy
  - Mirror Kafka topics to blob storage
  - Replicate topics to multiple regions

Lessons & Considerations

- Metrics, metrics, metrics … and alerts
  - Monitors as code
  - Alert rules as code
  - Dashboards as code





Thank you!