Running Kafka on Kubernetes with Strimzi

Sean Glover, Lightbend
@seg1o

Lightbend

Who am I?

I'm Sean Glover

- Principal Engineer at Lightbend
- Member of the Lightbend Pipelines team
- Organizer of Scala Toronto (scalator)
- Author and contributor to various projects in the Kafka ecosystem including Kafka, Alpakka Kafka (reactive-kafka), Strimzi, Kafka Lag Exporter, DC/OS Commons SDK

/ seg1o

https://seanglover.com/
sean@seanglover.com

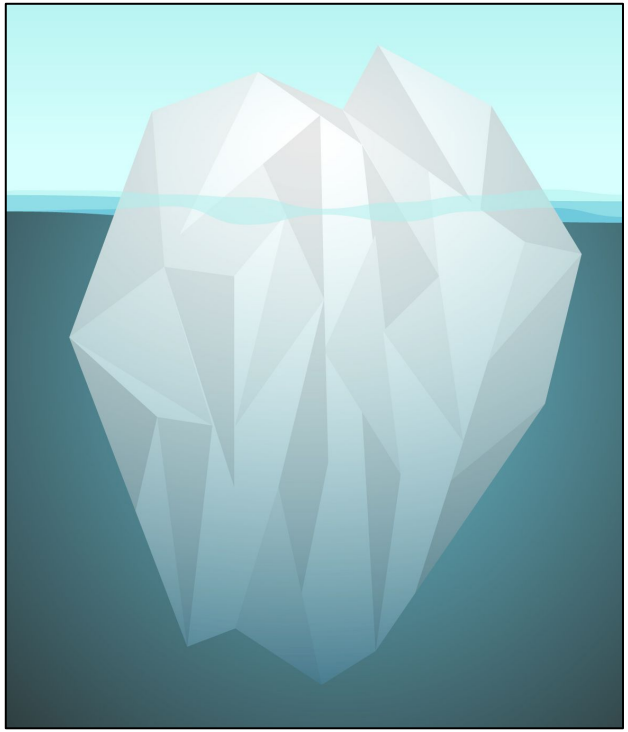
Operations Is Hard

STRIMZI

“Technology will make our lives easier”

Technology makes running other technology easier

Automate as much operations work as we can



Designed by Freepik

Motivating Example:
Zero-downtime Kafka Upgrade

Motivating Example: Upgrading Kafka

High level steps to upgrade Kafka

1. Rolling update to explicitly define broker properties `inter.broker.protocol.version`and `log.message.format.version`
2. Download new Kafka distribution and perform rolling upgrade 1 broker at a time
3. Rolling update to upgrade `inter.broker.protocol.version`to new version
4. Upgrade Kafka clients
5. Rolling update to upgrade `log.message.format.version`to new version

Motivating Example: Upgrading Kafka

Any update to the Kafka cluster must be performed in a serial “rolling update”. The complete Kafka upgrade process requires 3 “rolling updates”

Each broker update requires

- Secure login
- Configuration linting - Any change to a broker requires a rolling broker update
- Graceful shutdown - Send SIGINT signal to broker
- Broker initialization - Wait for Broker to join cluster and signal it’s ready

This operation is **error-prone to do manually** and **difficult to model declaratively** using generalized infrastructure automation tools.

Automation

“If it hurts, do it more frequently, and bring the pain forward.”

- Jez Humble, Continuous Delivery

Automation of Operations

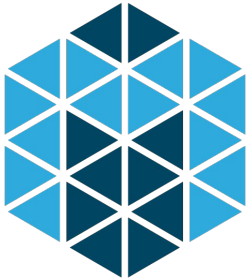
Upgrading Kafka is just one of many complex operational concerns. For example)

- Initial deployment
- Manage ZooKeeper
- Replacing brokers
- Topic partition rebalancing
- Decommissioning or adding brokers

How do we automate complex operational workflows in a reliable way?

Container Orchestrated Clusters

Cluster Resource Managers



MESOS

DC/OS

kubernetes

Task Isolation with Containers

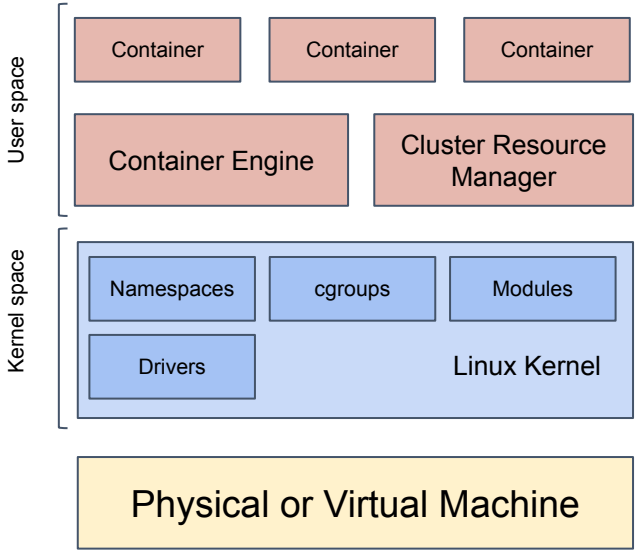
- Cluster Resource Manager's use **Linux Containers** to constrain resources and provide isolation
- **cgroups** constrain resources
- **Namespaces** isolate file system/process trees
- Docker is just a project to describe and share containers efficiently (others: rkt, LXC, Mesos)
- Containers are available for several platforms

Jail

Linux Container

Windows Container

Linux Containers (LXC)



Kubernetes and the Operator Pattern

kubernetes

The Operator Pattern

1. Controller/Operator

```
// Active Reconciliation Loop
for {
  desired := getDesiredState()
  current := getCurrentState()
  makeChanges(desired, current)
}
```

2. Configuration State

watches CRUD changes

"Kafka" Custom Resource

```
apiVersion: kafka.strimzi.io/v1alpha1
kind: Kafka
metadata:
  name: simple-strimzi
spec:
  kafka:
    config:
      ...
```

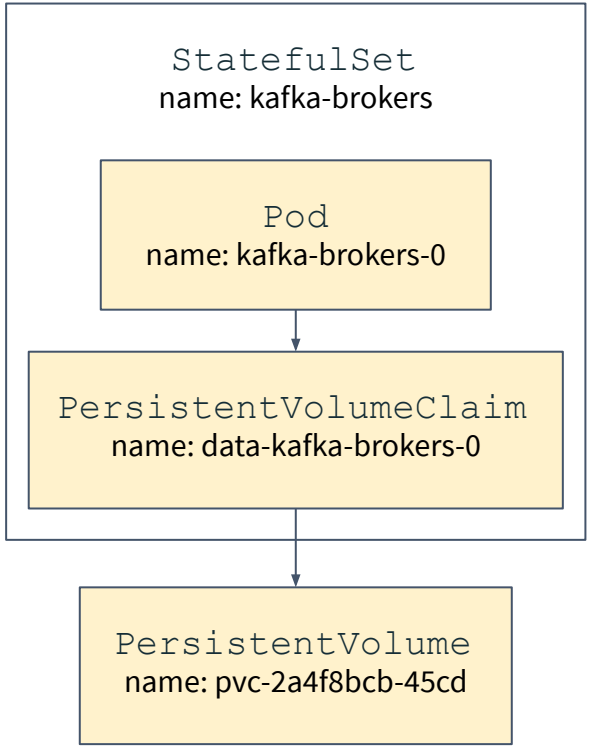
deploy reconciliation plan

Kafka Cluster

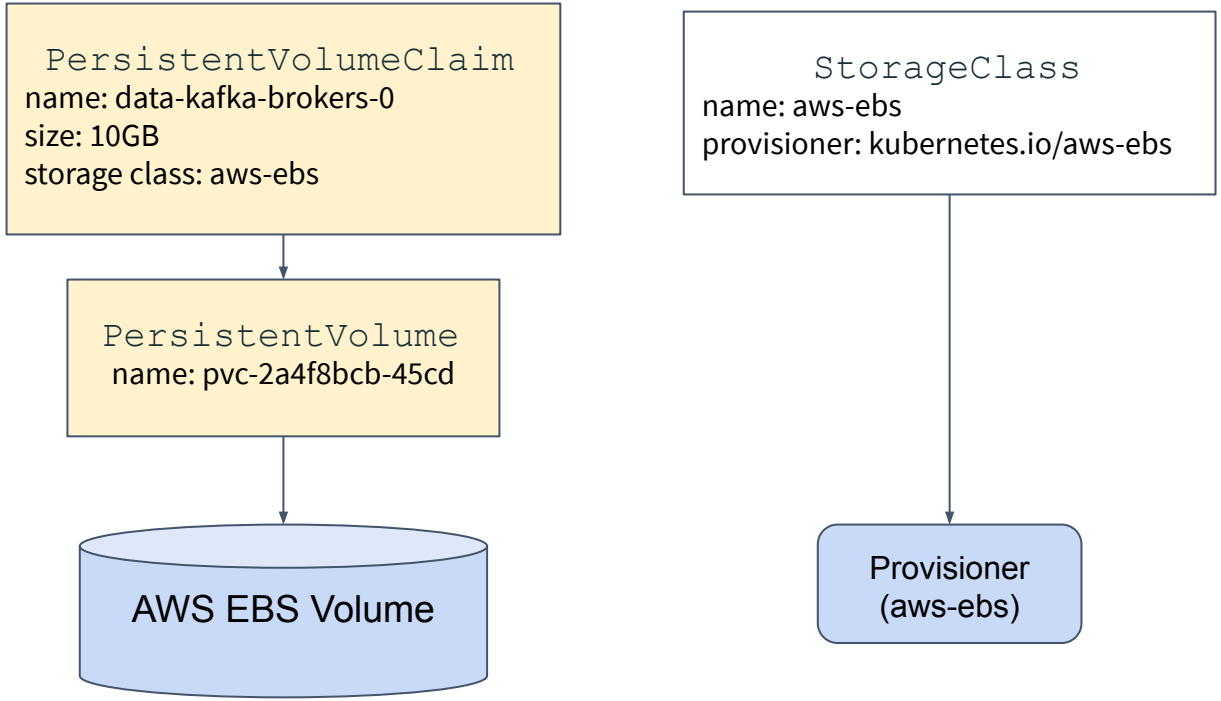
Stateful Services in Kubernetes

`StatefulSet`'s

- Stable pod & network identity
- Stable persistent storage
- Ordered deployment and updates
- Ordered graceful deletion and termination
- Ordered automated rolling updates.



Abstracting Persistence



Strimzi

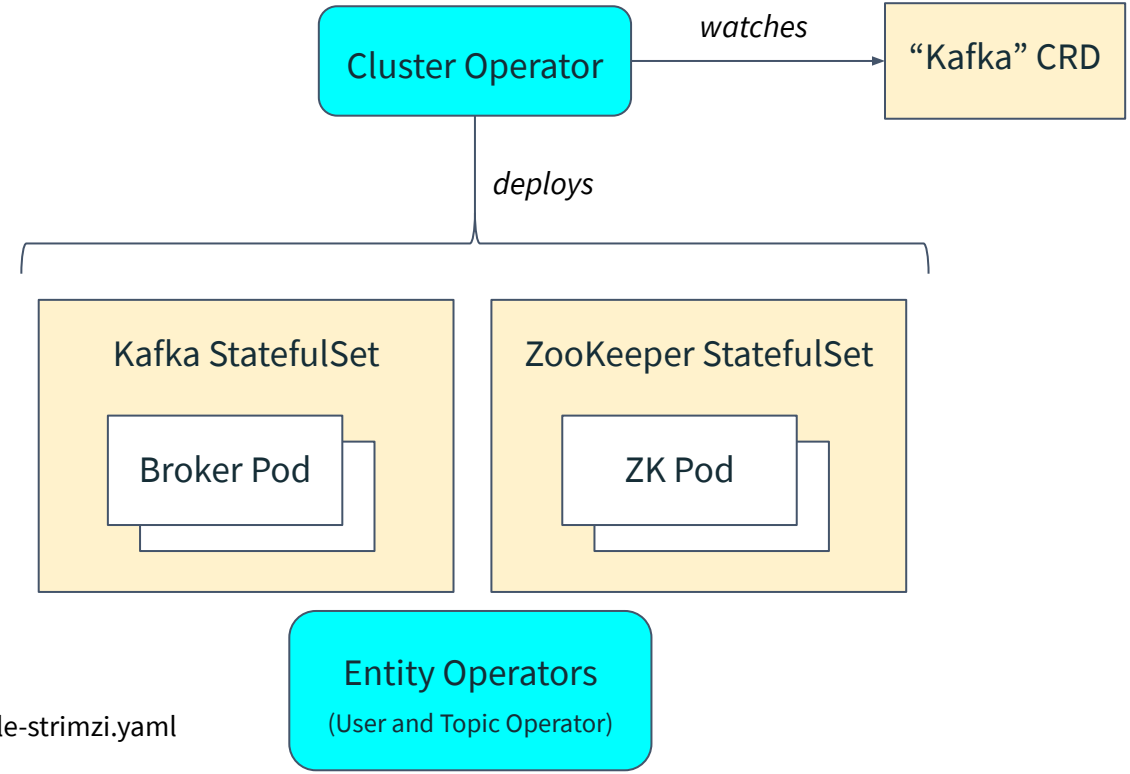
An operator-based Kafka on Kubernetes project

Strimzi

Strimzi is an open source **operator-based** Apache Kafka project for Kubernetes and OpenShift

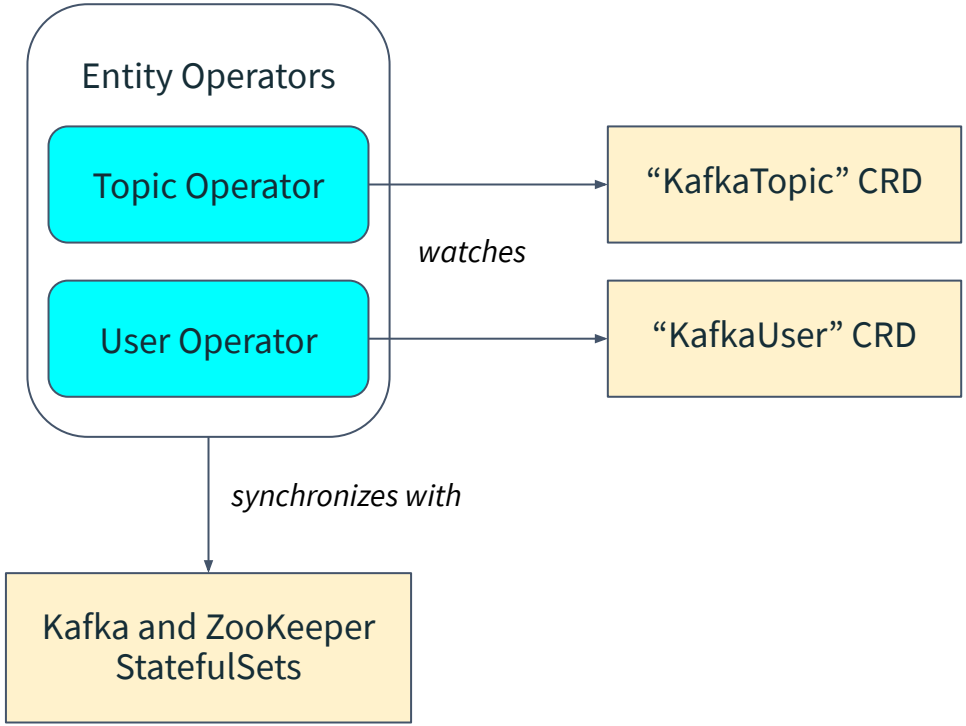
- Announced Feb 25th, 2018
- Evolved from non-operator project known as Barnabas by Paolo Patierno, Red Hat
- Part of Red Hat Developer Program
- “Streams” component of Red Hat AMQ, a commercial product of messaging technologies by Red Hat

Cluster Operator



Demo: ./resources/simple-strimzi.yaml

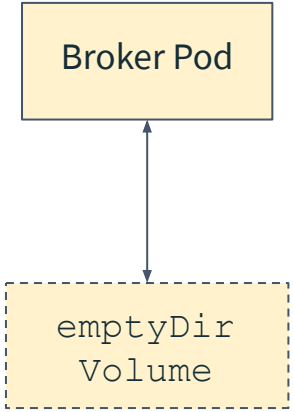
Entity Operator (User and Topic Operators)



Demo: ./resources/simple-topic.yaml

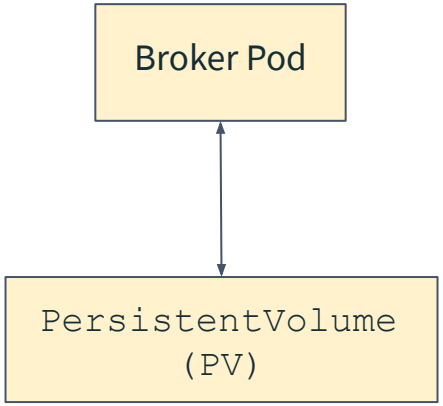
Strimzi Storage Modes

1. Ephemeral



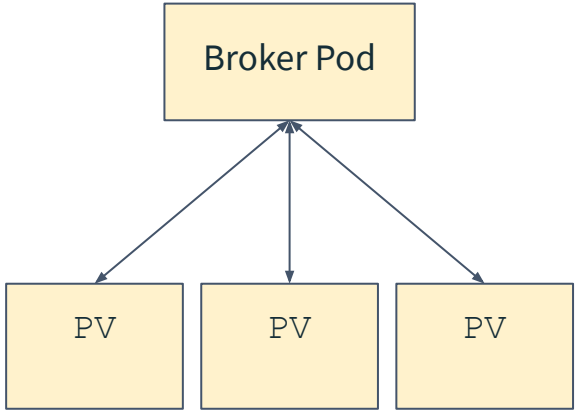
transient

2. Persistent



persistent

2 (b). Persistent JBOD



persistent

Broker config

```
log.dirs = [PV1, PV2, PV3]
```

Operational Concerns

Install Strimzi

Installation and running a Strimzi Kafka cluster is a two step process.

1. Install the Strimzi Helm Chart
2. Create a Kafka Kubernetes resource

Helm Chart Install:

```
helm repo add strimzi http://strimzi.io/charts/
helm install strimzi/strimzi-kafka-operator
```

Demo: ./demo/01-create-simple-strimzi-cluster.sh

Connecting Clients

Fully qualified service hostname:

```
simple-strimzi-kafka-bootstrap.strimzi.svc.cluster.local:9092
```

- `simple-strimzi` — Kafka resource metadata.name
- `kafka-bootstrap` — Broker load balancer name
- `strimzi` — Namespace
- `svc` — K8s Service
- `9092`

“Plain”	9092
TLS	9093
Interbroker	9094
Prometheus	9404

Demo: ./demo/02-connecting-clients.sh
run-kafka-perf-producer.sh

Rolling Configuration Updates

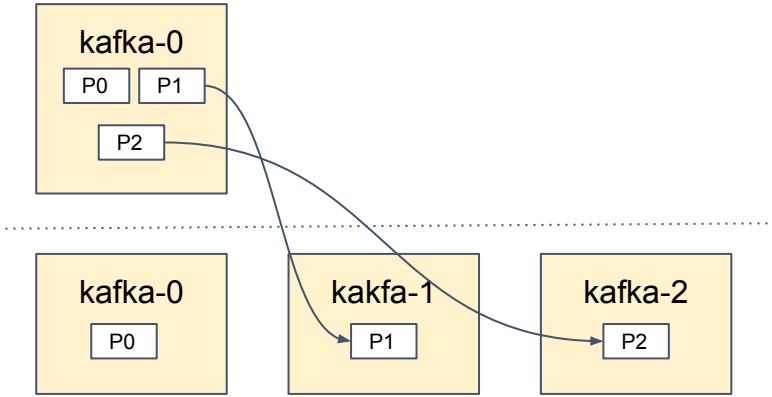
Rolling Configuration Process

1. Watched Kafka resource change
2. Apply new config to Kafka StatefulSet spec
3. Starting from pod 0, delete the pod and allow the StatefulSet to recreate it
4. Kafka pod will generate new broker.config
5. Kafka is started
6. Wait until the readiness check is good.
7. Repeat from step 3 for the next pod

Demo: ./demo/03-broker-config-update.sh

Scaling Brokers Up

1. Increase replica count `spec.kafka.replicas`
2. Reassign partitions: `./bin/kafka-reassign-partitions.sh`



Demo: ./demo/04-scale-brokers.sh
./partition-reassignment/generate-plan-output.json

Rolling Broker Upgrades

Rolling Broker Upgrade Process:

1. Upgrade Strimzi Cluster Operator
2. Update config:
   a. (Optional) Set `log.message.format.version` broker config
   b. Set desired Kafka release version

Rolling Updates (1-2x)

3. (Optional) Upgrade clients using cluster
4. (Optional) Set `log.message.format.version` broker config

Rolling Update (0-1x)

Broker Replacement & Movement

Replacing brokers is common with large busy clusters

```
$ kubectl delete pod kafka-1
```

Broker replacement also useful to facilitate broker movement across the cluster

1. Research the max bitrate per partition for your cluster
2. Move partitions from broker to replace
3. Replace broker
4. Rebalance/move partitions to new broker

Broker Replacement & Movement

1. Research the max bitrate per partition for your cluster

Run a controlled test

- Bitrate depends on message size, producer batch, and consumer fetch size
- Create a standalone cluster with 1 broker, 1 topic, and 1 partition
- Run producer and consumer perf tests using average message/client properties
- Measure broker metric for average bitrate

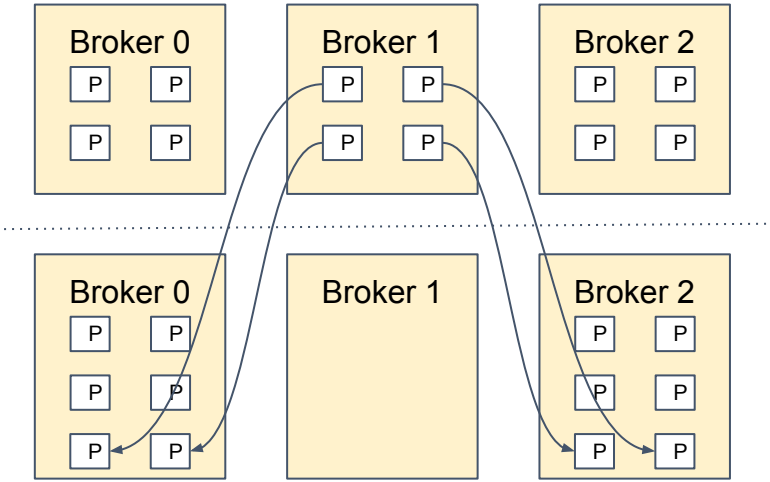
```
kafka.server:type=BrokerTopicMetrics,name=BytesInPerSec
kafka.server:type=BrokerTopicMetrics,name=BytesOutPerSec
```

Broker Replacement & Movement

2. Move partitions from broker to replace

Use Kafka partition reassignment tool

- Generate an assignment plan **without** old broker 1
- Pick a fraction of the measured max bitrate found in step 1 (Ex. 75%, 80%)
- Apply plan with bitrate throttle
- Wait till complete




```
kafka-reassign-partitions … --topics-to-move-json-file topics.json --broker-list "0,2" --generate

kafka-reassign-partitions … --reassignment-json-file reassignment.json --execute --throttle 10000000

kafka-reassign-partitions … --topics-to-move-json-file topics.json --reassignment-json-file reassignment.json --verify
```

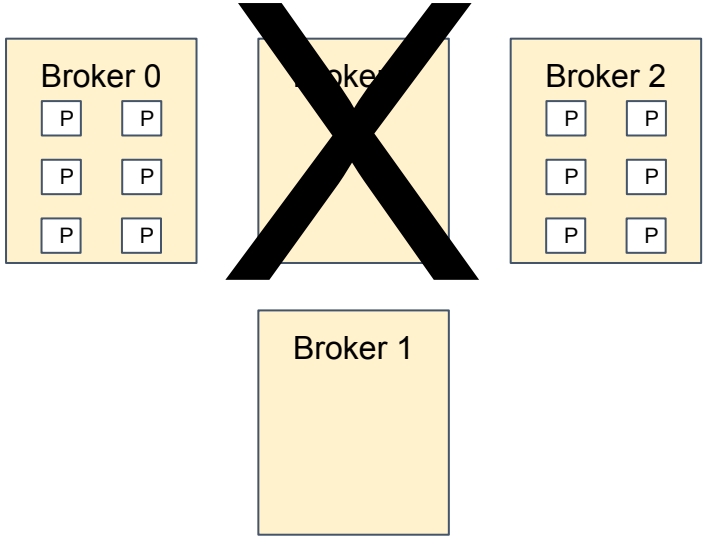
Broker Replacement & Movement

3. Replace broker

Replace broker pod instance with kubectl

```
$ kubectl delete pod kafka-1
```

- Old broker 1 instance is shutdown and resources deallocated
- Deploy plan provisions a new broker 1 instance
- New broker 1 is assigned same id as old broker 1: **1**

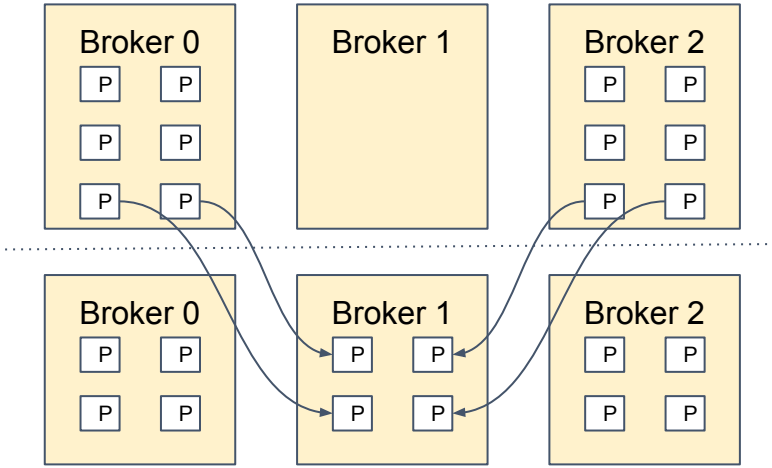


Broker Replacement & Movement

4. Rebalance/move partitions to new broker

Use Kafka partition reassignment tool

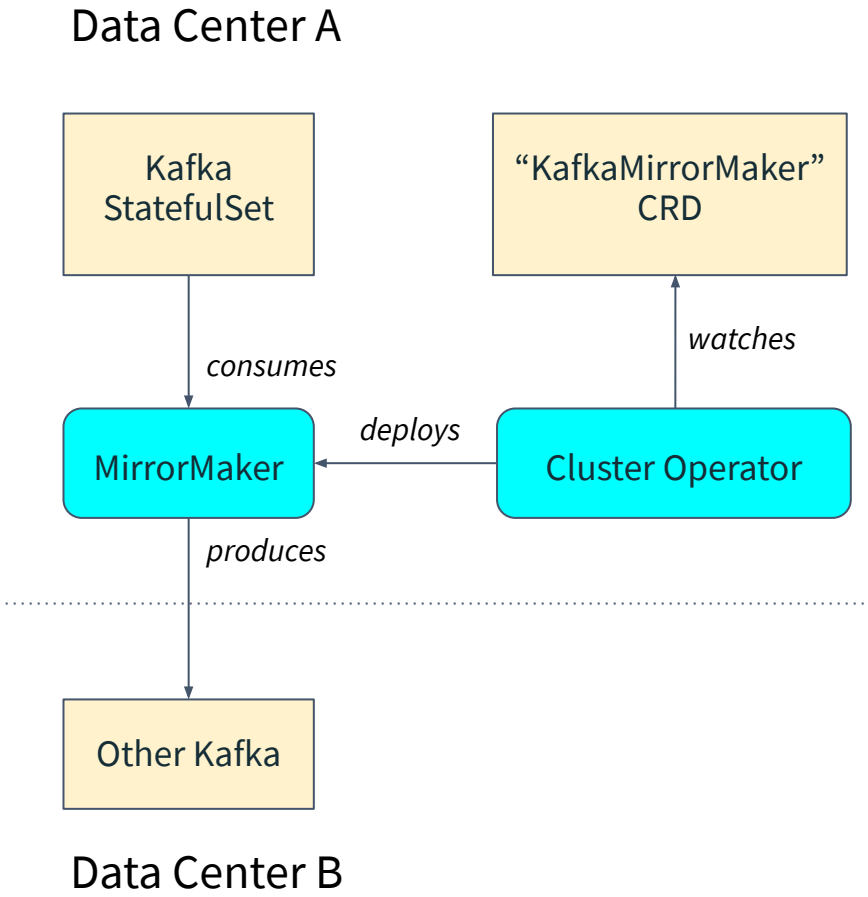
- Generate an assignment plan **with** new broker 1
- Pick a fraction of the measured max bitrate found in step 1 (Ex. 75%, 80%)
- Apply plan with bitrate throttle
- Wait till complete



MirrorMaker

Synchronize Kafka topics between clusters

- Disaster Recovery
- Multi Data Center
  - Active / Passive cluster
  - Active / Active cluster



Demo: resources/kafka-mirror-maker.yaml

Monitoring

Kubernetes

+

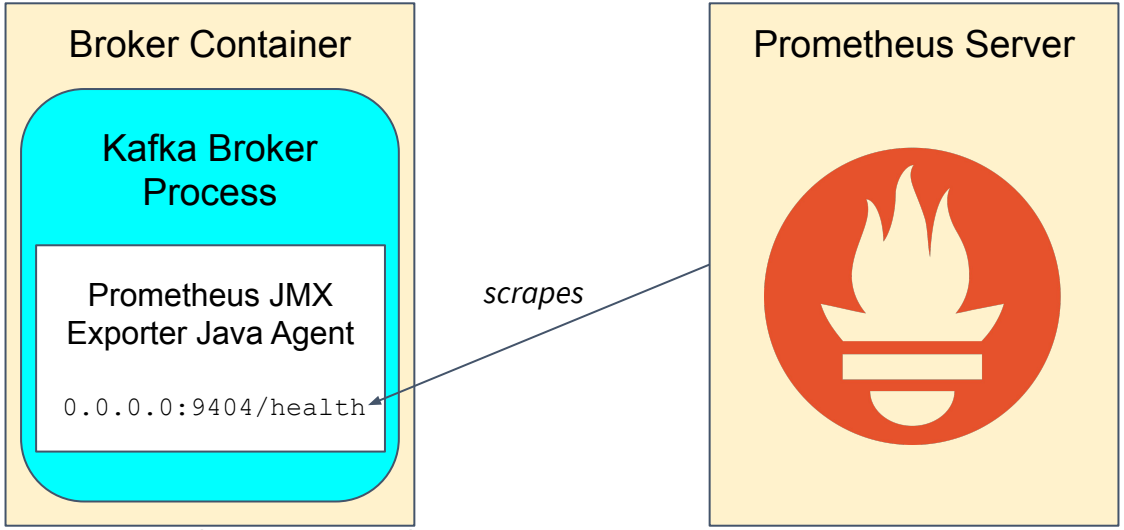
Prometheus

+

Grafana

Monitoring

Strimzi exposes a Prometheus Health Endpoint with Prometheus JMX Exporter



Demo:
"Production" Strimzi resource: ./resources/pipelines-strimzi.yaml
Grafana Dashboard

Conclusion

Is running Kafka on Kubernetes safe?

Is running Kafka on Kubernetes safe?

Pros

- Confluent cloud runs on Kubernetes clusters on Google and Amazon
- Strimzi is an open source component of a commercial product: Red Hat AMQ
- Kafka data is usually transient

Cons

⚠️ Beware of risks running `PersistentVolumes` and `StatefulSets` ⚠️

- Still need SRE's and operations knowledge in production
- More abstractions -> Harder to reason about
- Simplistic update strategies for large clusters

Strimzi Project

- Apache Kafka project for Kubernetes and OpenShift
- Licensed under Apache License 2.0
- Considered stable as of 0.8.2 release (0.11.4 current)
- Web site: http://strimzi.io/
- GitHub: https://github.com/strimzi/strimzi-kafka-operator
- Slack: strimzi.slack.com
- Mailing list: strimzi@redhat.com
- Twitter: @strimziio

One More Thing...

Kafka Lag Exporter

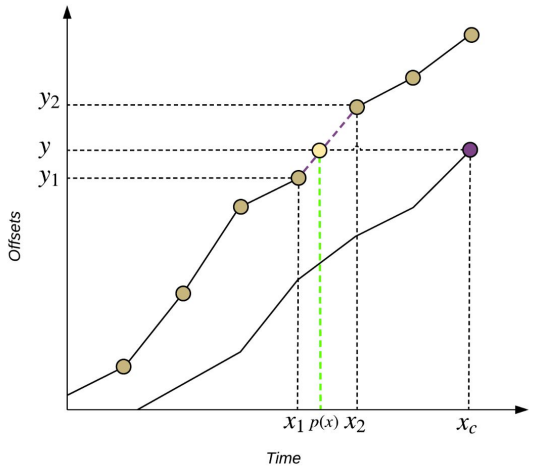
Monitor Kafka Consumer Group Latency and Lag of Apache Kafka applications

Main features include

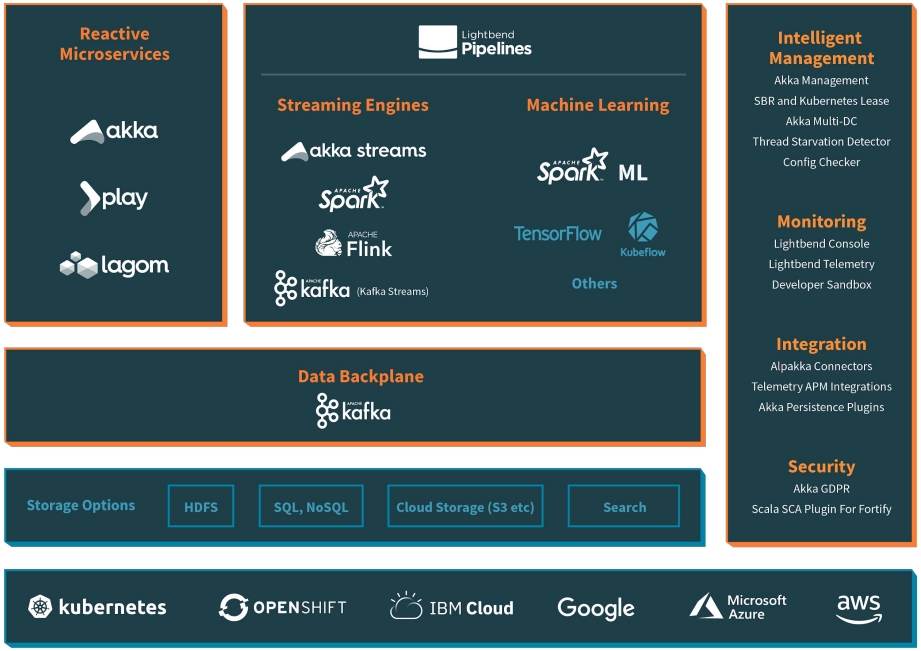
- Report group and partition metadata as Prometheus metrics
- Estimate consumer group latency in time
- Auto-discovery of Strimzi Apache Kafka clusters
- Installed as a Helm chart

GitHub repo: https://github.com/lightbend/kafka-lag-exporter

Blog post: https://bit.ly/2Jzvg8p



Lightbend Platform



https://www.lightbend.com/lightbend-platform

Lightbend Pipelines

Lightbend

Thank You!

Sean Glover

@seg1o

in/seanaglover

sean.glover@lightbend.com

Free eBook!

https://bit.ly/2J9xmZm