Deploying Kafka for ServiceX

Ilija Vukotic
SSL Monthly Call
Jan 10, 2020

Task

- Investigate ways to deploy Kafka for ServiceX
- Benchmark it
- Document it

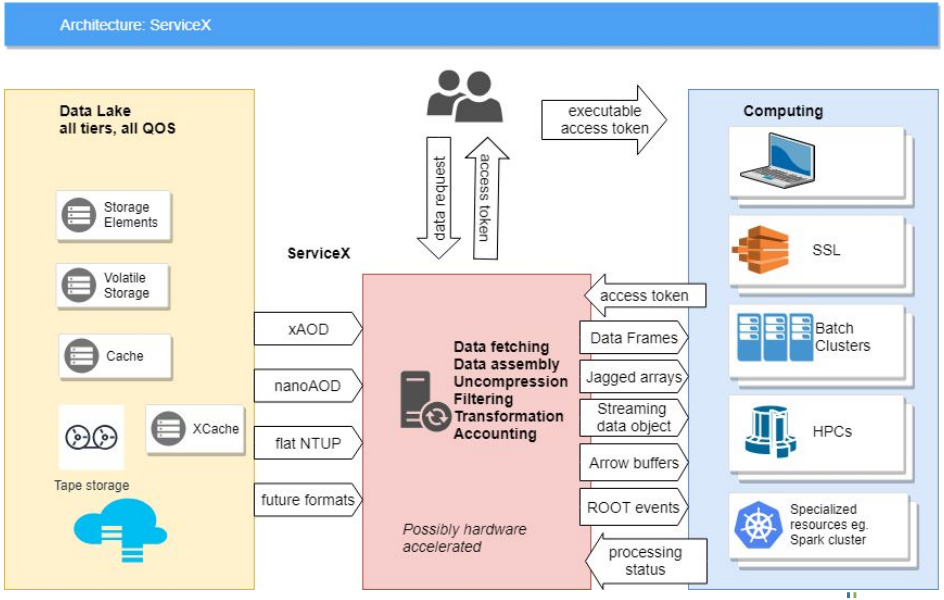
ServiceX

Users specify needed events/columns and desired output format

- Dataset
- Any required preselection

ServiceX

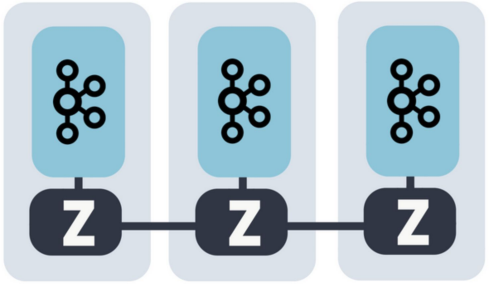
- Queries backend (Rucio) to find data
- Gives unique token to identify request
- Access data from storage through XCache
- Validates request
- Perform data transformations
- Keeps track of data delivered and processed; ensures all events processed



Requirements

- Resilient
- Performant
- Accessible from all the namespaces in the cluster
- Not accessible from outside
- Optional persistency
- Monitoring
- Helm chart

Zookeeper



Apache Zookeeper is a centralized service and is used to maintain naming and configuration data and to provide flexible and robust synchronization to distributed systems.

Controller election

The controller is doing brokering, and has the responsibility to maintain the leader-follower relationship across all the partitions. Whenever a node shuts down, a new controller can be elected and it can also be made sure that at any given time, there is only one controller and all the follower nodes have agreed on that.

Configuration Of Topics

The configuration regarding all the topics including the list of existing topics, the number of partitions for each topic, the location of all the replicas, list of configuration overrides for all topics, which node is the preferred leader, etc.

Access control lists

Access control lists or ACLs for all the topics are also maintained within Zookeeper.

Membership of the cluster

Zookeeper also maintains a list of all the brokers that are functioning at any given moment and are a part of the cluster.

Zookeeper deployment

- Stateful set
  - 5 (different) nodes. 3 is minimal, disruption budget of 1, rolling upgrade 1
  - Antiaffinity to itself
- Services
  - Headless (ClusterIP none) ports 2888 and 3888 (server and leader election)

Currently servicex is letting topics be autocreated. It would probably be better to create them correctly in Zookeeper first.

Kafka

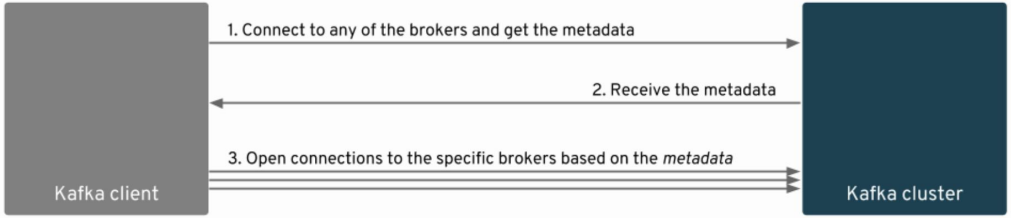
- Stateful set
  - 5 (different) nodes. 3 is minimal, 1 for failover, 1 for rolling upgrade
  - 12GB/4 CPU per pod
  - Antiaffinity to itself, affinity to zookeeper
- Services
  - Headless svc 9092 and each pod 9092.
- Storage
  - volumeClaimTemplate with provisioner (local-volumes or nfs-provisioner)

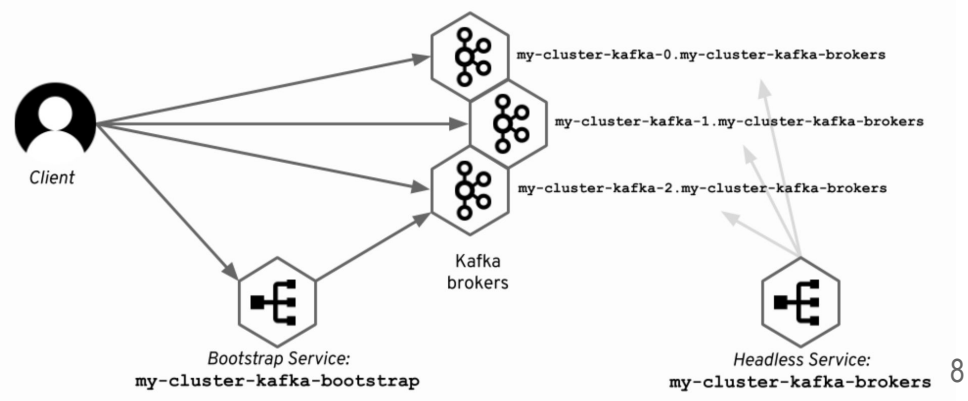
Accessing Kafka

ZOO
kafka-inc-zookeeper.kafka-inc.svc.cluster.local:2181

KAFKA BOOTSTRAP
kafka-inc-headless.kafka-inc.svc.cluster.local:9092

KAFKA BROKERS
kafka-inc-0.kafka-inc.svc.cluster.local:9092,
kafka-inc-1.kafka-inc.svc.cluster.local:9092,
kafka-inc-2.kafka-inc.svc.cluster.local:9092





Monitoring

Prometheus JMX Exporter

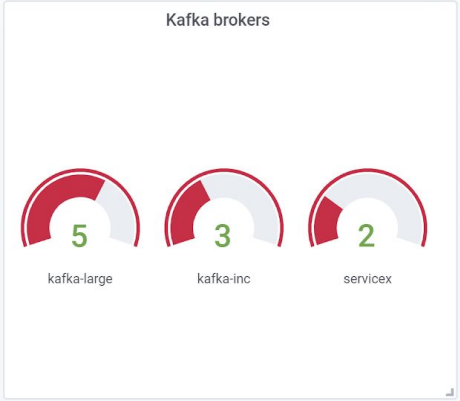
- port 5555
- kafka.controller, kafka.server, java, kafka.network

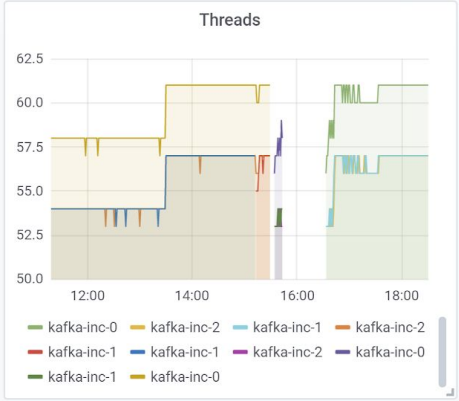
Kafka-exporter

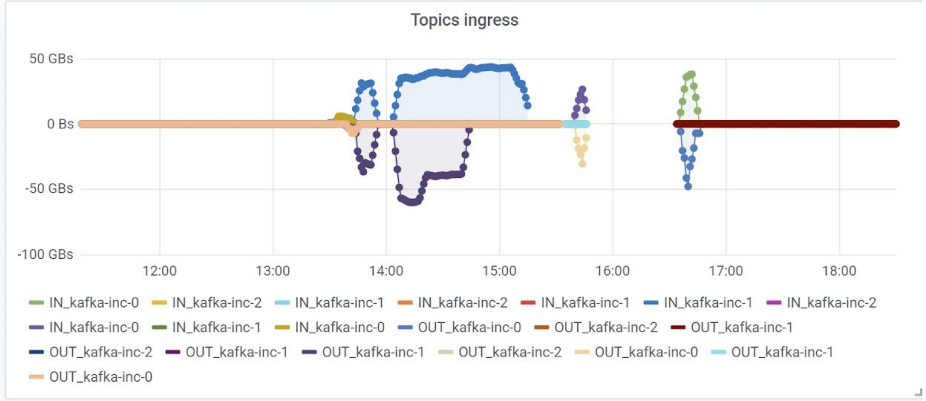
- A regular kafka client running in its own deployment
- Collects data from both kafka and zookeeper
- Port 9308

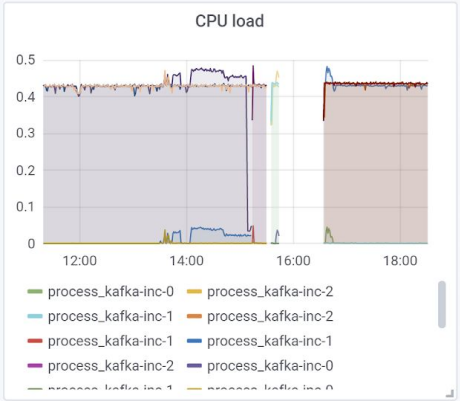
http://monitoring.uc.ssl-hep.org/d/8DLQtIYWk/river2-kafka-exporter

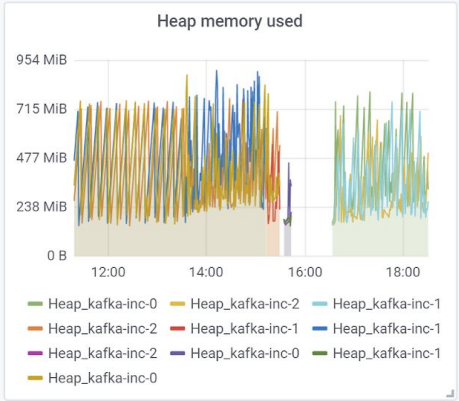
Monitoring


Kafka brokers
5
kafka-large
3
kafka-inc
2
servicex


Threads
62.5
60.0
57.5
55.0
52.5
50.0
12:00
14:00
16:00
18:00
kafka-inc-0 kafka-inc-2 kafka-inc-1 kafka-inc-2
kafka-inc-1 kafka-inc-1 kafka-inc-2 kafka-inc-0
kafka-inc-1 kafka-inc-0


Topics ingress
50 GBs
0 Bs
-50 GBs
-100 GBs
12:00
13:00
14:00
15:00
16:00
17:00
18:00
IN_kafka-inc-0 IN_kafka-inc-2 IN_kafka-inc-1 IN_kafka-inc-2 IN_kafka-inc-1 IN_kafka-inc-1 IN_kafka-inc-2
IN_kafka-inc-0 IN_kafka-inc-1 IN_kafka-inc-0 OUT_kafka-inc-0 OUT_kafka-inc-2 OUT_kafka-inc-1
OUT_kafka-inc-2 OUT_kafka-inc-1 OUT_kafka-inc-1 OUT_kafka-inc-2 OUT_kafka-inc-0 OUT_kafka-inc-1
OUT_kafka-inc-0


CPU load
0.5
0.4
0.3
0.2
0.1
0
12:00
14:00
16:00
18:00
process_kafka-inc-0 process_kafka-inc-2
process_kafka-inc-1 process_kafka-inc-2
process_kafka-inc-1 process_kafka-inc-1
process_kafka-inc-2 process_kafka-inc-0


Heap memory used
954 MiB
715 MiB
477 MiB
238 MiB
0 B
12:00
14:00
16:00
18:00
Heap_kafka-inc-0 Heap_kafka-inc-2 Heap_kafka-inc-1
Heap_kafka-inc-2 Heap_kafka-inc-1 Heap_kafka-inc-1
Heap_kafka-inc-2 Heap_kafka-inc-0 Heap_kafka-inc-1
Heap_kafka-inc-0


Consumer group offsets
220
200
180
160
140
120
12:00
13:00
14:00
15:00
16:00
17:00
18:00
kafka-inc_console-consumer-1045

Deployment

Created straight k8s deployment and tested several helm charts.

This one does everything we need it to do, just slight changes to values.yaml needed.

```
> helm repo add incubator http://storage.googleapis.com/kubernetes-charts-incubator
> kubectl create ns kafka-inc
> helm install kafka-inc incubator/kafka -n kafka-inc -f values.yaml
```

Benchmarks

3 nodes cluster. Local storage. 100kB messages. 10 partitions. NFS storage

Producer

```
Single thread, no replication, async
2965.4 records/sec (282.80 MB/sec), 223.5 ms avg latency, 650.0 max latency.

Single thread, x3 replication, async
2304.8 records/sec (219.80 MB/sec), 282.2 ms avg latency, 1046.0 max latency.

Single thread, x3 replication, sync
1444.6 records/sec (137.77 MB/sec), 470.9 ms avg latency, 1316.0 max latency.

Three Producers, 3x async replication
2330.1 records/sec (222.22 MB/sec), 292.7 ms avg latency, 869.0 max latency.
```

Consumer

```
single thread      452.2135 MB/s, 4741.8022 msg/s
three threads      511.1890 MB/s, 5360.2049 msg/s
```

Producer + Consumer in different pods 1051.4 records/sec (100.27 MB/sec), 632.4 ms avg latency, 782.0 max latency.

Benchmarks

5 nodes cluster. Local storage. 100kB messages.

10 partitions. Local storage

Producer

```
Single thread, no replication, async
6046.0 records/sec (576.59 MB/sec), 8.3 ms avg latency, 73.0 max latency.

Single thread, x3 replication, async
3397.6 records/sec (324.02 MB/sec), 193.1 ms avg latency, 1046.0 max latency.

Single thread, x3 replication, sync
2074.2 records/sec (197.81 MB/sec), 326.8 ms avg latency, 961.0 max latency.

Three Producers, 3x async replication
14101 records sent, 2805.1 records/sec (267.51 MB/sec), 236.7 ms avg latency, 1577.0 max latency.
13738 records sent, 2747.6 records/sec (262.03 MB/sec), 233.0 ms avg latency, 877.0 max latency.
16433 records sent, 3286.6 records/sec (313.43 MB/sec), 206.9 ms avg latency, 878.0 max latency.
Aggregate 842MB/sec
```

Consumer

```
single thread      579.9821 MB/s  6081.5535 mes/s
three threads      779.1776 MB/s  8170.2692 mes/s
Three consumers    691.1397 MB/s  7247.1251 mes/s
                   570.3313 MB/s  5980.3568 mes/s
                   628.8930 MB/s  6594.4213 mes/s
```

Producer + Consumer in different pods 1418.2 records/sec (135.25 MB/sec), 474.3 ms avg latency, 557.0 max latency.